3GPP TSG SA WG3 Security ó SA3#36 **S3-040914**
November 23-26, 2004
Shenzhen, China

ETSI SAGE **SAGE (04) 23**

11 November 2004

Title: Proposed key derivation function for the Generic Bootstrapping Architecture
Source: ETSI SAGE
To: 3GPP SA3
Cc:

Contact Person:
Name: **Steve Babbage**
Tel. Number: + 44 1635 676209
E-mail Address: steve.babbage@vodafone.com

Attachments: None

Introduction

SA3 has asked SAGE to propose a key derivation function for the GBA. So here it is.

Algorithm parameters

We are expecting the function to have the following form:

INPUTS

Ks	256 bits
IMPI	The official liaison from SA3 says that we should consider this an arbitrary bitstream
NAF_Id	The official liaison from SA3 says that we should consider this an arbitrary bitstream
RAND	128 bits

OUTPUT

Ks_NAF	256 bits

However, it will be seen that the function we propose is more flexible than this: it can readily be adapted to accommodate additional input parameters, or alternative sizes of RAND. We also allow for alternate versions of the function to be specified in future.

Assumptions ó SA3 please confirm whether or not these are OK

We have made two assumptions to simplify the form of the function:

1. We assume that both IMPI and NAF_Id will in fact be *octet* strings ó not bit strings of completely arbitrary length. Is this OK? If not, we can adapt the function definition to accommodate arbitrary length strings, but it will be a bit messier and a bit less efficient.
2. We assume that the lengths of IMPI and NAF_Id (and any additional parameters in possible future versions of the function) will have lengths no greater than 65535 octets. Is this OK? Again, we can adapt the function definition to accommodate arbitrarily long strings, but it will be a bit messier.

Basic approach to the function design

We split the design approach into two parts:

1. Concatenate all the parameters apart from Ks into a string S in a collision-free way (no two sets of inputs could give the same S).

2. Construct a MAC on that string using the key Ks.

Two octet coding of string length

In constructing our string S, we will incorporate two-octet representations of the lengths of individual input parameters. So let Pi be an octet string; Li = TwoOctetLengthCoding(Pi) is then defined as follows:

- Express the number of octets in Pi as a number λ in the range $0 \le \lambda \le 65535$ (this is where assumption 2 comes in).
- Li is then a two-octet representation of the number n, with the msb of the first octet of Li equal to the msb of λ, and the lsb of the second octet of Li equal to the lsb of λ.
- As an example, if Pi contains 258 octets then Li will be the two-octet string 0x01 0x02.

Part 1: construct string S from parameters other than Ks

The string S is constructed as follows:

(a) Let P0 be a 16-octet representation of RAND. (We leave it to SA3 to ensure that the bit order is specified unambiguously.) Let L0 = TwoOctetLengthCoding(P0). In this case P0 contains 16 octets, so L0 will be equal to the two octet string 0x00 0x10.

(b) Let P1 = IMPI, and P2 = NAF_Id. Let L1 = TwoOctetLengthCoding(P1), and L2 = TwoOctetLengthCoding(P2).

(c) Let the string S be FC || P0 || L0 || P1 || L1 || P2 || L2,

where FC is a single octet used to distinguish between different instances of the algorithm ó including the two particular instances that SA3 may require, and any future variants.

This construction can be generalised in future to accommodate additional parameters P3, P4 etc.

It can be seen quite easily that this construction is collision free: it is impossible for two different sets of input parameters to yield the same string S. For the purpose of collision-free-ness, Li could come before or after Pi for each i; we have put Li after Pi because we think that may be slightly easier to implement (read in the string Pi, then write down the length of the string you have just read in.)

Part 2: construct a MAC on the string S using the key Ks

The final output Ks_NAF is equal to HMAC-SHA-256 computed on the string S using the key Ks.