

TSG SA #5, Kyongju, 11-13 Oct. 1999

***TSGS#5(99)423***

**Source: S3**

**Subject: 3G TR 33.902 V 0.1.0 - Formal Analysis of the 3G Authentication  
Protocol with Modified Sequence number Management**

**Presented for: Information**

**The attached specification is presented to SA#5 for information. The main body of the specification at present exists in LATEX and PDF. Due to lack of time it had not been possible to perform the necessary adaptations for the specification to become version 1.0.0. In particular, the header and the footnote on page 1 of the main body are not intended to be a part of the document.**

## **3<sup>rd</sup> Generation Partnership Project; Technical Specification Group Services and System Aspects;**

---

**Formal Analysis of the 3G Authentication  
with Modified Sequence number**



**3G TR 33.902 V 0.1.0**

The present document has been developed within the 3<sup>rd</sup> Generation Partnership Project (3GPP<sup>TM</sup>) and may be further elaborated for the purposes of 3GPP.

The present document has not been subject to any approval process by the 3GPP Organisational Partners and shall not be implemented.

This Report is provided for future development work within 3GPP only. The Organisational Partners accept no liability for any use of this Report.

Specifications and reports for implementation of the 3GPP<sup>TM</sup> system should be obtained via the 3GPP Organisational Partners' Publications Offices.

---

---

Reference

DTS/TSGS-\_\_\_\_\_

---

Keywords

Security, Algorithms, Design, Criteria

**3GPP**

---

Postal address

---

3GPP support office address

650 Route des Lucioles - Sophia Antipolis  
Valbonne - FRANCE  
Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

---

Internet

<http://www.3gpp.org>

---

**Copyright Notification**

No part may be reproduced except as authorised by written permission.  
The copyright and the foregoing restrictions extend to reproduction in all media.

© 3GPP 1999  
All rights reserved.

3GPP

<b>Document history</b>		
0.1.0	October 1999	Presented to SA#5 for information

**Source:** Siemens AG**Title:** Formal Analysis of the 3G Authentication Protocol with  
Modified Sequence Number Management\*

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>The TLA Notation</b>	<b>2</b>
<b>3</b>	<b>Notation</b>	<b>3</b>
3.1	Domains, Constants and Functions . . . . .	4
3.2	The variables of the system . . . . .	6
3.3	The Messages: the Transitions of the System . . . . .	7
<b>4</b>	<b>Transitions of the normal System</b>	<b>10</b>
4.1	Changing the Location . . . . .	10
4.2	The Serving Network . . . . .	11
4.3	The Home Environment . . . . .	12
4.4	The Communication Channel . . . . .	14
4.5	The Mobile Station . . . . .	15
<b>5</b>	<b>Definition of Normal Behavior</b>	<b>16</b>
<b>6</b>	<b>Transitions of the incorrect System</b>	<b>18</b>
6.1	Events: more Transitions . . . . .	18
6.2	Changing the Location . . . . .	21
6.3	The Serving Network . . . . .	22
6.4	The Home Environment . . . . .	22
6.5	The Communication Channel . . . . .	23
6.6	The Mobile Station . . . . .	25
<b>7</b>	<b>Definition of Incorrect Behavior</b>	<b>25</b>
<b>8</b>	<b>Theorems</b>	<b>28</b>

---

\*This document is based on work carried out in the EU-sponsored collaborative research project AC336 USECA. Nevertheless, only the author is responsible for the views expressed here.

## 1 Introduction

TLA is a specification language for the compositional specification and verification of distributed programs. The complete protocol was specified in TLA [1], a formal language used mainly for writing specifications of concurrent systems and proving properties of the system. TLA is a state based, first-order temporal logic. The main part of a specification of a system is typically given by a set of events or transitions, each one being a first-order logic predicate that describes the relation between the variables in one state and the next. The specification is completed by expressing the conditions under which the system should start (initial condition) and how a part of the system will eventually respond or act, the so called fairness properties.

The goal of this paper is to give a formal description of the specification of the protocol, of the assumptions, failure models, properties of the system or parts of the system, scenarios and theorems in TLA.

The first theorem presented has the following form: if some conditions on the observable behavior hold during a certain period of time, then at the end of that interval some condition on the internal state of the system holds. We call those conditions on observable behavior *scenarios* (examples: loss of messages, crashes, etc.) and the conditions on the internal state of the system (example: synchronicity of counters) we simply call *(internal) conditions of the system*. Therefore, the first theorem may be expressed as follows: if a certain scenario holds, then a certain condition of the system is true. Other theorems have the following form: if a certain condition of the system is not true at certain time but from that time on a "good" scenario holds, the missing condition will become true again.

## 2 The TLA Notation

The simplest use of TLA is to describe a system as a set of initial conditions,  $\text{Init}$ , together with an "evolution" equation,  $\mathcal{S}$ . This resembles the way a physicist models a continuous system by initial conditions and a differential equation. A state of the system is any set of values (valuation) for some variables  $\{x_1, x_2, \dots, x_n\}$ . Let us call  $x$  the tuple  $x = (x_1, x_2, \dots, x_n)$ . The formula  $\text{Init} = \text{Init}(x)$  is a (first order or higher order) formula of predicate logic on  $x_1, x_2, \dots, x_n$ , stating the conditions in which the system starts. The formula  $\mathcal{S}$  relates two states. Using "primed" versions of the variables  $x_i$ , (that is, using fresh variables  $x'_i$  of the same type as  $x_i$ ),  $\mathcal{S} = \mathcal{S}(x, x')$  is a formula on the set  $\{x_1, x_2, \dots, x_n, x'_1, x'_2, \dots, x'_n\}$ .

In TLA this discrete dynamical system is written as:

$$\mathcal{A} :\Leftrightarrow \text{Init} \wedge \Box[\mathcal{S}]_x$$

The symbol  $\Box$  is read: "box" or "always".  $[\mathcal{S}]_x$  is just an abbreviation of  $\mathcal{S} \vee x' = x$ . Using the convention  $x \uparrow :\Leftrightarrow x' \neq x$ ,  $[\mathcal{S}]_x$  is equivalent to  $x \uparrow \Rightarrow \mathcal{S}$ . We will write

$$\mathcal{A} :\Leftrightarrow \text{Init} \wedge \Box(x \uparrow \Rightarrow \mathcal{S})$$

A sequence  $(\pi_1, \pi_2, \dots, \pi_n, \dots)$  of states satisfies  $\mathcal{A}$  if and only if  $\pi_1$  satisfies  $\text{Init}$ , and each pair  $(\pi_i, \pi_{i+1})$  satisfies  $\mathcal{S}$  or  $\pi_i$  is equal to  $\pi_{i+1}$ . (This is a so called "stutter step"). This is exactly the purpose of writing the subindex  $x$  in  $[\mathcal{S}]_x$  (or the condition  $x \uparrow$  in  $(x \uparrow \Rightarrow \mathcal{S})$ ): allowing stuttering steps. This is quite reasonable: without introducing a global "clock", you may view the system as a set of modules each of which is governed by an equation of the form

$$\mathcal{A}_i :\Leftrightarrow \text{Init}(X_i) \wedge \Box( X_i \uparrow \Rightarrow \mathcal{S}(\tilde{X}_i, \tilde{X}'_i) )$$

where  $X_i, \tilde{X}_i$  are subsets of  $\{x_1, x_2, \dots, x_n\}$  (or subtuples of  $x$ ). Then the whole system is given by the conjunction:

$$\mathcal{A} = \bigwedge_i \text{Init}(X_i) \wedge \bigwedge_i \Box( X_i \uparrow \Rightarrow \mathcal{S}(\tilde{X}_i, \tilde{X}'_i) )$$

We will describe the different versions (scenarios) of the system by formulas  $\mathcal{A}_{normal}$ ,  $\mathcal{A}_{critical}$ ,  $\mathcal{A}_{incorr}$  of this type. More precisely, we will introduce transition relations, ( $\mathcal{N}$  with some indexes) of the form  $\mathcal{N} : \Leftrightarrow (X_i \uparrow \Rightarrow \mathcal{S}(\tilde{X}_i, \tilde{X}'_i))$  to build up (using conjunction or all-quantification) the formulas  $\mathcal{A}$ .

Our modules do not (explicitely) share variables, but communicate via “actions” (events or messages). This may be modeled in TLA by introducing a variable for each message. The variable changes exactly when the message (or event) happens. Therefore if IN is an input message with parameter  $x$  then IN( $x$ ) happens exactly when a suitable variable  $n_{\text{IN}}(x)$  changes:

$$\text{IN}(x) \Leftrightarrow n_{\text{IN}}(x) \uparrow$$

Typically, the next step relations  $\mathcal{N}$  that we will use are of the form<sup>1</sup>:

$$\mathcal{N} : \Leftrightarrow \text{IN}(x) \wedge \text{cond} \Rightarrow \text{OUT}(g(x)) \wedge y' = f(x, y)$$

(if the input IN( $x$ ) happens and the conditions *cond* are true, the module produces the output OUT( $y$ ), with  $y = g(x)$  and changes the local variable  $y$  according to formula  $f$ ) or of the form:

$$\mathcal{N} : \Leftrightarrow \text{OUT}(y) \Rightarrow \text{IN} \quad \text{or} : \quad \mathcal{N} : \Leftrightarrow y \uparrow \Rightarrow \text{IN}$$

(if the output OUT( $y$ ) happens (or  $y$  changes), the only reason is that IN( $x$ ) has also happened. The “dummy” variable “ $x$ ” or “ $y$ ” in those formulas is not a variable of the system: the formula  $\mathcal{N} : \Leftrightarrow \text{IN}(x) \wedge \text{cond} \Rightarrow \dots$  is equivalent to  $\mathcal{N} : \Leftrightarrow \forall_x \text{IN}(x) \wedge \text{cond} \Rightarrow \dots$ .

### 3 Notation

In this section we set our notation for the specification. First, we list the data types or, more properly, domains that will be used in the sequel. This also includes the introduction of constants and functions. Then we introduce the variables of the program that we are interested in. We conclude this section by introducing some notation for the “messages” and “events” of the program.

<sup>1</sup>If  $\mathcal{S}$  is of the form  $\text{cond} \Rightarrow \mathcal{S}_1$ , then  $[\mathcal{S}]_x$  is equivalent to  $x \uparrow \wedge \text{cond} \Rightarrow \mathcal{S}_1$ .

### 3.1 Domains, Constants and Functions

The domains (“data types”) that we need are:

- $\mathbb{B} = \{\underline{0}, \underline{1}\}$
- $\mathbb{N} = \{1, 2, 3, \dots\}$   
= the set of natural numbers
- $\mathcal{SEQ} := \{1, \dots, \text{MaxSEQ}\} \subseteq \mathbb{N}$   
= the set of possible values for the sequence number
- $\mathcal{SN}$  = the identifier set of possible service networks
- $\mathcal{RAND}$  = the set of possible values for the random numbers
- $\mathcal{AUTN}$  = the set of possible values for the variable  $AUTN$
- $\mathcal{AV}$  = the set of admissible authentication vectors
- $\mathcal{CHAL}$  = the set of admissible authentication challenges
- $\mathcal{RESP}_A$  = the set of admissible authentication responses
- $\mathcal{RESP}_J$  = the set of admissible authentication reject responses
- $\mathcal{RESP}_S$  = the set of admissible authentication synchronisation  
fail responses
- $\mathcal{FAIL} := \{\text{Loss, DB, Crash, Steal, Race}\}$

For boolean variables  $x$  or boolean-valued functions  $f$  (i.e., with domain  $\mathbb{B}$ ) we will use the following shorthand, if no confusion arises: instead of writing  $x = 0, f(v) = 0$  or  $x = 1, f(v) = 1$  we will write simply  $\neg x, \neg f(v)$ , or  $x, f(v)$  respectively. The numerical constants that we need are:

- $\text{MaxSEQ}$  = the maximal possible value for the sequence number.
- $\Delta$  = the normal maximal difference between the counters  
 $seq_{MS}$  and  $seq_{HE}$ .
- $N$  = the maximal increment of  $seq_{HE}$  when a batch of  
authentication vectors is produced.
- $N$  is much smaller than  $\Delta$ , we will assume:  $N < \Delta/2$ .

In the specification in [2], (together with the CRs [3], [4], and [5] accepted at 3GPPSA3 London meeting) an  $AV$  is a tuple (“vector”)  $AV = (\text{Rand}, \text{resp}, CK, IK, \text{seq} \oplus AK, MODE, MAC)$ . The only values that we are explicitly interested in are:  $\text{Rand}$ ,  $\text{resp}$  and  $\text{seq}$ . We will not use the components  $CK, IK$  in this specification, and we abstract away from  $MODE$  (say, this is part of the user ID and this is encoded in the secret key  $K$ ). The secret key  $K$  as well as the mac  $MAC$  are only used implicitly to define further functions, as will become clear below. Instead of assuming that  $\text{Rand}$ ,  $\text{resp}$  and  $\text{seq}$  are components of  $AV$ , we assume the existence of functions:  $\underline{Rand}_{AV} : AV \mapsto \text{Rand}$ ,  $\underline{Resp}_{AV} : AV \mapsto \text{resp}$ , and  $\underline{Seq}_{AV} : AV \mapsto \text{seq}$ . The service network,  $\mathbf{SN}$  receives from the home environment the complete  $AV$  and sends to  $\mathbf{MS}$ , the mobile station,  $\text{chall} := \underline{Chall}(AV)$  (in the original specification,  $\text{chall}$  is part of the authentication vector:  $= (\text{Rand}, \text{seq} \oplus AK, MODE, MAC)$ ). The  $\mathbf{MS}$  is able to check the consistency of the challenge  $\text{chall}$  and to calculate the original parameter  $\text{resp}$  of the  $AV$ . Thus we assume the existence of functions:  $\text{cons}_{\text{chall}}$  (to check the consistency of  $\text{chall}$ ) and  $\underline{Resp}_{\text{chall}}$  (to calculate  $\text{resp}$ , given  $\text{chall}$ ). (Those functions, and most of the ones that we will introduce now, depend on the secret key  $K$ ; the function  $\text{cons}_{\text{chall}}$  depends also in particular on the parameter  $MAC$ ). Further, the  $\mathbf{MS}$  is able to calculate the



sequence number  $seq$  with a function  $\underline{Seq}_{MS}$  applied to  $chall$ , and, in case something goes wrong, also the responses  $\underline{RejResp}(chall)$  and  $\underline{SynResp}(la\_chall, chall)$  for an user authentication reject or user authentication synchronisation failure. In this last case, in the response  $SynResp = \underline{SynResp}(la\_chall, chall)$  the current value of the sequence number of the **MS** (= the sequence number of  $la\_chall$ , as will be explained later) is encoded. The home environment **HE** is able to decode this value via a function  $\underline{Seq}_{MS}$  and to verify the freshness of the response  $SynResp$  sent by the mobile station, comparing it to the random number  $Rand$  used by the **SN** in the last challenge. We call this verification function  $verif$ . In the case of a normal response, the **SN** uses a function  $cons_{res}$  to check that the response  $resp$  is consistent with the challenge  $\underline{Chall}(AV)$ . Also, let  $cons_{AV}(AV)$  denote that the authentication vector  $AV$  is consistent. Last, let  $synchr$  be the function used by the **MS** to determine if the two sequence numbers,  $seq_{MS}$ ,  $seq$  are or not “synchronous”, ( $seq_{MS}$  is the sequence number in the **MS**, and  $seq$  is the sequence number of the challenge). What “synchronous” exactly means, is for the most part of the specification, irrelevant, except that 1. if not too many  $AV$ s get lost, then all new challenges are synchronous, and 2. old (for instance, used or lost) challenges become non-synchronous with the passage of time or with successful authentications. But for the statements and proofs of the properties of the system, we will assume that  $synchr(seq_1, seq_2) := (seq_1 < seq_2) \wedge (seq_2 < seq_1 + \Delta)$ .

The (constant, i.e rigid) functions that we need are:

$$\begin{aligned}
\underline{Seq}_{AV} &: AV \rightarrow SEQ \\
\underline{Seq}_{ch} &: CHAL \rightarrow SEQ \\
\underline{Seq}_{MS} &: RESP_S \rightarrow SEQ \\
\underline{Chall} &: AV \rightarrow CHAL \\
\underline{Resp}_{AV} &: AV \rightarrow RESP_A \\
\underline{Resp}_{ch} &: CHAL \rightarrow RESP_A \\
\underline{Rand}_{AV} &: AV \rightarrow RAND \\
\underline{Rand}_{ch} &: CHAL \rightarrow RAND \\
\underline{RejResp} &: CHAL \rightarrow RESP_J \\
\underline{SynResp} &: CHAL \times CHAL \rightarrow RESP_S \\
cons_{AV} &: AV \rightarrow \mathbb{B} \\
cons_{chall} &: CHAL \rightarrow \mathbb{B} \\
cons_{res} &: CHAL \times RESP_A \rightarrow \mathbb{B} \\
synchr &: SEQ \times SEQ \rightarrow \mathbb{B} \\
verif &: RESP_S \times RAND \rightarrow \mathbb{B}
\end{aligned}$$

The function  $\underline{Seq}_{AV}$  defines a transitive, irreflexive relation  $\prec$  in  $AV$ :

$$AV_1 \prec AV_2 :\Leftrightarrow \underline{Seq}_{AV}(AV_1) < \underline{Seq}_{AV}(AV_2)$$

We will assume some properties of these functions. First, the trivial commutations:

$$\begin{aligned}
\underline{Seq}_{ch} \circ \underline{Chall} &= \underline{Seq}_{AV} \\
\underline{Resp}_{ch} \circ \underline{Chall} &= \underline{Resp}_{AV} \\
\underline{Rand}_{ch} \circ \underline{Chall} &= \underline{Rand}_{AV}
\end{aligned}$$

At its proper place, in Sections 4 and 6 (in the definitions of  $\mathcal{N}_{normal}^{HE}$  and  $\mathcal{N}_{critical}^{HE}$ ), it will be assumed that any  $AV$  generated by the **HE** is consistent.

Now, if  $AV$  is consistent, then its challenge is also consistent and also consistent to its corresponding response. One challenge has only one consistent corresponding response.

$$\begin{aligned}
& cons_{AV}(AV) \Rightarrow cons_{chall}(\underline{Chall}(AV)) \wedge cons_{res}(\underline{Chall}(AV), \underline{Resp}(AV)) \\
& cons_{res}(chall, resp) \wedge resp_1 \neq resp \Rightarrow \neg cons_{res}(chall, resp_1) \\
& verif(\text{SynResp}, \text{Rand}) \Leftrightarrow \\
& \quad \exists_{AV, chall_1} \text{SynResp} = \underline{SynResp}(chall_1, \underline{Chall}(AV)) \wedge \text{Rand} = \underline{Rand}(AV) \\
& \underline{Seq}_{MS}(\underline{SynResp}(chall_1, chall_2)) = \underline{Seq}_{ch}(chall_1)
\end{aligned}$$

In the sequel, if no confusion arises, we omit the subscripts, writing  $\underline{Seq}$  instead of  $\underline{Seq}_{ch}$ ,  $\underline{Seq}_{AV}$  or  $\underline{Seq}_{MS}$ ,  $\underline{Resp}$  instead of  $\underline{Resp}_{ch}$  or  $\underline{Resp}_{AV}$ ,  $\underline{Rand}$  instead of  $\underline{Rand}_{ch}$  or  $\underline{Rand}_{AV}$  and  $cons$  instead of  $cons_{AV}$ ,  $cons_{chall}$  or  $cons_{res}$ .

### 3.2 The variables of the system

First let us introduce some rather standard notation for two higher-order constructs that we need:  $\mathcal{AV}^*$  is the set of all *words*  $AV_1 AV_2 AV_3 \dots AV_n$  built with “letters” from  $\mathcal{AV}$ :  $AV_i \in \mathcal{AV}$ . The basic operations in this domain are:  $Head : \mathcal{AV}^* \rightarrow \mathcal{AV}$  that chooses the first letter of the word:  $Head(AV_1 AV_2 AV_3 \dots AV_n) = AV_1$  and  $Tail : \mathcal{AV}^* \rightarrow \mathcal{AV}^*$  is the rest of word:  $Tail(AV_1 AV_2 AV_3 \dots AV_n) = (AV_2 AV_3 \dots AV_n)$ .  $\epsilon$  is the empty word. On the other hand  $\wp(\mathcal{AV})$  is the power-set of  $\mathcal{AV}$ : the set of all subsets of  $\mathcal{AV}$ .

The variables that we will need are:

$$\begin{aligned}
& seq_{HE} : \mathcal{SEQ} \\
& DB : \mathcal{SN} \rightarrow \mathcal{AV}^* \quad \text{called the database of AVs} \\
& la\_chall : \mathcal{CHAL} \quad \text{the last accepted challenge, that is, the} \\
& \quad \text{last challenge accepted by the MS}
\end{aligned}$$

Thus, for  $SN$  in  $\mathcal{SN}$ , the database  $DB(SN)$  of AVs stored in the node  $SN$  is a *word*  $AV_1 AV_2 AV_3 \dots AV_n$  of authentication vectors  $AV_i$ . We will denote by  $Set(\omega)$  the set  $\{AV_1, AV_2, AV_3, \dots, AV_n\}$  of letters in the word  $\omega = AV_1 AV_2 AV_3 \dots AV_n$ . By abuse of notation we will use sometimes  $DB(SN)$  in a context where a *set* is expected instead of a *word*, meaning:  $Set(DB(SN))$ . Thus  $\dots \cup DB(SN)$  is  $\dots \cup Set(DB(SN))$  and  $\dots \subseteq DB(SN)$  is  $\dots \subseteq Set(DB(SN))$ .

The auxiliary variables, defined later in Sections 4 and 6 are:

$$\begin{aligned}
& Gen : \wp(\mathcal{AV}) \quad \text{the set of generated AVs} \\
& Used : \wp(\mathcal{AV}) \quad \text{the set of used (sent) AVs} \\
& Stolen_{HE} : \wp(\mathcal{AV}) \quad \text{the set of AVs that were stolen in the HE} \\
& Stolen_{SN} : \wp(\mathcal{AV}) \quad \text{the set of AVs that were stolen in an SN} \\
& Stolen : \wp(\mathcal{AV}) \quad \text{the set of stolen AVs} \\
& Accepted : \wp(\mathcal{AV}) \quad \text{the set of AVs accepted by the MS} \\
& Successful : \wp(\mathcal{AV}) \quad \text{the set of AVs accepted by the MS and correctly replied} \\
& Lost : \wp(\mathcal{AV}) \quad \text{the set of lost AVs} \\
& last\_SN : \mathcal{SN} \quad \text{in normal behavior, the SN where the user is registered.} \\
& curr\_SN : \wp(\mathcal{SN}) \quad \text{the current set of SNs where the user is registered} \\
& \quad \text{in normal behavior, it is } \{last\_SN\}, \text{ in incorrect behavior} \\
& \quad \text{it may contain no or several SNs.}
\end{aligned}$$

**Definition 3.1.** *The state functions are:*

$$seq_{MS} := \underline{Seq}(la\_chall) : \mathcal{SEQ}$$

**Definition 3.2.**

$$Init :\Leftrightarrow seq_{MS} = 0 \wedge seq_{HE} = 0 \wedge \forall_{SN} DB(SN) = \epsilon \wedge Stolen = \emptyset$$

### 3.3 The Messages: the Transitions of the System

There is a simple way of modeling the occurrence of messages in a purely state-based approach like TLA: for each message or event MESSAGE\_X introduce a variable  $n_x$  that changes exactly when MESSAGE\_X occurs. For convenience, if MESSAGE\_X is a message with parameters of types  $\mathcal{X}_1, \mathcal{X}_2, \dots, \mathcal{X}_n$  we take the variable  $n_x$  to be of type  $\mathcal{X}_1 \times \mathcal{X}_2 \times \dots \times \mathcal{X}_n \rightarrow \mathbb{N}$ .  $n_x(x_1, x_2, \dots, x_n)$  may for instance count (in  $\mathbb{N}$  or modulo a convenient number) how often the message (or event) MESSAGE\_X with parameters  $x_1, x_2, \dots, x_n$  happens. Instead of using the variable  $n_x$  in our specification, we introduce a new predicate, say  $X(x_1, x_2, \dots, x_n)$ , which is an abbreviation:

$$X(x_1, x_2, \dots, x_n) :\Leftrightarrow n_x(x_1, x_2, \dots, x_n) \updownarrow$$

(If  $x$  is a variable, we denote by  $x \updownarrow$  the transition predicate  $x' \neq x$ .)

If MESSAGE\_X is a message between **SN** and **MS**, then this message may get lost. Therefore, we have to distinguish the two events: sending MESSAGE\_X and receiving MESSAGE\_X, which may happen independently of each other.

For instance, USER AUTHENT. REQUEST gives rise to two different events, USER AUTHENT. REQUEST SEND (short:  $UAR_s$ ) and USER AUTHENT. REQUEST RECEIVE ( $UAR_r$ ):

1. Normally, if a  $UAR_s$  happens, then a  $UAR_r$  also happens, with the same parameters.
2. Due to an attack, the  $UAR_r$  may contain different parameters than the corresponding Send.
3. A sent  $UAR_s$  may get lost in the transmission channel, thus producing no  $UAR_r$  event.
4. Due to an attack, the USIM may receive a  $UAR_r$  that was not sent at all by the SN.

The SN receives (or produces) a TIME OUT, if the response to the USER AUTHENT. REQUEST SEND is lost or delayed MVAV (“Move Authentication Vectors”) is the closed action of SEND ID REQ and SEND ID RESP.

<i>Message</i>	<i>Trans. Pred.</i>	<i>Param. Name</i>	<i>Param. Type</i>	<i>Var. Name</i>
AUTHENT. DATA REQUEST (no syn. fail)	ADR <sub>0</sub>	SN	$\mathcal{SN}$	$n_{ADR,0}$
AUTHENT. DATA REQUEST (syn. fail)	ADR <sub>1</sub>	(SynResp, Rand, SN)	( $\mathcal{RESP}_S$ , $\mathcal{RAND}$ , $\mathcal{SN}$ )	$n_{ADR,1}$
AUTHENT. DATA RESPONSE	ADS	(AVs <sub>new</sub> , SN)	( $\mathcal{AV}^*$ , $\mathcal{SN}$ )	$n_{ADS}$
USER AUTHENT. REQUEST	UAR <sub>s</sub>	(chall, SN)	( $\mathcal{CHAL}$ $\mathcal{SN}$ )	$n_{UAR,s}$
	UAR <sub>r</sub>	(chall, SN)	( $\mathcal{CHAL}$ $\mathcal{SN}$ )	$n_{UAR,r}$
USER AUTHENT. RESPONSE	UAS <sub>s</sub>	(resp, SN)	( $\mathcal{RESP}_A$ $\mathcal{SN}$ )	$n_{UAS,s}$
	UAS <sub>r</sub>	(resp, SN)	( $\mathcal{RESP}_A$ $\mathcal{SN}$ )	$n_{UAS,r}$
USER AUTHENT. REJECT	UAJ <sub>s</sub>	(RejResp, SN)	( $\mathcal{RESP}_J$ $\mathcal{SN}$ )	$n_{UAJ,s}$
	UAJ <sub>r</sub>	(RejResp, SN)	( $\mathcal{RESP}_J$ $\mathcal{SN}$ )	$n_{UAJ,r}$
USER AUTHENT. SYNCHRON. FAIL INDICATION	UASF <sub>s</sub>	(SynResp, SN)	( $\mathcal{RESP}_S$ $\mathcal{SN}$ )	$n_{UASF,s}$
	UASF <sub>r</sub>	(SynResp, SN)	( $\mathcal{RESP}_S$ $\mathcal{SN}$ )	$n_{UASF,r}$

Table 1: Messages of the Protocol

<i>Message or Event</i>	<i>Trans. Pred.</i>	<i>Param. Name</i>	<i>Param. Type</i>	<i>Var. Name</i>
LOCATION UPDATE REQUEST	LUR	SN SN <sub>1</sub>	$\mathcal{SN}$ $\mathcal{SN}$	$n_{LUR,s}$
CANCEL LOCATION	CANLOC <sub>s</sub>	SN	$\mathcal{SN}$	$n_{CanLoc,s}$
	CANLOC <sub>r</sub>	SN	$\mathcal{SN}$	$n_{CanLoc,r}$
MOVE AVs (SEND ID REQ and SEND ID RESP)	MvAV	SN SN <sub>1</sub>	$\mathcal{SN}$ $\mathcal{SN}$	$n_{MvAV}$
TIME OUT	TiO	SN	$\mathcal{SN}$	$n_{TiO}$

Table 2: Messages or Events outside of the protocol

**Definition 3.3.** (The Messages)

$$\begin{aligned}
\text{ADR}_0(\text{SN}) &:\Leftrightarrow n_{\text{ADR},0}(\text{SN})\Downarrow \\
\text{ADR}_1(\text{SynResp}, \text{Rand}, \text{SN}) &:\Leftrightarrow n_{\text{ADR},1}(\text{SynResp}, \text{Rand}, \text{SN})\Downarrow \\
\text{ADS}(\text{AVs\_new}, \text{SN}) &:\Leftrightarrow n_{\text{ADS}}(\text{AVs\_new}, \text{SN})\Downarrow \\
\text{UAR}_s(\text{chall}, \text{SN}) &:\Leftrightarrow n_{\text{UAR},s}(\text{chall}, \text{SN})\Downarrow \\
\text{UAR}_r(\text{chall}, \text{SN}) &:\Leftrightarrow n_{\text{UAR},r}(\text{chall}, \text{SN})\Downarrow \\
\text{UAS}_s(\text{resp}, \text{SN}) &:\Leftrightarrow n_{\text{UAS},s}(\text{resp}, \text{SN})\Downarrow \\
\text{UAS}_r(\text{resp}, \text{SN}) &:\Leftrightarrow n_{\text{UAS},r}(\text{resp}, \text{SN})\Downarrow \\
\text{UAJ}_s(\text{RejResp}, \text{SN}) &:\Leftrightarrow n_{\text{UAJ},s}(\text{RejResp}, \text{SN})\Downarrow \\
\text{UAJ}_r(\text{RejResp}, \text{SN}) &:\Leftrightarrow n_{\text{UAJ},r}(\text{RejResp}, \text{SN})\Downarrow \\
\text{UASF}_s(\text{SynResp}, \text{SN}) &:\Leftrightarrow n_{\text{UASF},s}(\text{SynResp}, \text{SN})\Downarrow \\
\text{UASF}_r(\text{SynResp}, \text{SN}) &:\Leftrightarrow n_{\text{UASF},r}(\text{SynResp}, \text{SN})\Downarrow \\
\text{LUR}(\text{SN}, \text{SN}_1) &:\Leftrightarrow n_{\text{LUR}}(\text{SN}, \text{SN}_1)\Downarrow \\
\text{CANLOC}_s(\text{SN}) &:\Leftrightarrow n_{\text{canLoc},s}(\text{SN})\Downarrow \\
\text{CANLOC}_r(\text{SN}) &:\Leftrightarrow n_{\text{canLoc},r}(\text{SN})\Downarrow \\
\text{MvAV}(\text{SN}, \text{SN}_1) &:\Leftrightarrow n_{\text{MvAV}}(\text{SN}, \text{SN}_1)\Downarrow \\
\text{TIO}(\text{SN}) &:\Leftrightarrow n_{\text{TIO}}(\text{SN})\Downarrow
\end{aligned}$$

By abuse of notation, we will use the predicates  $\text{ADR}_0, \text{ADR}_1, \text{ADS}, \dots$  *without parameters* to intend an implicit existential quantification:

**Convention 3.1.**

$$\begin{aligned}
\text{ADR}_0 &:\Leftrightarrow \exists_{\text{SN}} \text{ADR}_0(\text{SN}) \\
\text{ADR}_1 &:\Leftrightarrow \exists_{\text{SynResp}, \text{Rand}, \text{SN}} \text{ADR}_1(\text{SynResp}, \text{Rand}, \text{SN}) \\
\text{ADS} &:\Leftrightarrow \exists_{\text{AVs\_new}, \text{SN}} \text{ADS}(\text{AVs\_new}, \text{SN}) \\
&\dots
\end{aligned}$$

Further we will use the predicates  $\text{ADR}_1(\text{SN}), \text{ADS}(\text{SN}), \dots$  *without other parameters* to intend an implicit existential quantification over the non mentioned parameters:

**Convention 3.2.**

$$\begin{aligned}
\text{ADR}_1(\text{SN}) &:\Leftrightarrow \exists_{\text{SynResp}, \text{Rand}} \text{ADR}_1(\text{SynResp}, \text{Rand}, \text{SN}) \\
\text{ADS}(\text{SN}) &:\Leftrightarrow \exists_{\text{AVs\_new}} \text{ADS}(\text{AVs\_new}, \text{SN}) \\
&\dots
\end{aligned}$$

We will not really use the variables  $n_{\text{ADR},0}(\text{SN})$ , etc. any further. Their only purpose is to accommodate to TLA. One note on TLA: instead of using the conventional syntax  $[\mathcal{A}]_x$  of TLA, we prefer the equivalent more readable form:  $x\Downarrow \Rightarrow \mathcal{A}$ .

Notice also that  $x\Downarrow \wedge \mathcal{P} \Rightarrow \mathcal{A}$  is  $[\mathcal{P} \Rightarrow \mathcal{A}]_x$

It is impossible that a message MESSAGE\_X happens at the same time with two different sets of parameters:  $X(x_1, x_2, \dots, x_n) \wedge X(y_1, y_2, \dots, y_n) \wedge (x_1, x_2, \dots, x_n) \neq$

$(y_1, y_2, \dots, y_n)$ :

$$\begin{aligned}
\mathcal{A}_{normal}^{interleave} &:\Leftrightarrow \\
&\Box( \forall_{SN, SynResp, Rand, AVs\_new, chall, resp, RejResp, SynResp} \\
&\quad \forall_{SN_1, SynResp_1, Rand_1, AVs\_new_1, chall_1, resp_1, RejResp_1, SynResp_1} \\
&\quad \wedge \text{ADR}_0(SN) \wedge \text{ADR}_0(SN_1) \Rightarrow SN = SN_1 \\
&\quad \wedge \text{ADR}_1(SynResp, Rand, SN) \wedge \text{ADR}_1(SynResp_1, Rand_1, SN_1) \\
&\quad \quad \Rightarrow SynResp = SynResp_1 \wedge Rand = Rand_1 \wedge SN = SN_1 \\
&\quad \wedge \text{ADS}(AVs\_new, SN) \wedge \text{ADS}(AVs\_new_1, SN_1) \\
&\quad \quad \Rightarrow AVs\_new = AVs\_new_1 \wedge SN = SN_1 \\
&\quad \wedge \text{UAR}_s(chall, SN) \wedge \text{UAR}_s(chall_1, SN_1) \\
&\quad \quad \Rightarrow chall = chall_1 \wedge SN = SN_1 \\
&\quad \wedge \text{UAR}_r(chall, SN) \wedge \text{UAR}_r(chall_1, SN_1) \\
&\quad \quad \Rightarrow chall = chall_1 \wedge SN = SN_1 \\
&\quad \wedge \text{UAS}_s(resp, SN) \wedge \text{UAS}_s(resp_1, SN_1) \\
&\quad \quad \Rightarrow resp = resp_1 \wedge SN = SN_1 \\
&\quad \wedge \text{UAS}_r(resp, SN) \wedge \text{UAS}_r(resp_1, SN_1) \\
&\quad \quad \Rightarrow resp = resp_1 \wedge SN = SN_1 \\
&\quad \wedge \text{UAJ}_s(RejResp, SN) \wedge \text{UAJ}_s(RejResp_1, SN_1) \\
&\quad \quad \Rightarrow RejResp = RejResp_1 \wedge SN = SN_1 \\
&\quad \wedge \text{UAJ}_r(RejResp, SN) \wedge \text{UAJ}_r(RejResp_1, SN_1) \\
&\quad \quad \Rightarrow RejResp = RejResp_1 \wedge SN = SN_1 \\
&\quad \wedge \text{UASF}_s(SynResp, SN) \wedge \text{UASF}_s(SynResp_1, SN_1) \\
&\quad \quad \Rightarrow SynResp = SynResp_1 \wedge SN = SN_1 \\
&\quad \wedge \text{UASF}_r(SynResp, SN) \wedge \text{UASF}_r(SynResp_1, SN_1) \\
&\quad \quad \Rightarrow SynResp = SynResp_1 \wedge SN = SN_1 \\
&\quad \wedge \text{UAS}_s \Rightarrow \neg \text{UAJ}_s \wedge \neg \text{UASF}_s \\
&\quad \wedge \text{UAJ}_s \Rightarrow \neg \text{UAS}_s \wedge \neg \text{UASF}_s \\
&\quad \wedge \text{UASF}_s \Rightarrow \neg \text{UAS}_s \wedge \neg \text{UAJ}_s )
\end{aligned}$$

## 4 Transitions of the normal System

### 4.1 Changing the Location

Let us first define the auxiliary variable  $curr\_SN : \wp(\mathcal{SN})$  in the following way:

$$\begin{aligned}
\mathcal{A}_{normal}^{CurrSN} &:\Leftrightarrow \\
&\wedge \exists_{SN} curr\_SN = \{SN\} \\
&\Box( \wedge curr\_SN \uparrow \Rightarrow \text{LUR} \vee \text{CANLOC} \\
&\quad \wedge \text{LUR}(SN, SN_1) \wedge \text{CANLOC}(SN_2) \Rightarrow \\
&\quad \quad curr\_SN' = (curr\_SN \setminus \{SN_2\}) \cup \{SN_1\} \\
&\quad \wedge \text{LUR}(SN, SN_1) \wedge \neg \text{CANLOC} \Rightarrow \\
&\quad \quad curr\_SN' = curr\_SN \cup \{SN_1\} \\
&\quad \wedge \text{CANLOC}(SN) \wedge \neg \text{LUR} \Rightarrow \\
&\quad \quad curr\_SN' = (curr\_SN \setminus \{SN\}) )
\end{aligned}$$

The intuition is that, under ideal<sup>2</sup> behavior, a  $LUR(SN, SN_1)$  implies a  $CANLOC(SN)$ . In this case, in the set of current SNs the old SN,  $SN$ , is replaced by the new one,  $SN_1$ :  $curr\_SN' = (curr\_SN \setminus \{SN\}) \cup \{SN_1\}$ . Thus  $curr\_SN$  would always be a singleton. We will allow *in normal conditions* that a  $CANLOC(SN)$  occurs without a  $LUR(SN, SN_1)$ , thus  $curr\_SN$  is always a singleton or empty. In more critical situations we will allow  $curr\_SN$  to be an arbitrary (finite) set: it is the set of SNs for which a  $LUR(SN, SN_1)$  has not been followed by a  $CANLOC(SN)$ .

With this definition, our specification of the location update step may be written as:

$$\begin{aligned}
\mathcal{N}_{normal}^{LU} : \Leftrightarrow & \\
& \wedge curr\_SN' = \emptyset \vee \exists_{SN_0} curr\_SN' = \{SN_0\} \\
& \wedge LUR(SN, SN_1) \Rightarrow MvAV(SN, SN_1) \vee DB'(SN_1) = \epsilon \\
& \wedge MvAV(SN, SN_1) \Rightarrow LUR(SN, SN_1) \\
& \wedge ADS \Rightarrow curr\_SN' = curr\_SN \\
& \wedge MvAV(SN, SN_1) \Rightarrow \wedge DB'(SN_1) = DB(SN) \\
& \quad \wedge DB'(SN) = \epsilon \\
& \quad \wedge \forall_{SN_2} (SN_2 \neq SN \wedge SN_2 \neq SN_1 \Rightarrow \\
& \quad \quad DB'(SN_2) = DB'(SN_2))
\end{aligned}$$

The five points in this requirement are:

first, as explained before, a  $CANLOC$  may happen without a  $LUR$ , (resulting in  $curr\_SN' = \emptyset$ ). On the other hand, a  $LUR$  can happen without a  $CANLOC$  only if  $curr\_SN = \emptyset$ , (resulting in  $curr\_SN'$  being a singleton, that is,  $\exists_{SN_0} curr\_SN' = \{SN_0\}$ ), or in other words, if  $CANLOC$  has already happened. In any case, both a  $LUR$  and its corresponding  $CANLOC$  can happen simultaneously.

Second, a  $LUR$  may trigger a  $MvAV$ , but not necessarily; if no  $MvAV$  happens, then  $DB'(SN_1) = \epsilon$ . If  $MvAV$  happens, then  $DB'(SN_1) = DB(SN)$ . Thus, in any case, any old existing AVs are to be discarded.

Third, a  $MvAV$  is always produced by a  $LUR$ .

Fourth, no race condition happens. This type of race condition will be discussed later in Section 6.

And last, when a  $MvAV$  happens, the AVs of the old  $SN$  are moved to the new  $SN$ .

Let us now define the auxiliary variable  $last\_SN : \mathcal{SN}$  in the following way:

$$\begin{aligned}
\mathcal{A}_{normal}^{LastSN} : \Leftrightarrow & \\
& \wedge curr\_SN = \{last\_SN\} \\
& \wedge \square( \wedge last\_SN \uparrow \Rightarrow curr\_SN \downarrow \\
& \quad \wedge \forall_{SN_0} (curr\_SN \downarrow \wedge curr\_SN' = \{SN_0\} \Rightarrow last\_SN' = SN_0) \\
& \quad \wedge (curr\_SN \downarrow \wedge \forall_{SN_0} curr\_SN' \neq \{SN_0\} \Rightarrow last\_SN' = last\_SN )
\end{aligned}$$

Notice that we in the context of  $\square \mathcal{N}_{normal}^{LU}$  the predicate  $\forall_{SN_0} curr\_SN' \neq \{SN_0\}$  in the last line of the last formula, is equivalent to  $curr\_SN' = \emptyset$ . Thus, in this context, even if  $curr\_SN$  is empty,  $last\_SN$  is the last SN where the user was registered.

## 4.2 The Serving Network

Let us consider first the **AUTHENT. DATA REQUEST** with the synchronisation flag turned off ( $ADR_0$ , that is, no synchronisation fail has happened). The reason for

<sup>2</sup>We do not model explicitly “ideal behavior”. We let our best scenario, “normal behavior”, to contain already “normal” errors, like an  $LUR$  without a  $CANLOC$ .

issuing this message is that the SN has only few or no authentication vectors left. For simplicity, we assume the last case, i.e.,  $ADR_0 \Rightarrow DB(SN) = \epsilon$ .

On the other hand, the only reason for asking AUTHENT. DATA REQUEST with the synchronisation flag turned on ( $ADR_1$ ), is that a synchronisation fail has happened, or, more precisely, a UASF has been obtained as response to a  $UAR_s$ .

The SN reacts to AUTHENT. DATA RESPONSE by updating the database of AVs ( $DB(SN)$ ).

When the SN sends a USER AUTHENT. REQUEST SEND, it updates the database of AVs by deleting the first AV in the list  $DB(SN)$ . (This AV is now the “current AV”, and the values are used to compare with the expected response,  $UAS_r(\text{resp}, SN)$ ). If, sending a user authentication request, no answer ( $UAS_r$  or  $UAJ_r$  or  $UASF_r$ ) is received, then a TiO happens.

$$\begin{aligned}
\mathcal{N}_{normal}^{SN} &:\Leftrightarrow \\
&\wedge ADR_0(SN) \Rightarrow DB(SN) = \epsilon \wedge SN \in curr\_SN \\
&\wedge ADR_1(\text{SynResp}, \text{Rand}, SN) \Rightarrow \\
&\quad \wedge UASF_r(\text{SynResp}, SN) \\
&\quad \wedge SN \in curr\_SN \\
&\quad \wedge \exists_{AV} (UAR_s(\text{Chall}(AV), SN) \wedge \text{Rand} = \underline{Rand}(AV)) \\
&\wedge ADS(\text{AVs\_new}, SN) \Rightarrow \wedge \text{AVs\_new} \neq \epsilon \Rightarrow DB'(SN) = \text{AVs\_new} \\
&\quad \wedge \text{AVs\_new} = \epsilon \Rightarrow DB'(SN) = DB(SN) \\
&\wedge UAR_s(\text{chall}, SN) \Rightarrow \wedge SN \in curr\_SN \wedge DB(SN) \neq \epsilon \\
&\quad \wedge DB'(SN) = \text{Tail}(DB(SN)) \\
&\quad \wedge \text{chall} = \underline{Chall}(\text{Head}(DB(SN))) \\
&\wedge UAR_s(\text{chall}, SN) \Rightarrow \vee \exists_{\text{resp}} UAS_r(\text{resp}, SN) \\
&\quad \vee \exists_{\text{RejResp}} UAJ_r(\text{RejResp}, SN) \\
&\quad \vee \exists_{\text{SynResp}} UASF_r(\text{SynResp}, SN) \\
&\quad \vee \text{TiO} \\
&\wedge DB(SN) \neq \epsilon \Rightarrow \vee \exists_{\text{AVs\_new}, SN} ADS(\text{AVs\_new}, SN) \wedge \text{AVs\_new} \neq \epsilon \\
&\quad \vee \exists_{\text{chall}, SN} UAR_s(\text{chall}, SN) \\
&\quad \vee \exists_{SN_1} M \vee AV(SN, SN_1) \\
&\quad \vee \exists_{SN_1} M \vee AV(SN_1, SN) \\
&\wedge [ \wedge UASF_r(\text{SynResp}, SN) \\
&\quad \wedge SN \in curr\_SN \\
&\quad \wedge UAR_s(\underline{Chall}(AV), SN) \\
&\quad \wedge \text{Rand} = \underline{Rand}(AV) ] \Rightarrow ADR_1(\text{SynResp}, \text{Rand}, SN)
\end{aligned}$$

### 4.3 The Home Environment

In the next definition we use a new (bound) variable  $seq_{he}$  that contains a temporary value for  $seq_{HE}$ . The reader may understand the specification of the step  $\mathcal{N}_{normal}^{HE}$  as the sequential composition of two “micro-steps”:

$$\begin{aligned}
\mathcal{N}_{normal}^{HE}(seq_{HE}, seq_{HE}') &= \exists_{seq_{he}} \\
&\quad ( \mathcal{N}_{normal}^{1, HE}(seq_{HE}, seq_{he}) \wedge \mathcal{N}_{normal}^{2, HE}(seq_{he}, seq_{HE}') )
\end{aligned}$$



Notice, by the way, that if  $\mathcal{N}_{normal}^{HE} \wedge (ADR_0 \vee ADR_1)$  the value of  $seq_{HE}$  uniquely determines the value of  $seq_{he}$ .

Recall that  $AVs\_new$  is a word (or ordered sequence) of AVs. We write  $AV_1 \ll_{AVs\_new} AV_2$  iff both  $AV_1$  and  $AV_2$  appear in  $AVs\_new$  and  $AV_1$  appears (anywhere) *left* of  $AV_2$ .

Here we write  $\ll$  instead of  $\ll_{AVs\_new}$ .

$$\begin{aligned}
\mathcal{N}_{normal}^{HE} &:\Leftrightarrow \exists_{seq_{he}} [ \\
&\wedge ADS(AVs\_new, SN) \Rightarrow \vee ADR_0(SN) \\
&\quad \vee \exists_{SynResp, Rand} ADR_1(SynResp, Rand, SN) \\
&\wedge ADR_0 \vee ADR_1 \Rightarrow ADS \\
&\wedge ADR_0 \Rightarrow seq_{he} = seq_{HE} \\
&\wedge ADR_1(SynResp, Rand, SN) \Rightarrow \\
&\quad \wedge [verif(SynResp, Rand) \wedge \\
&\quad \quad \neg synchr_1(\underline{Seq}_{MS}(SynResp), seq_{HE})] \Rightarrow seq_{he} = \underline{Seq}_{MS}(SynResp) \\
&\quad \wedge [\neg verif(SynResp, Rand) \vee \\
&\quad \quad synchr_1(\underline{Seq}_{MS}(SynResp), seq_{HE})] \Rightarrow seq_{he} = seq_{HE} \\
&\wedge ADS(AVs\_new, SN) \Rightarrow \\
&\quad \wedge AVs\_new \neq \epsilon \\
&\quad \wedge AV \in AVs\_new \Rightarrow cons(AV) \\
&\quad \wedge \forall_{AV_1, AV_2 \in AVs\_new} (AV_1 \prec AV_2 \Leftrightarrow AV_1 \ll AV_2) \\
&\quad \wedge \forall_{AV \in AVs\_new} (seq_{he} < \underline{Seq}(AV) \leq seq'_{HE}) \\
&\quad \wedge \forall_{i \in \mathbb{N}} \exists_{AV \in AVs\_new} (seq_{he} < i \leq seq'_{HE} \Rightarrow \underline{Seq}(AV) = i) \\
&\quad \wedge seq'_{HE} - seq_{he} \leq N \\
&\wedge seq_{HE} \uparrow \Rightarrow \exists_{AVs\_new, SN} ADS(AVs\_new, SN) \wedge AVs\_new \neq \epsilon ]
\end{aligned}$$

In this specification we have used a new function  $synchr_1$ , instead of our old  $synchr$ . The reason is the following: we want not only that the current value of  $seq_{HE}$  is in the correct range:  $synchr(seq_{MS}, seq_{HE})$ , but also that the new value of  $seq_{HE}$  is also in the correct range (else, although  $seq_{HE}$  is in the correct range the HE could generate AVs which are outside of this range). Thus we also want:  $synchr(seq_{MS}, seq_{HE}')$ , or in other words:  $synchr(seq_{MS}, seq_{HE} + N)$ . The definition of  $synchr_1(x, y)$  is therefore:

$$synchr_1(x, y) := synchr(x, y) \wedge synchr(x, y + N)$$

This specification says nothing about where do the AVs in  $AVs\_new$  come from. They can be generated in the moment in which they are sent (through an Authentication Data Response), or “pre-generated” and kept in an internal HE Database.

It is important for our proofs that, in normal behavior, when  $ADR_0$  or  $ADR_1(SynResp, Rand, SN)$  with  $verif(SynResp, Rand) \wedge \neg synchr_1(\underline{Seq}_{MS}(SynResp), seq_{HE})$ , the parameter  $AVs\_new$  in  $ADS(AVs\_new, SN)$  is not the empty word. In other cases it could be empty without changing our properties or proofs. For the meantime, we follow the original specification, in which  $AVs\_new$  is never  $\epsilon$ . Later, for the incorrect system, we will weaken this assumption.

#### 4.4 The Communication Channel

Recall from Convention 3.2 that for instance  $UAR_s(SN)$  means  $\exists_{\text{resp}} UAR_s(\text{resp}, SN)$ , i.e. an implicit existential quantification over the non mentioned parameters. Let us say that the mobile station communicates with the SN if in any one of the two direction a message is sent or received.

$$\begin{aligned} Comm(SN) : \Leftrightarrow & \vee UAR_s(SN) \vee UAR_r(SN) \\ & \vee UAS_s(SN) \vee UAS_r(SN) \\ & \vee UAJ_s(SN) \vee UAJ_r(SN) \\ & \vee UASF_s(SN) \vee UASF_r(SN) \end{aligned}$$

We will assume that the MS communicates at the same time with only one SN:  $Comm(SN) \wedge Comm(SN_1) \Rightarrow SN = SN_1$

The message USER AUTHENT. REQUEST may be received correctly ( $OKR$ ), or it may be corrupted during the transmission ( $CorrR$ ), or it may get lost ( $LossR$ ):

$$\begin{aligned} OKR : \Leftrightarrow & \exists_{\text{chall}, SN} \wedge UAR_s(\text{chall}, SN) \\ & \wedge UAR_r(\text{chall}, SN) \\ CorrR : \Leftrightarrow & \exists_{\text{chall}, SN} \wedge UAR_s(\text{chall}, SN) \\ & \wedge \exists_{\text{chall}_1} \neg cons(\text{chall}_1) \wedge UAR_r(\text{chall}_1, SN) \\ LossR : \Leftrightarrow & \exists_{\text{chall}, SN} \wedge UAR_s(\text{chall}, SN) \\ & \wedge \neg UAR_r(SN) \end{aligned}$$

We will assume that during each step of the normal system,  $UAR_s \Rightarrow OKR \vee CorrR \vee LossR$ . In other words, our assumption is that the challenge  $\text{chall}$  in  $UAR_s(\text{chall}, SN)$  can not be replaced during the communication by another challenge  $\text{chall}_1$  (in  $UAR_r(\text{chall}_1, SN)$ ) which is also consistent. This sort of situation will be discussed later in Section 6.

On the other direction, the message USER AUTHENT. RESPONSE may be received correctly ( $OKS$ ), or it may be corrupted during the transmission ( $CorrS$ ), or it may get lost ( $LossS$ ). As in the other directions, our assumption is that the response can not be replaced during the communication by another consistent or verifiable response. For the case of a normal response, this amounts to nothing, because there is only one consistent response. For the case of a synchronisation fail, we have to state explicitly, that if  $UAR_s(\underline{Chall}(AV), SN)$  happens in the same step, then the corrupted response is not verifiable (with respect to the random number of this  $AV$ ). This is exactly what we ask in the Assumption 4.1. Another possibility would be to impose  $\mathcal{N}_{critical}^{Ass_3}$ , discussed later in Assumption 6.3.

$$\begin{aligned} OKS : \Leftrightarrow & \vee \exists_{\text{resp}, SN} \wedge UAS_s(\text{resp}, SN) \\ & \wedge UAS_r(\text{resp}, SN) \\ & \vee \exists_{\text{RejResp}, SN} \wedge UAJ_s(\text{RejResp}, SN) \\ & \wedge UAJ_r(\text{RejResp}, SN) \\ & \vee \exists_{\text{SynResp}, SN} \wedge UASF_s(\text{SynResp}, SN) \\ & \wedge UASF_r(\text{SynResp}, SN) \end{aligned}$$

$$\begin{aligned}
CorrS &:\Leftrightarrow \vee \exists_{resp, SN} \wedge UAS_s(resp, SN) \\
&\quad \wedge \exists_{resp_1} resp_1 \neq resp \wedge UAS_r(resp_1, SN) \\
&\vee \exists_{RejResp, SN} \\
&\quad \wedge UAJ_s(RejResp, SN) \\
&\quad \wedge \exists_{RejResp_1} RejResp_1 \neq RejResp \wedge UAJ_r(RejResp_1, SN) \\
&\vee \exists_{SynResp, SN} \\
&\quad \wedge UASF_s(SynResp, SN) \\
&\quad \wedge \exists_{SynResp_1} \wedge SynResp_1 \neq SynResp \\
&\quad \quad \wedge UASF_r(SynResp_1, SN) \\
\\
LossS &:\Leftrightarrow \wedge \vee UAS_s(SN) \\
&\quad \vee UAJ_s(SN) \\
&\quad \vee UASF_s(SN) \\
&\quad \wedge \neg UAS_r(SN) \\
&\quad \wedge \neg UAJ_r(SN) \\
&\quad \wedge \neg UASF_r(SN)
\end{aligned}$$

The corruption of messages does not generate consistent fail synchronisation responses to the challenge.

**Assumption 4.1.**

$$\begin{aligned}
\mathcal{N}_{normal}^{Ass} &:\Leftrightarrow \\
&[\wedge UAR_s(chall, SN) \wedge UASF_r(SynResp, SN) \\
&\quad \wedge \neg UASF_s(SynResp, SN)] \\
&\Rightarrow \neg \text{verif}(SynResp, \underline{Rand}(AV))
\end{aligned}$$

We will assume that during each non-stutter step of the communication channel, either the channel is OK or there is a corruption or a loss of a challenge/response:

$$\begin{aligned}
\mathcal{N}_{normal}^{CC} &:\Leftrightarrow \\
&\wedge \forall_{SN, SN_1} Comm(SN) \wedge Comm(SN_1) \Rightarrow SN = SN_1 \\
&\wedge UAR_s \Rightarrow OKR \vee CorrR \vee LossR \\
&\wedge UAR_r \Rightarrow OKR \vee CorrR \vee LossR \\
&\wedge UAS_s \vee UAJ_s \vee UASF_s \Rightarrow OKS \vee CorrS \vee LossS \\
&\wedge UAS_r \vee UAJ_r \vee UASF_r \Rightarrow OKS \vee CorrS \vee LossS \\
&\wedge \mathcal{N}_{normal}^{Ass}
\end{aligned}$$

## 4.5 The Mobile Station

**Definition 4.1.** *The system is during the lifetime of the USIM if the number of User Authentication Responses is less than  $SQNmax/\Delta$ :*

$$Lifetime :\Leftrightarrow n_{UAS, s} \leq SQNmax/\Delta$$

When the mobile station receives a USER AUTHENT. REQUEST, if the challenge is consistent and synchronous. it updates the variable  $la\_chall$  and sends the corresponding response, USER AUTHENT. RESPONSE. But if the challenge is not consistent, it sends a USER AUTHENT. REJECT, and if the challenge is not synchronous,



(3.) between two serving networks (during a MVAV). From the point of view of the **HE** and the **MS**, at least, it is equivalent to loose the AV in any one of those three situations or to loose it in the communication Channel from the SN to the USIM.

**Definition 5.3.** An AV copy is used if its challenge was sent in an authentication request. More precisely, the variable *Used*, of type  $\wp(\mathcal{AV})$  is defined by:

$$\begin{aligned} \mathcal{A}_{normal}^{Used} &:\Leftrightarrow \\ &\wedge Used = \emptyset \\ &\wedge \Box( \wedge Used \Downarrow \Rightarrow UAR_s \\ &\quad \wedge UAR_s(\underline{Chall}(\mathcal{AV}), SN) \Rightarrow Used' = Used \cup \{\mathcal{AV}\} \end{aligned}$$

**Definition 5.4.** An AV copy is accepted if its challenge was accepted by the mobile station. Accepted, of type  $\wp(\mathcal{AV})$  is defined by:

$$\begin{aligned} \mathcal{A}_{normal}^{Accepted} &:\Leftrightarrow \\ &\wedge Accepted = \emptyset \\ &\wedge \Box( \wedge Accepted \Downarrow \Rightarrow la\_chall \Downarrow \\ &\quad \wedge la\_chall \Downarrow \Rightarrow Accepted' = Accepted \cup \\ &\quad \quad \quad \{\mathcal{AV} \in Gen' \mid \underline{Chall}(\mathcal{AV}) = la\_chall'\} ) \end{aligned}$$

**Definition 5.5.** An unused AV copy is usable if its location is the current SN where the user is registered (or where the user was last registered), that is, it is an element of  $DB(last\_SN)$ .

**Definition 5.6.** The sequence numbers  $seq_{MS}$  in the USIM and  $seq_{HE}$  in the home environment are called synchronous iff  $synchron(seq_{MS}, seq_{HE} + 1)$ . In this case we call the system weakly-synchronous:

$$WeakSynchron :\Leftrightarrow synchron(seq_{MS}, seq_{HE} + 1)$$

**Definition 5.7.** An unused AV copy is called synchronous (with respect to  $seq_{MS}$ ) if  $synchron(seq_{MS}, \underline{Seq}(\mathcal{AV}))$

**Definition 5.8.** The system is strongly-synchronous if it is weakly-synchronous and all usable AV copies are also synchronous (w.r. to  $seq_{MS}$ ).

$$StrongSynchron :\Leftrightarrow WeakSynchron \wedge \forall_{\mathcal{AV} \in DB(last\_SN)} synchron(seq_{MS}, \underline{Seq}(\mathcal{AV}))$$

**Definition 5.9.** Let  $\mathcal{A} \subseteq \mathcal{AV}$ .  $\mathcal{A}$  is said to have no  $m$  consecutive AVs or to be interrupted each  $m$  elements iff between any two elements of  $\mathcal{A}$  whose sequence numbers differ by at least  $m - 1$  there is a number  $k$  between those two sequence numbers such that no element of  $\mathcal{A}$  has  $k$  as its sequence number.

$$\begin{aligned} Interr_m(\mathcal{A}) &:\Leftrightarrow \\ &\forall_{\mathcal{AV}_1, \mathcal{AV}_2 \in \mathcal{A}} ( \underline{Seq}(\mathcal{AV}_2) - \underline{Seq}(\mathcal{AV}_1) \geq m - 1 \Rightarrow \\ &\quad \exists_k ( \underline{Seq}(\mathcal{AV}_1) < k < \underline{Seq}(\mathcal{AV}_2) \wedge \forall_{\mathcal{AV} \in \mathcal{A}} \underline{Seq}(\mathcal{AV}) \neq k ) ) \end{aligned}$$

**Definition 5.10.** "Normal Behavior Scenario": The system behaves normally (from the beginning on) if for any  $(\Delta - N - 1)$  AVs in sequence, at least 1 AV is not lost, and on each transition step, the formulas  $\mathcal{N}_{normal}^{LU}$ ,  $\mathcal{N}_{normal}^{SN}$ ,  $\mathcal{N}_{normal}^{HE}$ ,  $\mathcal{N}_{normal}^{CC}$ , and  $\mathcal{N}_{normal}^{MS}$  hold:

$$\begin{aligned} \mathcal{N}_{normal}^{Step} &:\Leftrightarrow \mathcal{N}_{normal}^{LU} \wedge \mathcal{N}_{normal}^{SN} \wedge \mathcal{N}_{normal}^{HE} \wedge \mathcal{N}_{normal}^{CC} \wedge \mathcal{N}_{normal}^{MS} \\ \mathcal{A}_{normal} &:\Leftrightarrow \\ Init &\wedge \mathcal{A}_{normal}^{interleave} \wedge \mathcal{A}_{normal}^{CurrSN} \wedge \mathcal{A}_{normal}^{LastSN} \wedge \mathcal{A}_{normal}^{Gen} \wedge \mathcal{A}_{normal}^{Lost} \wedge \mathcal{A}_{normal}^{Used} \\ &\wedge \Box( \mathcal{N}_{normal}^{Step} \wedge Interr_{\Delta - N - 1}(Lost') ) \end{aligned}$$

## 6 Transitions of the incorrect System

### 6.1 Events: more Transitions

<i>Message or Event</i>	<i>Trans. Pred.</i>	<i>Param. Name</i>	<i>Param. Type</i>	<i>Var. Name</i>
ERROR HE	XHE	Failure	<i>FAIL</i>	$n_{\text{XHE}}$
ERROR SN	XSN	SN Failure	<i>SN</i> <i>FAIL</i>	$n_{\text{XSN}}$
ERROR LU	XLU	SN Failure	<i>SN</i> <i>FAIL</i>	$n_{\text{XLU}}$

Table 3: Failure Events

**Definition 6.1.** (The Failure Events)

$$\begin{aligned} \text{XHE(Failure)} &:\Leftrightarrow n_{\text{XHE}}(\text{Failure})\updownarrow \\ \text{XSN(SN, Failure)} &:\Leftrightarrow n_{\text{XSN}}(\text{SN, Failure})\updownarrow \\ \text{XLU(SN, Failure)} &:\Leftrightarrow n_{\text{XLU}}(\text{SN, Failure})\updownarrow \end{aligned}$$

We also use conventions similar to the ones in Conventions 3.1 and 3.2. We do not write them explicitly.

Some remarks to the assumptions/failure models: Most race conditions (the non-intended ordering of the processing of events due to concurrency and communication delays) are non-critical. This is due to the fact that the protocol is constructed as a set of requests and responses (or timeouts). In our modeling we use as atomic granularity *complete* actions (request+response or time-out). Nevertheless it is possible to formulate race conditions as the simultaneous performance of two actions (that should happen in order and such that the simultaneous performance is not equivalent to any of the two orderings) or by adding events (like XLU(SN,Race)), that have some unexpected consequences.

In our case, the unexpected consequence of XLU(SN,Race) is that the USIM may change its location simultaneously to a ADR (or equivalently, to an ADS).

Also in that case, the data-base of authentication vectors may have been updated in an unexpected order. There is no real need for explicitly requiring this (as a single transition step) since it is equivalent to a sequential composition of the transitions (in any order): update the database *DB* correctly once, loose, eventually several times, the order of the DB (event: XSN(SN,DB)) and loose AVs (event: XSN(SN,Loss)).

Another more drastic but simple way of modeling this type of situation, allowing even more strange race conditions in which many different SNs are involved (but not changing our properties or proofs), is to allow in the event XSN(DB) (without a parameter SN) to mix the different *DB*s of the different SNs in an arbitrary way:

$$\text{XSN(DB)} \Rightarrow \bigcup_{\text{SN}} \text{Set}(DB'(\text{SN})) = \bigcup_{\text{SN}} \text{Set}(DB(\text{SN}))$$

instead of, as we will have now (see the definition of  $\mathcal{N}_{\text{critical}}^{\text{SN}}$ ):

$$\text{XSN(SN, DB)} \Rightarrow \text{Set}(DB'(\text{SN})) = \text{Set}(DB(\text{SN}))$$

Component	Assumption/Failure Model	Description
USIM	(only case)	The USIM always works correctly. The lifetime of the USIM is not exceeded. (See Def. 4.1).
SN	SN 1. No failure	SN works correctly
	SN 2. AV loss	Loss or corruption of AVs (event: XSN(Loss) )
	SN 3. AV disordering	Disordering of AVs (event: XSN(DB) )
	SN 4. Crash SN	Use of old AVs (event: XSN(Crash) )
	SN 5. SN is compromised	AVs are stolen (event: XSN(Steal) )
HE	HE 1. No failure	HE works correctly
	HE 2. DB-failures	SQN is reset to an older value (event: XHE(DB) )
	HE 3. HE crash	Critical failures: SQN is set to an arbitrary value (event: XHE(Crash) )
	HE 4. HE is compromised	An attacker sets SQN to an arbitrarily chosen value; then AVs are generated and stolen and eventually SQN is set to a new less suspicious value. (But: not generated AVs are never compromised) (event: XHE(Steal) )

Table 4: Assumptions and Failure Models. Part I

Component	Assumption/Failure	Description
Transmission channel (between SN and USIM)	Ch 1. normal situation	In a sequence of transmissions, a certain maximal number of consecutive failures happen (loss or corruption of messages)
	Ch 2. critical situation, probably due to attacks	A huge amount of consecutive messages are lost or corrupted
	Ch 3. replay attacks	Old (=seen) messages are inserted
	Ch 4. complex attacks	Messages using unseen AVs are inserted. Those AVs have been stolen.
Location Update	LU 1. normal situation	Cancel location implies all AVs are deleted. With a Location update request all old AVs are deleted, fresh AVs are requested from the old SN or from the HE/AuC. No race condition happens
	LU 2. failure	After a Location update request old AVs are still present and will be used (event: XLU(SN,DB) )
	LU 3. race conditions	There are several race conditions, for instance: when an SN asks for Authentication Data, ADR, (and in particular, after a synchronisation failure is detected), the USIM changes SN (location update) and the new SN collects new AVs, before the HE is able to process the old ADR. (event: XLU(SN,Race) )

Table 5: Assumptions and Failure Models: Part II



disordering the  $DB$  of only one SN independently of all other SNs.

It is in principle possible that several errors happen within the same transition, but sometimes the specifications of them contradict each other. In any case it is always possible that errors occur immediately after another.

For simplicity, we defined all three types of error (XHE,XSN,XLU) as being of the same type. But each one may happen only for certain parameter values:

$$\begin{aligned} \mathcal{A}_{critical}^{interleave} &:\Leftrightarrow \mathcal{A}_{normal}^{interleave} \wedge \\ &\Box( \wedge \text{XHE(Failure)} \Rightarrow \text{Failure} \in \{\text{DB, Crash, Steal}\} \\ &\wedge \text{XSN(SN, Failure)} \Rightarrow \text{Failure} \in \{\text{Loss, DB, Crash, Steal}\} \\ &\wedge \text{XLU(SN, Failure)} \Rightarrow \text{Failure} \in \{\text{DB, Race}\} ) \end{aligned}$$

## 6.2 Changing the Location

Recall the definition of  $curr\_SN : \wp(\mathcal{SN})$  given in Def. 4.1. This definition imposes no restriction in our specification and remains as it is. The definition of  $last\_SN : \mathcal{SN}$  will also be left unchanged: the value of  $last\_SN$  is of no interest to us when  $curr\_SN$  is a set with two or more elements. But as soon as  $curr\_SN$  is a singleton or empty,  $last\_SN$  has the meaning that we intend: either is  $curr\_SN = \{last\_SN\}$  or it is the last SN where the user was registered.

$$\begin{aligned} \mathcal{A}_{incorr}^{CurrSN} &:\Leftrightarrow \mathcal{A}_{critical}^{CurrSN} :\Leftrightarrow \mathcal{A}_{normal}^{CurrSN} \\ \mathcal{A}_{incorr}^{LastSN} &:\Leftrightarrow \mathcal{A}_{critical}^{LastSN} :\Leftrightarrow \mathcal{A}_{normal}^{LastSN} \end{aligned}$$

$$\begin{aligned} \mathcal{N}_{critical}^{LU} &:\Leftrightarrow \\ &\wedge \text{LUR(SN, SN}_1) \wedge \neg \text{XLU(SN, DB)} \Rightarrow \text{MvAV(SN, SN}_1) \vee \text{DB}'(\text{SN}_1) = \epsilon \\ &\wedge \text{MvAV(SN, SN}_1) \Rightarrow \text{LUR(SN, SN}_1) \\ &\wedge (\text{ADS} \wedge \neg \exists_{\text{SN}} \text{XLU(SN, Race)}) \Rightarrow \text{curr\_SN}' = \text{curr\_SN} \\ &\wedge \text{MvAV(SN, SN}_1) \Rightarrow \wedge \text{DB}'(\text{SN}_1) = \text{DB(SN)} \\ &\wedge \text{DB}'(\text{SN}) = \epsilon \\ &\wedge \forall_{\text{SN}_2} (\text{SN}_2 \neq \text{SN} \wedge \text{SN}_2 \neq \text{SN}_1 \Rightarrow \\ &\quad \text{DB}'(\text{SN}_2) = \text{DB}'(\text{SN}_2)) \end{aligned}$$

The definition of  $\mathcal{N}_{critical}^{LU}$  is very close to the one of  $\mathcal{N}_{normal}^{LU}$ . There we had (rewriting a bit the original formula):

$$\text{LUR(SN, SN}_1) \wedge \neg \text{MvAV(SN, SN}_1) \Rightarrow \text{DB}'(\text{SN}_1) = \epsilon$$

but now, if  $\text{XLU(SN, DB)}$  happens,  $\text{LUR(SN, SN}_1) \wedge \neg \text{MvAV(SN, SN}_1)$  may imply  $\text{DB}'(\text{SN}_1) \neq \epsilon$  (thus old AVs may be used: LU 2.). It is not necessary to explicitly state what happens if  $\text{LUR(SN, SN}_1) \wedge \text{XLU(SN, DB)}$ : either  $\text{MvAV(SN, SN}_1)$  (in which case  $DB$  changes in the prescribed way) or  $\text{DB(SN}_1)$  does not change, unless there is another reason for changing  $\text{DB}'(\text{SN}_1)$  (those reasons are given in the definition of  $\mathcal{N}_{critical}^{SN}$  after  $\text{DB(SN)} \Downarrow \Rightarrow \dots$ ).

The other difference to  $\mathcal{N}_{normal}^{LU}$  is that if the race condition happens, then while ADS is performed,  $(\text{ADS} \wedge \neg \exists_{\text{SN}} \text{XLU(SN, Race)})$ , then it may not be excluded that either a LUR or a CANLOC happen, the mobile station thus changing the location.

### 6.3 The Serving Network

$$\begin{aligned}
\mathcal{N}_{critical}^{SN} : \Leftrightarrow & \\
& \wedge \text{ADR}_0(\text{SN}) \Rightarrow \text{DB}(\text{SN}) = \epsilon \wedge \text{SN} \in \text{curr\_SN} \\
& \wedge \text{ADR}_1(\text{SynResp}, \text{Rand}, \text{SN}) \Rightarrow \\
& \quad \wedge \text{UASF}_R(\text{SynResp}, \text{SN}) \\
& \quad \wedge \text{SN} \in \text{curr\_SN} \\
& \quad \wedge \exists_{AV} (\text{UAR}_s(\underline{\text{Chall}}(\text{AV}), \text{SN}) \wedge \text{Rand} = \underline{\text{Rand}}(\text{AV})) \\
& \wedge \text{ADS}(\text{AVs\_new}, \text{SN}) \Rightarrow \wedge \text{AVs\_new} \neq \epsilon \Rightarrow \text{DB}'(\text{SN}) = \text{AVs\_new} \\
& \quad \wedge \text{AVs\_new} = \epsilon \Rightarrow \text{DB}'(\text{SN}) = \text{DB}(\text{SN}) \\
& \wedge \text{UAR}_s(\text{chall}, \text{SN}) \wedge \neg \text{XSN}(\text{SN}, \text{Loss}) \Rightarrow \\
& \quad \wedge \text{SN} \in \text{curr\_SN} \wedge \text{DB}(\text{SN}) \neq \epsilon \\
& \quad \wedge \text{DB}'(\text{SN}) = \text{Tail}(\text{DB}(\text{SN})) \\
& \quad \wedge \text{chall} = \underline{\text{Chall}}(\text{Head}(\text{DB}(\text{SN}))) \\
& \wedge \text{XSN}(\text{SN}, \text{Crash}) \Rightarrow \text{UAR}_s \\
& \wedge \text{UAR}_s(\text{chall}, \text{SN}) \wedge \text{XSN}(\text{SN}, \text{Crash}) \Rightarrow \\
& \quad \exists_{AV \in \text{Used}} \text{chall} = \underline{\text{Chall}}(\text{AV}) \\
& \wedge \text{XSN}(\text{SN}, \text{DB}) \Rightarrow \text{Set}(\text{DB}'(\text{SN})) = \text{Set}(\text{DB}(\text{SN})) \\
& \wedge \text{XSN}(\text{SN}, \text{Loss}) \Rightarrow \wedge \text{Set}(\text{DB}'(\text{SN})) \subseteq \text{Set}(\text{DB}(\text{SN})) \\
& \quad \wedge \text{AV}_1 \ll_{\text{DB}'(\text{SN})} \text{AV}_2 \Rightarrow \text{AV}_1 \ll_{\text{DB}(\text{SN})} \text{AV}_2 \\
& \wedge \text{DB}(\text{SN}) \Downarrow \Rightarrow \vee \exists_{\text{SN}_1} \text{LUR}(\text{SN}_1, \text{SN}) \wedge \neg \text{XLU}(\text{SN}_1, \text{DB}) \\
& \quad \vee \text{XSN}(\text{SN}, \text{DB}) \vee \text{XSN}(\text{SN}, \text{Loss}) \\
& \quad \vee \exists_{\text{AVs\_new}, \text{SN}} \text{ADS}(\text{AVs\_new}, \text{SN}) \wedge \text{AVs\_new} \neq \epsilon \\
& \quad \vee \exists_{\text{chall}, \text{SN}} \text{UAR}_s(\text{chall}, \text{SN}) \wedge \neg \text{XSN}(\text{SN}, \text{Loss}) \\
& \quad \vee \exists_{\text{SN}_1} \text{MvAV}(\text{SN}, \text{SN}_1) \\
& \quad \vee \exists_{\text{SN}_1} \text{MvAV}(\text{SN}_1, \text{SN}) \\
& \wedge [ \wedge \text{UASF}_R(\text{SynResp}, \text{SN}) \\
& \quad \wedge \text{SN} \in \text{curr\_SN} \\
& \quad \wedge \text{UAR}_s(\underline{\text{Chall}}(\text{AV}), \text{SN}) \\
& \quad \wedge \text{Rand} = \underline{\text{Rand}}(\text{AV}) ] \Rightarrow \text{ADR}_1(\text{SynResp}, \text{Rand}, \text{SN})
\end{aligned}$$

### 6.4 The Home Environment

In the real system it seems to be the case that if a race condition happens:

$$(\text{ADR}_0 \vee \text{ADR}_1) \wedge \text{XLU}(\text{SN}, \text{Race}) \wedge \text{LUR}(\text{SN}, \text{SN}_1)$$

then a  $\text{CANLOC}(\text{SN})$  can be produced, instead of the  $\text{ADS}$  expected by the serving network. But we insist that  $(\text{ADR}_0 \vee \text{ADR}_1) \Rightarrow \text{ADS}$ . We model the described situation as follows: first send a  $\text{ADS}(\text{AVs\_new}, \text{SN})$  with  $\text{AVs\_new} = \epsilon$  and then send a  $\text{CANLOC}$ . This sequence is equivalent to just sending one  $\text{CANLOC}$ . Notice that our specification does not constrain at all the occurrences of  $\text{CANLOC}$ : they may happen anytime. (They are seen as inputs to the system).

It is also assumed that AVs which have not been generated can not be stolen (the

code for the generation of AVs is secure).

$$\begin{aligned}
\mathcal{N}_{critical}^{HE} &:\Leftrightarrow \exists_{seq_{he}} [ \\
&\wedge ADS(AVs\_new, SN) \Rightarrow \vee ADR_0(SN) \\
&\qquad\qquad\qquad \vee \exists_{SynResp, Rand} ADR_1(SynResp, Rand, SN) \\
&\wedge (ADR_0 \vee ADR_1) \Rightarrow ADS \\
&\wedge ADR_0 \Rightarrow seq_{he} = seq_{HE} \\
&\wedge ADR_1(SynResp, Rand, SN) \Rightarrow \\
&\quad \wedge [verif(SynResp, Rand) \wedge \\
&\quad \quad \neg synchr_1(\underline{Seq}_{MS}(SynResp), seq_{HE})] \Rightarrow seq_{he} = \underline{Seq}_{MS}(SynResp) \\
&\quad \wedge [\neg verif(SynResp, Rand) \vee \\
&\quad \quad synchr_1(\underline{Seq}_{MS}(SynResp), seq_{HE})] \Rightarrow seq_{he} = seq_{HE} \\
&\wedge ADS(AVs\_new, SN) \Rightarrow \\
&\quad \wedge \neg XLU(SN, Race) \Rightarrow AVs\_new \neq \epsilon \\
&\quad \wedge AV \in AVs\_new \Rightarrow cons(AV) \\
&\quad \wedge \forall_{AV_1, AV_2 \in AVs\_new} (AV_1 \prec AV_2 \Leftrightarrow AV_1 \ll AV_2) \\
&\quad \wedge \forall_{AV \in AVs\_new} (seq_{he} < \underline{Seq}(AV) \leq seq'_{HE}) \\
&\quad \wedge \forall_{i \in \mathbb{N}} \exists_{AV \in AVs\_new} (seq_{he} < i \leq seq'_{HE} \Rightarrow \underline{Seq}(AV) = i) \\
&\quad \wedge seq'_{HE} - seq_{he} \leq N \\
&\wedge XHE(DB) \Rightarrow seq_{HE}' < seq_{HE} \\
&\wedge seq_{HE} \uparrow \Rightarrow \vee \exists_{AVs\_new, SN} ADS(AVs\_new, SN) \wedge AVs\_new \neq \epsilon \\
&\quad \vee XHE(DB) \\
&\quad \vee XHE(Steal) \\
&\quad \vee XHE(Crash) ]
\end{aligned}$$

Notice that XHE(Steal) or XHE(Crash) do not impose any restriction on the value of  $seq_{HE}'$ . Therefore, after this sort of failures the sequence number of the home environment may assume an arbitrary value.

## 6.5 The Communication Channel

The definitions of *OKR*, *CorrR*, *LossR*, *OKS*, *CorrS*, and *LossS* were given in Section 4.4. These definitions are still valid for the incorrect and the critical system. As before, the message USER AUTHENT. REQUEST may be received correctly (*OKR*), or it may be corrupted during the transmission (*CorrR*), or it may get lost (*LossR*). But now there is one possibility more: it may be also replaced by another USER AUTHENT. REQUEST with another challenge (*ATTR*).

$$\begin{aligned}
ATTR &:\Leftrightarrow \exists_{chall, SN} \wedge UAR_s(chall, SN) \\
&\quad \wedge \exists_{chall_1 \neq chall} cons(chall_1) \wedge UAR_r(chall_1, SN)
\end{aligned}$$

Notice that the following is a tautology:

$$\begin{aligned}
UAR_s(chall, SN) &\Rightarrow \vee UAR_r(chall, SN) \\
&\quad \vee \exists_{chall_1} \neg cons(chall_1) \wedge UAR_r(chall_1, SN) \\
&\quad \vee \neg UAR_r(SN) \\
&\quad \vee \exists_{chall_1 \neq chall} cons(chall_1) \wedge UAR_r(chall_1, SN)
\end{aligned}$$

Thus,  $UAR_S \Rightarrow OKR \vee CorrR \vee LossR \vee ATTR$  is a tautology.

There is another form of attack,  $ATTR_i$ , the *insertion* of a message that was not sent. In this situation, the only interesting case is when the inserted challenge is consistent:

$$ATTR_i :\Leftrightarrow \exists_{\text{chall}, \text{SN}} \wedge UAR_R(\text{chall}, \text{SN}) \wedge \text{cons}(\text{chall}) \\ \wedge \neg UAR_S$$

On the other direction, the message USER AUTHENT. RESPONSE may be received correctly ( $OKS$ ), or it may be corrupted during the transmission ( $CorrS$ ), or it may get lost ( $LossS$ ), or it may be replaced by another USER AUTHENT. RESPONSE with another response ( $ATTR$ ). Notice that in the definition of  $\mathcal{N}_{normal}^{CC}$ , if a message was received, then a corresponding message was also sent (perhaps with different parameter values, they can be corrupted). For instance, if  $UAS_R$  happens, then either  $OKR$  or  $CorrR$  or  $LossR$  happens, and in any case,  $UAS_S$  happens as well. This is not true anymore. Notice that we do not have to model extra an attack  $ATTS$ , since it is indistinguishable from a corruption  $CorrS$ . (In the other direction  $ATTR$  is needed, since  $CorrR$  implies that the corrupted challenge is inconsistent). The insertion attack for messages from the mobile station to the service network are only interesting when the service network has issued an authentication request (else the insertion of the message has no consequences):

$$ATTS_i(\text{SN}) :\Leftrightarrow \wedge \exists_{\text{chall}} UAR_S(\text{chall}, \text{SN}) \\ \wedge \neg \exists_{\text{resp}} UAS_S(\text{resp}, \text{SN}) \\ \wedge \neg \exists_{\text{RejResp}} UAJ_S(\text{RejResp}, \text{SN}) \\ \wedge \neg \exists_{\text{SynResp}} UASF_S(\text{SynResp}, \text{SN}) \\ \wedge \vee \exists_{\text{resp}} UAS_R(\text{resp}, \text{SN}) \\ \vee \exists_{\text{RejResp}} UAJ_R(\text{RejResp}, \text{SN}) \\ \vee \exists_{\text{SynResp}} UASF_R(\text{SynResp}, \text{SN}) \\ ATTS_i :\Leftrightarrow \exists_{\text{SN}} ATTS_i(\text{SN})$$

The attacker is not able to generate consistent challenges that he has not seen, that is, either have been transmitted already, or he has stolen. (The definition of *Stolen* is given in Definition 5.2).

**Assumption 6.1.**

$$\mathcal{N}_{critical}^{Ass_1} :\Leftrightarrow \\ [ UAR_R(\text{chall}, \text{SN}) \wedge \neg UAR_S(\text{chall}, \text{SN}) \wedge \text{cons}(\text{chall}) ] \\ \Rightarrow \exists_{\text{AV} \in \text{Stolen} \cup \text{Used}} \text{chall} = \underline{\text{Chall}}(\text{AV})$$

The attacker is not able to generate consistent responses to challenges that he has not seen.

**Assumption 6.2.**

$$\mathcal{N}_{critical}^{Ass_2} :\Leftrightarrow \\ [ UAR_S(\text{chall}, \text{SN}) \wedge \neg UAS_S(\text{resp}, \text{SN}) \wedge UAS_R(\text{resp}, \text{SN}) \wedge \text{cons}(\text{chall}, \text{resp}) ] \\ \Rightarrow \exists_{\text{AV} \in \text{Stolen} \cup \text{Used}} \text{chall} = \underline{\text{Chall}}(\text{AV}) \wedge \text{resp} = \underline{\text{Resp}}(\text{AV})$$

The attacker is not able to generate consistent fail synchronisation responses to fresh challenges.

**Assumption 6.3.**

$$\begin{aligned} \mathcal{N}_{critical}^{Ass_3} &:\Leftrightarrow \\ &[\wedge \text{UAR}_s(\text{chall}, \text{SN}) \wedge \text{UASF}_R(\text{SynResp}, \text{SN}) \\ &\wedge \neg \text{UASF}_s(\text{SynResp}, \text{SN}) \wedge \text{verif}(\text{SynResp}, \underline{\text{Rand}}(\text{AV}))] \\ &\Rightarrow (\text{AV} \in \text{Stolen} \cup \text{Used}) \wedge \text{SynResp} = \underline{\text{SynResp}}(\text{AV}) \end{aligned}$$

Those assumptions are the specification of  $\mathcal{N}_{critical}^{CC}$ :

$$\mathcal{N}_{critical}^{CC} :\Leftrightarrow \mathcal{N}_{critical}^{Ass_1} \wedge \mathcal{N}_{critical}^{Ass_2} \wedge \mathcal{N}_{critical}^{Ass_3}$$

**6.6 The Mobile Station**

The mobile station is assumed to work always correctly, therefore,

$$\mathcal{N}_{critical}^{MS} :\Leftrightarrow \mathcal{N}_{normal}^{MS}$$

**7 Definition of Incorrect Behavior**

**Definition 7.1.** *The stealing of AVs generates a clone (not a "copy") of an AV.  $\text{Stolen}^3$ , the set of clones, is defined by the formulas:*

$$\begin{aligned} \text{Stolen} &:= \text{Stolen}_{HE} \cup \text{Stolen}_{SN} \\ \text{Stolen}_{HE} &= \emptyset \wedge \square(\wedge \text{Stolen}_{HE} \uparrow \Rightarrow \text{XHE}(\text{Steal}) \\ &\quad \wedge \text{XHE}(\text{Steal}) \Rightarrow \text{Stolen}_{HE}' \supseteq \text{Stolen}_{HE}) \\ \text{Stolen}_{SN} &= \emptyset \wedge \square(\wedge \text{Stolen}_{SN} \uparrow \Rightarrow \text{XSN}(\text{SN}, \text{Steal}) \\ &\quad \wedge \text{XSN}(\text{SN}, \text{Steal}) \Rightarrow \\ &\quad \text{Stolen}_{SN} \subseteq \text{Stolen}_{SN}' \subseteq \text{Stolen}_{SN} \cup \text{DB}(\text{SN})) \end{aligned}$$

**Definition 7.2.** *As before (Def. 5.5), if  $\text{curr\_SN}$  is a singleton or empty, an unused AV copy is usable if its location is  $\text{last\_SN}$ , the current SN where the user is registered (or where the user was last registered), that is, it is an element of  $\text{DB}(\text{last\_SN})$ . If  $\text{curr\_SN}$  contains more than one element, then an unused AV is usable if its location is contained in  $\text{curr\_SN}$ , that is, it is an element of  $\cup_{\text{SN} \in \text{curr\_SN}} \text{DB}(\text{SN})$ .*

In the Definition 5.2, we have defined the variable *Lost*, the set of lost authentication vectors. This definition is now extended to the case where the protocol is not running under normal conditions, but under incorrect or critical ones. Now, the system may also lose AVs through the event XSN, or through attacks. Notice that also the disordering of AVs (or, if you prefer, the usage of AVs in disorder) leads to losing AVs.

**Definition 7.3.** *An AV copy is lost if it is either:*

1. *lost or corrupted by an error in SN (SN 2. or SN 3.)*
2. *lost or corrupted in the communication Channel between the SN and the USIM, perhaps also due to an attack (Ch. 1 or Ch. 2)*

---

<sup>3</sup>In our formal specification we do not distinguish between AVs and *occurrences* of AVs. Thus, in the formal specification one AV may be at the same time in *Stolen* and in *Used* or *Lost*.

3. *lost or intentionally discarded during a Location Update (typically LU 1, but also LU 2 and LU 3)*

Formally, the variable *Lost*, of type  $\wp(\mathcal{AV})$  is defined by:

$$\begin{aligned}
\mathcal{A}_{critical}^{Lost} : \Leftrightarrow & \\
& \wedge \text{Lost} = \emptyset \\
& \wedge \square( \wedge \text{Lost} \not\Leftarrow \Rightarrow \vee \text{UAR}_s \wedge \neg \text{OKR} \\
& \quad \vee \text{LUR}(\text{SN}, \text{SN}_1) \wedge \neg \text{MvAV}(\text{SN}, \text{SN}_1) \\
& \quad \vee \exists_{\text{SN}} \text{XSN}(\text{SN}, \text{Loss}) \\
& \quad \vee \exists_{\text{SN}} \text{XSN}(\text{SN}, \text{DB}) \\
& \wedge \text{UAR}_s(\underline{\text{Chall}}(\text{AV}), \text{SN}) \wedge \neg \text{OKR} \\
& \quad \Rightarrow \text{Lost}' = \text{Lost} \cup \{\text{AV}\} \\
& \wedge \text{LUR}(\text{SN}, \text{SN}_1) \wedge \neg \text{MvAV}(\text{SN}, \text{SN}_1) \Rightarrow \\
& \quad \text{Lost}' = \text{Lost} \cup \text{DB}(\text{SN}) \\
& \wedge \text{XSN}(\text{SN}, \text{Loss}) \Rightarrow \text{Lost}' = \text{Lost} \cup (\text{DB}(\text{SN}) \setminus \text{DB}'(\text{SN})) \\
& \wedge \text{XSN}(\text{SN}, \text{DB}) \Rightarrow \text{Lost}' = \text{Lost} \cup \text{DB}(\text{SN}) )
\end{aligned}$$

This definition of *Lost* is only one of several possible choices. We could say that if  $\text{XSN}(\text{SN}, \text{DB})$  happens, not all AVs in  $\text{DB}(\text{SN})$  are lost, only those AV for which there is an  $\text{AV}_1$  in  $\text{DB}'(\text{SN})$ , such that  $\text{AV}_1$  is left of AV (it will be used earlier than AV) but  $\text{AV}_1$  has a larger sequence number than AV:

$$\begin{aligned}
& \text{XSN}(\text{SN}, \text{DB}) \Rightarrow \\
& \quad \text{Lost}' = \text{Lost} \cup \{\text{AV} \in \text{DB}(\text{SN}) \mid \exists_{\text{AV}_1} \text{AV}_1 \ll \text{AV} \wedge \text{AV} \prec \text{AV}_1\}
\end{aligned}$$

Or we could also say: using an AV with a sequence number larger than one already used (or, accepted) is loosing this AV. The “exact” definition of “lost” is not so important. But: we *need* such a definition (to be able to define what it means to return to normal behavior) and this definition has to be consistent with the one given for normal behavior, which should be a particular case.

**Definition 7.4.** *The definitions of generated and used copy remain the same:*

$$\mathcal{A}_{critical}^{Gen} : \Leftrightarrow \mathcal{A}_{normal}^{Gen} \quad \mathcal{A}_{normal}^{Used} : \Leftrightarrow \mathcal{A}_{normal}^{Used}$$

**Definition 7.5.** *An AV clone is obsolete if  $\text{seq}_{MS} \geq \underline{\text{Seq}}(\text{AV})$ . The set of obsolete clones is denoted by *Obsolete*. By definition,*

$$\text{Obsolete} \subseteq \text{Stolen} = \text{Stolen}_{HE} \cup \text{Stolen}_{SN}.$$

**Definition 7.6.** *An AV clone is called synchronous (with respect to  $\text{seq}_{MS}$ ) if  $\text{synchron}(\text{seq}_{MS}, \underline{\text{Seq}}(\text{AV}))$*

Notice that this is the same definition as 5.7 but for clones.

**Definition 7.7.** *The system is in perfect conditions (at a certain moment of time) if it is strongly-synchronous and any AV clone is obsolete:*

$$\text{Perfect} : \Leftrightarrow \text{StrongSynchron} \wedge \text{Stolen} \subseteq \text{Obsolete}$$

Notice that in the case that there are no clones (and in particular, if from the beginning the system was behaving normally) then  $\text{Perfect} : \Leftrightarrow \text{StrongSynchron}$ .

**Definition 7.8.** "Critical Behavior Scenario": *The system behaves critically if an arbitrary combination of assumptions or failure models (SN 1 – LU 3) may occur:*

$$\begin{aligned} \mathcal{N}_{critical}^{Step} &:\Leftrightarrow \mathcal{N}_{critical}^{LU} \wedge \mathcal{N}_{critical}^{SN} \wedge \mathcal{N}_{critical}^{HE} \wedge \mathcal{N}_{critical}^{CC} \wedge \mathcal{N}_{critical}^{MS} \\ \mathcal{A}_{critical} &:\Leftrightarrow \\ &\text{Init} \wedge \mathcal{A}_{critical}^{interleave} \wedge \mathcal{A}_{critical}^{CurrSN} \wedge \mathcal{A}_{critical}^{LastSN} \wedge \mathcal{A}_{critical}^{Gen} \wedge \mathcal{A}_{critical}^{Lost} \wedge \mathcal{A}_{critical}^{Used} \\ &\wedge \Box(\mathcal{N}_{critical}^{Step}) \end{aligned}$$

**Definition 7.9.** "Incorrect Behavior Scenario": *The system behaves incorrectly if*

- For any  $\Delta$  AVs in sequence, at most  $\Delta - 1$  are lost,
- disordering of AVs occur, (SN 3) but no SN-crashes or SN-steal
- a failure in the Location Update may happen (LU 2), but no race condition
- no errors in the home environment happen.

$$\begin{aligned} \mathcal{A}_{incorr} &:\Leftrightarrow \mathcal{A}_{critical} \wedge \Box( \wedge \text{Interr}_{\Delta-N-1}(\text{Lost}') \\ &\wedge \neg \text{XSN}(\text{Crash}) \\ &\wedge \neg \text{XSN}(\text{Steal}) \\ &\wedge \neg \text{XLU}(\text{Race}) \\ &\wedge \neg \text{XHE} ) \end{aligned}$$

After the system has been behaving incorrectly, it may return to normal:

**Definition 7.10.** "Return to Normal Behavior": *Let  $x$  be a boolean variable (or state predicate) with the property that if it is  $\underline{1}$ , it remains  $\underline{1}$ :  $\Box(x \Downarrow \Rightarrow x' = \underline{1})$ . Then we say that the system behaves normally when  $x$  if during the time that  $x = \underline{1}$  for any  $(\Delta - N - 1)$  AVs in sequence, at least 1 AV is not lost, and on each transition step, the formulas  $\mathcal{N}_{normal}^{LU}$ ,  $\mathcal{N}_{normal}^{SN}$ ,  $\mathcal{N}_{normal}^{HE}$ ,  $\mathcal{N}_{normal}^{CC}$ , and  $\mathcal{N}_{normal}^{MS}$  hold:*

$$\begin{aligned} \mathcal{A}_{normal}^x &:\Leftrightarrow \\ &\text{Init} \wedge \mathcal{A}_{critical}^{interleave} \wedge \mathcal{A}_{critical}^{CurrSN} \wedge \mathcal{A}_{critical}^{LastSN} \wedge \mathcal{A}_{critical}^{Gen} \wedge \mathcal{A}_{critical}^{Lost} \wedge \mathcal{A}_{critical}^{Used} \\ &\wedge \forall_{\text{Lost}_0: \wp(\text{AV})} [ (\neg x \wedge x' \Rightarrow \text{Lost}' = \text{Lost}_0) \\ &\Rightarrow \Box\{ x \Rightarrow \mathcal{N}_{normal}^{Step} \wedge \text{Interr}_{\Delta-N-1}(\text{Lost}' \setminus \text{Lost}_0) \} ] \end{aligned}$$

Note that "normal behavior" is more a property of the environment of the system (attacks, loosing messages, race conditions, failures in data-bases, etc.) as of the proper system itself. Even if the system returns to "normal behavior", the old failures may still have consequences, for instance, AVs with an old sequence number may be used. Often, only a "succesfull" synchronisation failure will "clean up" the system".

Notice also that the definition of return to normal behavior does not exclude the possibility that some messages get lost. (There is no bound on how many messages from the MS to the SN may get lost!) In particular if the system is not synchronous, this condition will remain unnoticed as long as the messages User Authentication Request and User Authentication Synchronisation Fail Indication get lost. In this case a "succesfull" synchronisation failure is not only helpful, it is necessary:

**Definition 7.11.** *We say that a Synchronisation Failure is successful, if*

1. the corresponding messages *User Authentication Request* and *User Authentication Synchronisation Fail Indication* do not get lost or corrupted, and if
2. this *Fail Indication* is processed by the *HE* before the *USIM* changes the *SN* location, i.e. no race condition *LU 3* happens.

Formally, a successful *Synchronisation Failure* is given by the formula:

$$\begin{aligned}
\text{UASF}_{\text{successful}} & :\Leftrightarrow \exists_{\text{chall}, \text{SN}, \text{SynResp}} \\
& \wedge \text{UAR}_{\text{s}}(\text{chall}, \text{SN}) \wedge \text{UAR}_{\text{r}}(\text{chall}, \text{SN}) \\
& \wedge \text{UASF}_{\text{s}}(\text{SynResp}, \text{SN}) \wedge \text{UASF}_{\text{r}}(\text{SynResp}, \text{SN}) \\
& \wedge \neg \exists_{\text{SN}} \text{XLU}(\text{SN}, \text{Race})
\end{aligned}$$

## 8 Theorems

**Lemma 8.1.**

$$\begin{aligned}
\mathcal{A}_{\text{normal}}^{\text{interleave}} \wedge \mathcal{N}_{\text{normal}}^{\text{MS}} & \Rightarrow \\
& \wedge \text{UAS}_{\text{s}}(\text{resp}, \text{SN}) \Rightarrow \exists_{\text{chall}} \wedge \text{UAR}_{\text{r}}(\text{chall}, \text{SN}) \\
& \wedge \text{cons}(\text{chall}) \\
& \wedge \text{synchron}(\text{seq}_{\text{MS}}, \underline{\text{Seq}}(\text{chall})) \\
& \wedge \text{resp} = \underline{\text{Resp}}(\text{chall}) \\
& \wedge \text{UAJ}_{\text{s}}(\text{RejResp}, \text{SN}) \Rightarrow \exists_{\text{chall}} \text{UAR}_{\text{r}}(\text{chall}, \text{SN}) \wedge \neg \text{cons}(\text{chall}) \\
& \wedge \text{UASF}_{\text{s}}(\text{SynResp}, \text{SN}) \Rightarrow \exists_{\text{chall}} \wedge \text{UAR}_{\text{r}}(\text{chall}, \text{SN}) \\
& \wedge \text{cons}(\text{chall}) \\
& \wedge \neg \text{synchron}(\text{seq}_{\text{MS}}, \underline{\text{Seq}}(\text{chall})) \\
& \wedge \text{SynResp} = \underline{\text{SynResp}}(\text{la\_chall}, \text{chall})
\end{aligned}$$

PROOF: The proof is done by simple predicate logic. All three claims are proven in exactly the same way. Let us prove the second one, which is the shortest one. It amounts to showing

$$\begin{aligned}
\mathcal{A}_{\text{normal}}^{\text{interleave}} \wedge \mathcal{N}_{\text{normal}}^{\text{MS}} \wedge \text{UAJ}_{\text{s}}(\text{RejResp}, \text{SN}) & \Rightarrow \\
& \exists_{\text{chall}} \text{UAR}_{\text{r}}(\text{chall}, \text{SN}) \wedge \neg \text{cons}(\text{chall})
\end{aligned}$$

$$\begin{aligned}
& \mathcal{A}_{\text{normal}}^{\text{interleave}} \wedge \mathcal{N}_{\text{normal}}^{\text{MS}} \wedge \text{UAJ}_{\text{s}}(\text{RejResp}, \text{SN}) \\
& \Rightarrow \text{UAJ}_{\text{s}} && (\text{Def of } \text{UAJ}_{\text{s}}) \\
& \Rightarrow \text{UAS}_{\text{s}} \vee \text{UAJ}_{\text{s}} \vee \text{UASF}_{\text{s}} && (\text{Conjunction Rules}) \\
& \Rightarrow \text{UAR}_{\text{r}} && (\text{Def of } \mathcal{N}_{\text{normal}}^{\text{MS}}) \\
& \Rightarrow \exists_{\text{chall}, \text{SN}_1} \text{UAR}_{\text{r}}(\text{chall}, \text{SN}_1) && (\text{Def of } \text{UAR}_{\text{s}}) \\
& \Rightarrow \text{UAR}_{\text{r}}(\text{chall}, \text{SN}_1) && (\text{Skolemisation: Introd.} \\
& && \text{of fresh variables})
\end{aligned}$$



$$\begin{aligned}
& [ \text{Assume } \text{cons}(\text{chall}) \wedge \text{synchron}(seq_{MS}, \underline{Seq}(\text{chall})) \\
& \quad \Rightarrow \text{UAS}_s(\underline{Resp}(\text{chall}), SN_1) \quad (\text{Def of } \mathcal{N}_{normal}^{MS}) \\
& \quad \Rightarrow \text{UAS}_s \quad (\text{Def of } \text{UAS}_s) \\
& \quad \Rightarrow \neg \text{UAJ}_s \quad (\text{Def of } \mathcal{A}_{normal}^{interleave}) \\
& \quad \Rightarrow \text{Contradiction} \quad (\text{UAJ}_s) ]
\end{aligned}$$

$$\begin{aligned}
& \Rightarrow \neg(\text{cons}(\text{chall}) \wedge \text{synchron}(seq_{MS}, \underline{Seq}(\text{chall}))) \quad (\text{Assumption false}) \\
& \Rightarrow \neg \text{cons}(\text{chall}) \vee \neg \text{synchron}(seq_{MS}, \underline{Seq}(\text{chall})) \quad (\text{De Morgan})
\end{aligned}$$

$$\begin{aligned}
& [ \text{Assume } \text{cons}(\text{chall}) \wedge \neg \text{synchron}(seq_{MS}, \underline{Seq}(\text{chall})) \\
& \quad \Rightarrow \text{UASF}_s(\underline{SynResp}(la\_chall, \text{chall}), SN) \quad (\text{Def of } \mathcal{N}_{normal}^{MS}) \\
& \quad \Rightarrow \text{UASF}_s \quad (\text{Def of } \text{UAS}_s) \\
& \quad \Rightarrow \neg \text{UAJ}_s \quad (\text{Def of } \mathcal{A}_{normal}^{interleave}) \\
& \quad \Rightarrow \text{Contradiction} \quad (\text{UAJ}_s) ]
\end{aligned}$$

$$\begin{aligned}
& \Rightarrow \neg(\text{cons}(\text{chall}) \wedge \neg \text{synchron}(seq_{MS}, \underline{Seq}(\text{chall}))) \quad (\text{Assumption false}) \\
& \Rightarrow \neg \text{cons}(\text{chall}) \vee \text{synchron}(seq_{MS}, \underline{Seq}(\text{chall})) \quad (\text{De Morgan}) \\
& \Rightarrow \neg \text{cons}(\text{chall}) \quad (\text{Resolution}) \\
& \Rightarrow \text{UAJ}_s(\underline{RejResp}(\text{chall}), SN_1) \quad (\text{Def of } \mathcal{N}_{normal}^{MS} \\
& \quad \text{and } \text{UAR}_R(\text{chall}, SN_1)) \\
& \Rightarrow \text{RejResp} = \underline{RejResp}(\text{chall}) \wedge SN = SN_1 \quad (\text{UAJ}_s(\underline{RejResp}(\text{chall}), SN_1) \\
& \quad \text{UAR}_s(\text{RejResp}, SN) \\
& \quad \text{and Def of } \mathcal{A}_{normal}^{interleave}) \\
& \Rightarrow \text{UAR}_R(\text{chall}, SN) \quad (\text{SN} = SN_1) \\
& \Rightarrow \text{UAR}_R(\text{chall}, SN) \wedge \neg \text{cons}(\text{chall}) \quad (\text{Conjunction}) \\
& \Rightarrow \exists_{\text{chall}} \text{UAR}_R(\text{chall}, SN) \wedge \neg \text{cons}(\text{chall}) \quad (\text{Introd of Ex.})
\end{aligned}$$

**Lemma 8.2.**

$$\begin{aligned}
& \mathcal{A}_{normal}^{CurrSN} \wedge \mathcal{A}_{normal}^{LastSN} \Rightarrow \\
& \quad \square( (\mathcal{N}_{normal}^{LU} \wedge \text{curr\_SN}' \neq \emptyset) \Rightarrow \text{curr\_SN}' = \{\text{last\_SN}\} )
\end{aligned}$$

**Lemma 8.3.**

$$\begin{aligned}
& \mathcal{A}_{normal} \Rightarrow \square( \wedge DB(\text{last\_SN}) \neq \epsilon \Rightarrow seq_{HE} = \underline{Seq}(\max(DB(\text{last\_SN}))) \\
& \quad \wedge seq_{MS} = \underline{Seq}(\max(\text{Accepted})) \\
& \quad \wedge \text{UAR}_s(\underline{Chall}(\text{AV}), SN) \Rightarrow \text{AV} = \min(DB(\text{last\_SN})) )
\end{aligned}$$

PROOF: 1. The first goal is to prove

$$\mathcal{A}_{normal} \Rightarrow \square(DB(\text{last\_SN}) \neq \epsilon \Rightarrow seq_{HE} = \underline{Seq}(\max(DB(\text{last\_SN}))))$$

This follows if in any transition where  $seq_{HE}$  or  $DB$  or  $last\_SN$  changes,

$$DB'(last\_SN') \neq \epsilon \Rightarrow seq_{HE}' = \underline{Seq}(\max(DB'(last\_SN')))$$

holds.

1.1. Assume  $seq_{HE} \uparrow$ . First use the definition of  $\mathcal{N}_{normal}^{HE}$ .  $seq_{HE}$  changes only when ADS happens. Choosing fresh variables for the parameters we may assume  $ADS(AVs\_new, SN)$ . It is easy to see that  $AVs\_new \neq \epsilon$ , (else  $seq_{HE}$  does not change). Recalling the definition of  $\mathcal{N}_{normal}^{HE} : \Leftrightarrow \exists_{seq_{he}} \square$ , choose  $seq_{he}$  to be any value that makes the predicate in the square brackets to be true. (Skolemisation). Now, since

$$\forall_{i \in \mathbb{N}} \exists_{AV \in AVs\_new} (seq_{he} < i \leq seq_{HE}' \Rightarrow \underline{Seq}(AV) = i)$$

letting  $i = seq_{HE}'$  we have

$$\exists_{AV \in AVs\_new} \underline{Seq}(AV) = seq_{HE}'$$

Choose  $AV_0$  with  $\underline{Seq}(AV_0) = seq_{HE}'$ . Now, from

$$\forall_{AV \in AVs\_new} (seq_{he} < \underline{Seq}(AV) \leq seq_{HE}')$$

it follows that  $\forall_{AV \in AVs\_new} (\underline{Seq}(AV) \leq \underline{Seq}(AV_0))$ , which may be written as  $AV_0 = \max(AVs\_new)$ .

Using the definition of  $\mathcal{N}_{normal}^{SN}$ , we have that  $ADS(AVs\_new, SN)$  implies  $DB'(SN) = AVs\_new$  and therefore  $AV_0 = \max(DB'(SN))$ . Then

$$\underline{Seq}(AV_0) = \underline{Seq}(\max(DB'(SN))) = seq_{HE}'$$

proving the claim, since  $SN \in curr\_SN = curr\_SN'$  (no race condition in  $\mathcal{N}_{normal}^{LU}$ ) and  $curr\_SN' = \{last\_SN'\}$  imply that  $SN = \{last\_SN'\}$ .

1.2. Now assume that  $last\_SN \uparrow \wedge seq_{HE}' = seq_{HE}$ , and let us show:

$$DB'(last\_SN') \neq \epsilon \Rightarrow seq_{HE}' = \underline{Seq}(\max(DB'(last\_SN')))$$

Since

$$last\_SN \uparrow \Rightarrow curr\_SN \uparrow \wedge (\exists_{SN_1} curr\_SN' = \{SN_1\} \vee curr\_SN' = \emptyset)$$

but  $curr\_SN' = \emptyset$  implies  $last\_SN' = last\_SN$ . Choosing a new fresh variable, we conclude that  $curr\_SN \uparrow \wedge curr\_SN' = \{SN_1\}$ .

From  $\mathcal{A}_{normal}^{CurrSN} \wedge \mathcal{A}_{normal}^{LastSN}$  we obtain that LUR  $\vee$  CANLOC and from  $\mathcal{N}_{normal}^{LU}$  we know that CANLOC  $\Rightarrow$  LUR, proving LUR.

Consider now two cases: if  $\neg MvAV$ , then  $DB'(last\_SN') = \epsilon$ , in the other case, if  $MvAV$ , then  $DB'(last\_SN') = DB(last\_SN)$ . In both cases our claim is valid.

1.3. Now assume that  $DB \downarrow \wedge last\_SN' = last\_SN \wedge seq_{HE}' = seq_{HE}$ , and let us show:

$$DB'(last\_SN') \neq \epsilon \Rightarrow seq_{HE}' = \underline{Seq}(\max(DB'(last\_SN')))$$

If  $DB$  changes, but  $seq_{HE}$  does not, then  $UAR_s$  or  $MvAV$  happen ( $\mathcal{N}_{normal}^{SN}$ ). In the first one, only the smallest element of  $DB(last\_SN)$  is taken out, leaving  $DB(last\_SN)$  empty or its largest element unchanged. In both cases our goal is shown.

2. The second goal is that in each transition,

$$seq_{MS}' = \underline{Seq}(\max(Accepted')).$$

This follows easily from the definition of *Accepted* and  $\mathcal{N}_{normal}^{MS}$ .

3. The third goal is that in each transition,

$$\text{UAR}_s(\underline{\text{Chall}}(\text{AV}), \text{SN}) \Rightarrow \text{AV} = \min(\text{DB}(\text{last\_SN})).$$

The proof is similar to the proof of the first goal and uses the fact that  $\ll$  and  $\prec$  coincide: when  $\text{ADS}(\text{AVs\_new}, \text{SN})$  happens, the elements of  $\text{AVs\_new} = \text{DB}'(\text{SN})$  are in order (i.e.:  $\ll \Rightarrow \prec$ ). On each  $\text{UAR}_s$ , the smallest AV is used, and the remaining elements of  $\text{DB}(\text{SN})$  continue in order.

**Lemma 8.4.**

$$\mathcal{A}_{normal} \Rightarrow \Box(\forall_{0 < i < \text{seq}_{HE}} \exists_{\text{AV} \in \text{Gen}} i = \underline{\text{Seq}}(\text{AV}))$$

**Lemma 8.5.** *If the system behaves normally, then the set of generated AVs is the union of the used AVs, the usable ones, and the lost ones:*

$$\mathcal{A}_{normal} \Rightarrow \Box(\text{Gen} = \text{Accepted} \cup \text{DB}(\text{last\_SN}) \cup \text{Lost})$$

A simple consequence of the last two lemmas is:

**Lemma 8.6.**

$$\mathcal{A}_{normal} \Rightarrow \Box(\forall_{\text{seq}_{MS} < i < \underline{\text{Seq}}(\min(\text{DB}(\text{last\_SN}))} \exists_{\text{AV} \in \text{Lost}} i = \underline{\text{Seq}}(\text{AV}))$$

**Lemma 8.7.** *If the system behaves normally, then the set of usable AVs has never more than N elements:*

$$\mathcal{A}_{normal} \Rightarrow \Box(|\text{DB}(\text{last\_SN})| \leq N)$$

**Lemma 8.8.**

$$\mathcal{A}_{normal} \Rightarrow \Box(\text{UAR}_s(\text{chall}, \text{SN}) \Rightarrow \underline{\text{Seq}}(\text{chall}) - \text{seq}_{MS} < \Delta)$$

PROOF: This follows from  $\text{chall} = \min(\text{DB}(\text{SN}))$  and  $(\text{SN}) = \text{last\_SN}$

**Lemma 8.9.**

$$\begin{aligned} \mathcal{N}_{normal}^{CC} \wedge \text{UAR}_s(\underline{\text{Chall}}(\text{AV}), \text{SN}) \Rightarrow \\ \vee \wedge \text{Accepted}' = \text{Accepted} \cup \{\text{AV}\} \\ \wedge \text{seq}_{MS}' = \underline{\text{Seq}}(\text{AV}) \\ \wedge \text{Lost}' = \text{Lost} \\ \vee \wedge \text{Accepted}' = \text{Accepted} \cup \{\text{AV}\} \\ \wedge \text{seq}_{MS}' = \underline{\text{Seq}}(\text{AV}) \\ \wedge \text{Lost}' = \text{Lost} \cup \{\text{AV}\} \\ \vee \wedge \text{Accepted}' = \text{Accepted} \\ \wedge \text{seq}_{MS}' = \text{seq}_{MS} \\ \wedge \text{Lost}' = \text{Lost} \cup \{\text{AV}\} \end{aligned}$$

**Lemma 8.10.**

$$\begin{aligned} \mathcal{N}_{normal}^{CC} \wedge \text{UAR}_s(\underline{\text{Chall}}(\text{AV}), \text{SN}) \wedge \text{UAS}_R(\underline{\text{Resp}}(\text{AV}), \text{SN}) \Rightarrow \\ \wedge \text{Accepted}' = \text{Accepted} \cup \{\text{AV}\} \\ \wedge \text{seq}_{MS}' = \underline{\text{Seq}}(\text{AV}) \\ \wedge \text{Lost}' = \text{Lost} \end{aligned}$$

**Proposition 8.11.** *Initially the system is in perfect conditions. And as long as the system behaves normally, it remains in perfect conditions and no synchronisation failures happen. This may be formalised as follows<sup>4</sup>:*

$$\mathcal{A}_{normal} \Rightarrow \Box(\text{Perfect}) \wedge \Box(\neg \text{UASF}_s)$$

PROOF: First let us see why *Perfect* is an invariant, i.e.  $\mathcal{A}_{normal} \Rightarrow \Box(\text{Perfect})$ . It is clear that initially, *Perfect* holds, i.e.  $\text{Init} \Rightarrow \text{Perfect}$ . Now, if *Perfect* holds and  $\mathcal{N}_{normal}$  holds then also *Perfect'* holds. This follows from Lemma 8.8.

**Proposition 8.12.** *If the system behaves incorrectly, and then behaves normally, then after the first successful Synchronisation Fail Indication the system is in perfect conditions. This is formalised as follows:*

$$\mathcal{A}_{incorr} \wedge \mathcal{A}_{normal}^x \Rightarrow \Box(\text{UASF}_{successful} \wedge x \Rightarrow \text{Perfect}' )$$

**Proposition 8.13.** *If the system behaves critically, and then behaves normally, then after the first successful Synchronisation Fail Indication the system is strongly-synchronous. This is formalised as follows:*

$$\mathcal{A}_{normal}^x \Rightarrow \Box(\text{UASF}_{successful} \wedge x \Rightarrow \text{StrongSynchr}' )$$

**Proposition 8.14.** *If the system strongly-synchronous but not in perfect conditions, and from that point on it behaves normally, then after the stolen AVs have been used or become obsolete, at most one successful Synchronisation Fail Indication is needed to return to perfect conditions.*

**Proposition 8.15.** *If the system weakly-synchronous but not strongly-synchronous, and from that point on it behaves normally, then after the non-synchronous usable AVs have been used or lost, at most one successful Synchronisation Fail Indication is needed to return to a strongly-synchronous state.*

## References

- [1] L. Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3): 872–923, May 1994.
- [2] 3G TS 33.102 Version 3.0.0. *3G Security Architecture*
- [3] S3-99179 (CR to [2]), *Conditions on use of Authentication Information*.
- [4] S3-99180 (CR to [2]), *Modified re-synchronisation procedure for AKA-protocol*.
- [5] S3-99181 (CR to [2]), *Sequence number management scheme protecting against USIM lockout*.

---

<sup>4</sup>The formalisation states something slightly different, namely that if the system always behaves normally, then it is always in perfect conditions and never a synchronisation failure happens. This is slightly weaker than the formulation in the theorem. But the induction proof given in the text also proves the stronger assertion: the proof shows that initially the system is in perfect conditions and that as long as normal conditions hold, the system remains in perfect conditions and no synchronisation failure happens.