Title: Formal Analysis of the 3G Authentication Protocol with Modified Sequence Number Management

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, 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 $\mathsf{Init} = \mathsf{Init}(x)$ 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:

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

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

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

A sequence $(\pi_1, \pi_2, \dots, \pi_n, \dots)$ of states satisfies $\mathcal{A}$ if and only if π_1 satisfies Init, and each pair (π_i, π_{i+1}) satisfies $\mathcal{S}$ *or* π_i is equal to π_{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 $x\updownarrow$ in ($x\updownarrow \Rightarrow \mathcal{S}$): 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

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

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

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

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 $\mathcal{N} :\Leftrightarrow (X_i\updownarrow \Rightarrow \mathcal{S}(\tilde{X}_i, \tilde{X}_i'))$ 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 $n_{\mathtt{In}}(\mathsf{x})$ changes:

$$\text{IN}(\mathsf{x}) \Leftrightarrow n_{\mathtt{In}}(\mathsf{x})\updownarrow$$

Typically, the next step relations $\mathcal{N}$ that we will use are of the form[1]:

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

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

$$\mathcal{N} :\Leftrightarrow \text{OUT}(\mathsf{y}) \Rightarrow \text{IN} \qquad \text{or :} \qquad \mathcal{N} :\Leftrightarrow y\updownarrow \Rightarrow \text{IN}$$

(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 $\mathcal{N} :\Leftrightarrow \text{IN}(\mathsf{x}) \wedge \mathit{cond} \Rightarrow \ldots$ is equivalent to $\mathcal{N} :\Leftrightarrow \forall_{\mathsf{x}}\ \text{IN}(\mathsf{x}) \wedge \mathit{cond} \Rightarrow \ldots$.

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.

[1] If $\mathcal{S}$ is of the form $\mathit{cond} \Rightarrow \mathcal{S}_1$, then $[\mathcal{S}]_x$ is equivalent to $x\updownarrow \wedge \mathit{cond} \Rightarrow \mathcal{S}_1$.

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
$\mathcal{SEQ} := \{1, \dots, \mathsf{MaxSEQ}\} \subseteq \mathbb{N}$
= 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
$\mathcal{FAIL} := \{\mathsf{Loss}, \mathsf{DB}, \mathsf{Crash}, \mathsf{Steal}, \mathsf{Race}\}$

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:

MaxSEQ = the maximal possible value for the sequence number.
Δ = 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 Δ, we will assume: $\mathsf{N} < \Delta/2$.

In the specification in [2], (together with the CRs [3], [4],and [5] accepted at 3GPPSA3 London meeting) an AV is a tuple ("vector") $\mathsf{AV} = (\mathsf{Rand}, \mathsf{resp}, CK, IK, \mathsf{seq} \oplus AK, MODE, MAC)$. The only values that we are explicitly interested in are: Rand, resp and 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 Rand, resp and seq are components of AV, we assume the existence of functions: $\underline{Rand}_{\mathsf{AV}} : \mathsf{AV} \mapsto \mathsf{Rand}$, $\underline{Resp}_{\mathsf{AV}} : \mathsf{AV} \mapsto \mathsf{resp}$, and $\underline{Seq}_{\mathsf{AV}} : \mathsf{AV} \mapsto \mathsf{seq}$. The service network, SN receives from the home environment the complete AV and sends to MS, the mobile station, $\mathsf{chall} := \underline{Chall}(\mathsf{AV})$ (in the original specification, chall is part of the authentication vector: $= (\mathsf{Rand}, \mathsf{seq} \oplus AK, MODE, MAC)$). The MS is able to check the consistency of the challenge chall and to calculate the original parameter resp of the AV. Thus we assume the existence of functions: $cons_{chall}$ (to check the consistency of chall) and $\underline{Resp}_{chall}$ (to calculate resp, given chall). (Those functions, and most of the ones that we will introduce now, depend on the secret key K; the function $cons_{chall}$ depends also in particular on the parameter MAC). Further, the 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 $cons_{\mathsf{AV}}$(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 AVs 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:

$$\underline{Seq}_{\mathsf{AV}} : \mathcal{AV} \to \mathcal{SEQ}$$
$$\underline{Seq}_{ch} : \mathcal{CHAL} \to \mathcal{SEQ}$$
$$\underline{Seq}_{MS} : \mathcal{RESP}_S \to \mathcal{SEQ}$$
$$\underline{Chall} : \mathcal{AV} \to \mathcal{CHAL}$$
$$\underline{Resp}_{\mathsf{AV}} : \mathcal{AV} \to \mathcal{RESP}_A$$
$$\underline{Resp}_{ch} : \mathcal{CHAL} \to \mathcal{RESP}_A$$
$$\underline{Rand}_{\mathsf{AV}} : \mathcal{AV} \to \mathcal{RAND}$$
$$\underline{Rand}_{ch} : \mathcal{CHAL} \to \mathcal{RAND}$$
$$\underline{RejResp} : \mathcal{CHAL} \to \mathcal{RESP}_J$$
$$\underline{SynResp} : \mathcal{CHAL} \times \mathcal{CHAL} \to \mathcal{RESP}_S$$
$$cons_{\mathsf{AV}} : \mathcal{AV} \to \mathbb{B}$$
$$cons_{chall} : \mathcal{CHAL} \to \mathbb{B}$$
$$cons_{res} : \mathcal{CHAL} \times \mathcal{RESP}_A \to \mathbb{B}$$
$$synchr : \mathcal{SEQ} \times \mathcal{SEQ} \to \mathbb{B}$$
$$verif : \mathcal{RESP}_S \times \mathcal{RAND} \to \mathbb{B}$$

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

$$\mathsf{AV}_1 \prec \mathsf{AV}_2 :\Leftrightarrow \underline{Seq}_{\mathsf{AV}}(\mathsf{AV}_1) < \underline{Seq}_{\mathsf{AV}}(\mathsf{AV}_2)$$

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

$$\underline{Seq}_{ch} \circ \underline{Chall} = \underline{Seq}_{\mathsf{AV}}$$
$$\underline{Resp}_{ch} \circ \underline{Chall} = \underline{Resp}_{\mathsf{AV}}$$
$$\underline{Rand}_{ch} \circ \underline{Chall} = \underline{Rand}_{\mathsf{AV}}$$

At its proper place, in Sections 4 and 6 (in the definitions of $\mathcal{N}^{HE}_{normal}$ and $\mathcal{N}^{HE}_{critical}$), 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.

$$cons_{\mathsf{AV}}(\mathsf{AV}) \Rightarrow cons_{chall}(\underline{Chall}(\mathsf{AV})) \wedge cons_{res}(\underline{Chall}(\mathsf{AV}), \underline{Resp}(\mathsf{AV}))$$

$$cons_{res}(\mathsf{chall}, \mathsf{resp}) \wedge \mathsf{resp}_1 \neq \mathsf{resp} \Rightarrow \neg cons_{res}(\mathsf{chall}, \mathsf{resp}_1)$$

$$verif(\mathsf{SynResp}, \mathsf{Rand}) \Leftrightarrow$$

$$\exists_{\mathsf{AV},\mathsf{chall}_1}\ \mathsf{SynResp} = \underline{SynResp}(\mathsf{chall}_1, \underline{Chall}(\mathsf{AV})) \wedge\ \mathsf{Rand} = \underline{Rand}(\mathsf{AV})$$

$$\underline{Seq}_{MS}(\underline{SynResp}(\mathsf{chall}_1, \mathsf{chall}_2)) = \underline{Seq}_{ch}(\mathsf{chall}_1)$$

In the sequel, if no confusion arises, we omit the subscripts, writing $\underline{Seq}$ instead of $\underline{Seq}_{ch}, \underline{Seq}_{\mathsf{AV}}$ or $\underline{Seq}_{MS}$, $\underline{Resp}$ instead of $\underline{Resp}_{ch}$ or $\underline{Resp}_{\mathsf{AV}}$, $\underline{Rand}$ instead of $\underline{Rand}_{ch}$ or $\underline{Rand}_{\mathsf{AV}}$ and *cons* instead of $cons_{\mathsf{AV}}$, $cons_{chall}$ 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* $\mathsf{AV}_1\ \mathsf{AV}_2\ \mathsf{AV}_3\ \ldots \mathsf{AV}_n$ built with "letters" from $\mathcal{AV}$: $\mathsf{AV}_i \in \mathcal{AV}$. The basic operations in this domain are: $Head : \mathcal{AV}^* \rightarrow \mathcal{AV}$ that chooses the first letter of the word: $Head(\mathsf{AV}_1\ \mathsf{AV}_2\ \mathsf{AV}_3\ \ldots \mathsf{AV}_n) = \mathsf{AV}_1$ and $Tail : \mathcal{AV}^* \rightarrow \mathcal{AV}^*$ is the rest of word: $Tail(\mathsf{AV}_1\ \mathsf{AV}_2\ \mathsf{AV}_3\ \ldots \mathsf{AV}_n) = (\mathsf{AV}_2\ \mathsf{AV}_3\ \ldots \mathsf{AV}_n)$. ϵ 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:

$seq_{HE} : \mathcal{SEQ}$

$DB : \mathcal{SN} \rightarrow \mathcal{AV}^*$ called the database of AVs

$la\_chall : \mathcal{CHAL}$ the last accepted challenge, that is, the last challenge accepted by the MS

Thus, for SN in $\mathcal{SN}$, the database $DB(\mathsf{SN})$ of AVs stored in the node SN is a *word* $\mathsf{AV}_1\ \mathsf{AV}_2\ \mathsf{AV}_3\ \ldots \mathsf{AV}_n$ of authentication vectors AV_i. We will denote by $Set(\omega)$ the set $\{\mathsf{AV}_1, \mathsf{AV}_2, \mathsf{AV}_3, \ldots \mathsf{AV}_n\}$ of letters in the word $\omega = \mathsf{AV}_1\ \mathsf{AV}_2\ \mathsf{AV}_3\ \ldots \mathsf{AV}_n$. By abuse of notation we will use sometimes $DB(\mathsf{SN})$ in a context where a *set* is expected instead of a *word*, meaning : $Set(DB(\mathsf{SN}))$. Thus $\ldots \cup DB(\mathsf{SN})$ is $\ldots \cup Set(DB(\mathsf{SN}))$ and $\ldots \subseteq DB(\mathsf{SN})$ is $\ldots \subseteq Set(DB(\mathsf{SN}))$.

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

$Gen : \wp(\mathcal{AV})$ the set of generated AVs

$Used : \wp(\mathcal{AV})$ the set of used (sent) AVs

$Stolen_{HE} : \wp(\mathcal{AV})$ the set of AVs that were stolen in the HE

$Stolen_{SN} : \wp(\mathcal{AV})$ the set of AVs that were stolen in an SN

$Stolen : \wp(\mathcal{AV})$ the set of stolen AVs

$Accepted : \wp(\mathcal{AV})$ the set of AVs accepted by the MS

$Succsessful : \wp(\mathcal{AV})$ the set of AVs accepted by the MS and correctly replied

$Lost : \wp(\mathcal{AV})$ the set of lost AVs

$last\_SN : \mathcal{SN}$ in normal behavior, the SN where the user is registered.

$curr\_SN : \wp(\mathcal{SN})$ the current *set of* SNs where the user is registered in normal behavior, it is $\{last\_SN\}$, in incorrect behavior it may contain no or several SNs.

Definition 3.1. *The state functions are:*

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

Definition 3.2.

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

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 $n_{\mathtt{X}}$ that changes exactly when MESSAGE_X occurs. For convenience, if MESSAGE_X is a message with parameters of types $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_n$ we take the variable $n_{\mathtt{X}}$ to be of type $\mathcal{X}_1 \times \mathcal{X}_2 \times \ldots \mathcal{X}_n \to \mathbb{N}$. $n_{\mathtt{X}}(\mathsf{x}_1, \mathsf{x}_2, \ldots \mathsf{x}_n)$ may for instance count (in $\mathbb{N}$ or modulo a convenient number) how often the message (or event) MESSAGE_X with parameters $\mathsf{x}_1, \mathsf{x}_2, \ldots \mathsf{x}_n$ happens. Instead of using the variable $n_{\mathtt{X}}$ in our specification, we introduce a new predicate, say $\mathrm{X}(\mathsf{x}_1, \mathsf{x}_2, \ldots \mathsf{x}_n)$, which is an abbreviation:

$$\mathrm{X}(\mathsf{x}_1, \mathsf{x}_2, \ldots \mathsf{x}_n) :\Leftrightarrow n_{\mathtt{X}}(\mathsf{x}_1, \mathsf{x}_2, \ldots \mathsf{x}_n)\updownarrow$$

(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: $\mathrm{UAR_S}$) and USER AUTHENT. REQUEST RECEIVE ($\mathrm{UAR_R}$):

1. Normally, if a $\mathrm{UAR_S}$ happens, then a $\mathrm{UAR_R}$ also happens, with the same parameters.
2. Due to an attack, the $\mathrm{UAR_R}$ may contain different parameters than the corresponding Send.
3. A sent $\mathrm{UAR_S}$ may get lost in the transmission channel, thus producing no $\mathrm{UAR_R}$ event.
4. Due to an attack, the USIM may receive a $\mathrm{UAR_R}$ 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.

Message	*Trans. Pred.*	*Param. Name*	*Param. Type*	*Var. Name*
AUTHENT. DATA REQUEST (no syn. fail)	ADR_0	SN	$\mathcal{SN}$	$n_{\mathtt{ADR,0}}$
AUTHENT. DATA REQUEST (syn. fail)	ADR_1	(SynResp, Rand, SN)	($\mathcal{RESP}_S$, $\mathcal{RAND}$, $\mathcal{SN}$)	$n_{\mathtt{ADR,1}}$
AUTHENT. DATA RESPONSE	ADS	(AVs_new, SN)	($\mathcal{AV}^*$, $\mathcal{SN}$)	$n_{\mathtt{ADS}}$
USER AUTHENT. REQUEST	UAR_S	(chall, SN)	($\mathcal{CHAL}$ $\mathcal{SN}$)	$n_{\mathtt{UAR,s}}$
	UAR_R	(chall, SN)	($\mathcal{CHAL}$ $\mathcal{SN}$)	$n_{\mathtt{UAR,r}}$
USER AUTHENT. RESPONSE	UAS_S	(resp, SN)	($\mathcal{RESP}_A$ $\mathcal{SN}$)	$n_{\mathtt{UAS,s}}$
	UAS_R	(resp, SN)	($\mathcal{RESP}_A$ $\mathcal{SN}$)	$n_{\mathtt{UAS,r}}$
USER AUTHENT. REJECT	UAJ_S	(RejResp, SN)	($\mathcal{RESP}_J$ $\mathcal{SN}$)	$n_{\mathtt{UAJ,s}}$
	UAJ_R	(RejResp, SN)	($\mathcal{RESP}_J$ $\mathcal{SN}$)	$n_{\mathtt{UAJ,r}}$
USER AUTHENT. SYNCHRON. FAIL INDICATION	UASF_S	(SynResp, SN)	($\mathcal{RESP}_S$ $\mathcal{SN}$)	$n_{\mathtt{UASF,s}}$
	UASF_R	(SynResp, SN)	($\mathcal{RESP}_S$ $\mathcal{SN}$)	$n_{\mathtt{UASF,r}}$

Table 1: Messages of the Protocol

Message or Event	*Trans. Pred.*	*Param. Name*	*Param. Type*	*Var. Name*
LOCATION UPDATE REQUEST	LUR	SN SN_1	$\mathcal{SN}$ $\mathcal{SN}$	$n_{\mathtt{LUR,s}}$
CANCEL LOCATION	$\mathrm{CANLOC}_\mathrm{S}$ $\mathrm{CANLOC}_\mathrm{R}$	SN SN	$\mathcal{SN}$ $\mathcal{SN}$	$n_{\mathtt{CanLoc,s}}$ $n_{\mathtt{CanLoc,r}}$
MOVE AVS (SEND ID REQ and SEND ID RESP)	MVAV	SN SN_1	$\mathcal{SN}$ $\mathcal{SN}$	$n_{\mathtt{MvAV}}$
TIME OUT	TIO	SN	$\mathcal{SN}$	$n_{\mathtt{TiO}}$

Table 2: Messages or Events outside of the protocol

Definition 3.3. (The Messages)

$$\mathrm{ADR}_0(\mathsf{SN}) :\Leftrightarrow n_{\mathtt{ADR,0}}(\mathsf{SN})\updownarrow$$
$$\mathrm{ADR}_1(\mathsf{SynResp},\mathsf{Rand},\mathsf{SN}) :\Leftrightarrow n_{\mathtt{ADR,1}}(\mathsf{SynResp},\mathsf{Rand},\mathsf{SN})\updownarrow$$
$$\mathrm{ADS}(\mathsf{AVs\_new},\mathsf{SN}) :\Leftrightarrow n_{\mathtt{ADS}}(\mathsf{AVs\_new},\mathsf{SN})\updownarrow$$
$$\mathrm{UAR}_{\mathrm{S}}(\mathsf{chall},\mathsf{SN}) :\Leftrightarrow n_{\mathtt{UAR,s}}(\mathsf{chall},\mathsf{SN})\updownarrow$$
$$\mathrm{UAR}_{\mathrm{R}}(\mathsf{chall},\mathsf{SN}) :\Leftrightarrow n_{\mathtt{UAR,r}}(\mathsf{chall},\mathsf{SN})\updownarrow$$
$$\mathrm{UAS}_{\mathrm{S}}(\mathsf{resp},\mathsf{SN}) :\Leftrightarrow n_{\mathtt{UAS,s}}(\mathsf{resp},\mathsf{SN})\updownarrow$$
$$\mathrm{UAS}_{\mathrm{R}}(\mathsf{resp},\mathsf{SN}) :\Leftrightarrow n_{\mathtt{UAS,r}}(\mathsf{resp},\mathsf{SN})\updownarrow$$
$$\mathrm{UAJ}_{\mathrm{S}}(\mathsf{RejResp},\mathsf{SN}) :\Leftrightarrow n_{\mathtt{UAJ,s}}(\mathsf{RejResp},\mathsf{SN})\updownarrow$$
$$\mathrm{UAJ}_{\mathrm{R}}(\mathsf{RejResp},\mathsf{SN}) :\Leftrightarrow n_{\mathtt{UAJ,r}}(\mathsf{RejResp},\mathsf{SN})\updownarrow$$
$$\mathrm{UASF}_{\mathrm{S}}(\mathsf{SynResp},\mathsf{SN}) :\Leftrightarrow n_{\mathtt{UASF,s}}(\mathsf{SynResp},\mathsf{SN})\updownarrow$$
$$\mathrm{UASF}_{\mathrm{R}}(\mathsf{SynResp},\mathsf{SN}) :\Leftrightarrow n_{\mathtt{UASF,r}}(\mathsf{SynResp},\mathsf{SN})\updownarrow$$
$$\mathrm{LUR}(\mathsf{SN},\mathsf{SN}_1) :\Leftrightarrow n_{\mathtt{LUR}}(\mathsf{SN},\mathsf{SN}_1)\updownarrow$$
$$\textsc{CanLoc}_{\mathrm{S}}(\mathsf{SN}) :\Leftrightarrow n_{\mathtt{CanLoc,s}}(\mathsf{SN})\updownarrow$$
$$\textsc{CanLoc}_{\mathrm{R}}(\mathsf{SN}) :\Leftrightarrow n_{\mathtt{CanLoc,r}}(\mathsf{SN})\updownarrow$$
$$\textsc{MvAV}(\mathsf{SN},\mathsf{SN}_1) :\Leftrightarrow n_{\mathtt{MvAV}}(\mathsf{SN},\mathsf{SN}_1)\updownarrow$$
$$\textsc{TiO}(\mathsf{SN}) :\Leftrightarrow n_{\mathtt{TiO}}(\mathsf{SN})\updownarrow$$

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

Convention 3.1.

$$\mathrm{ADR}_0 :\Leftrightarrow \exists_{\mathsf{SN}} \quad \mathrm{ADR}_0(\mathsf{SN})$$
$$\mathrm{ADR}_1 :\Leftrightarrow \exists_{\mathsf{SynResp,Rand,SN}} \quad \mathrm{ADR}_1(\mathsf{SynResp},\mathsf{Rand},\mathsf{SN})$$
$$\mathrm{ADS} :\Leftrightarrow \exists_{\mathsf{AVs\_new,SN}} \quad \mathrm{ADS}(\mathsf{AVs\_new},\mathsf{SN})$$
$$\ldots$$

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

Convention 3.2.

$$\mathrm{ADR}_1(\mathsf{SN}) :\Leftrightarrow \exists_{\mathsf{SynResp,Rand}} \quad \mathrm{ADR}_1(\mathsf{SynResp},\mathsf{Rand},\mathsf{SN})$$
$$\mathrm{ADS}(\mathsf{SN}) :\Leftrightarrow \exists_{\mathsf{AVs\_new}} \quad \mathrm{ADS}(\mathsf{AVs\_new},\mathsf{SN})$$
$$\ldots$$

We will not really use the variables $n_{\mathtt{ADR,0}}(\mathsf{SN})$, 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: $x\updownarrow\Rightarrow \mathcal{A}$.
Notice also that $x\updownarrow \wedge \mathcal{P} \Rightarrow \mathcal{A}$ 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: $\mathrm{X}(\mathsf{x}_1,\mathsf{x}_2,\ldots\mathsf{x}_n) \wedge \mathrm{X}(\mathsf{y}_1,\mathsf{y}_2,\ldots\mathsf{y}_n) \wedge (\mathsf{x}_1,\mathsf{x}_2,\ldots\mathsf{x}_n) \neq$

$(\mathsf{y}_1, \mathsf{y}_2, \ldots \mathsf{y}_n)$:

$\mathcal{A}^{interleave}_{normal} :\Leftrightarrow$

$\Box(\forall_{\mathsf{SN},\mathsf{SynResp},\mathsf{Rand},\mathsf{AVs\_new},\mathsf{chall},\mathsf{resp},\mathsf{RejResp},\mathsf{SynResp}}$

$\forall_{\mathsf{SN}_1,\mathsf{SynResp}_1,\mathsf{Rand}_1,\mathsf{AVs\_new}_1,\mathsf{chall}_1,\mathsf{resp}_1,\mathsf{RejResp}_1,\mathsf{SynResp}_1}$

$\wedge\ \mathrm{ADR}_0(\mathsf{SN}) \wedge \mathrm{ADR}_0(\mathsf{SN}_1) \Rightarrow \mathsf{SN} = \mathsf{SN}_1$

$\wedge\ \mathrm{ADR}_1(\mathsf{SynResp}, \mathsf{Rand}, \mathsf{SN}) \wedge \mathrm{ADR}_1(\mathsf{SynResp}_1, \mathsf{Rand}_1, \mathsf{SN}_1)$

$\Rightarrow \mathsf{SynResp} = \mathsf{SynResp}_1 \wedge \mathsf{Rand} = \mathsf{Rand}_1 \wedge \mathsf{SN} = \mathsf{SN}_1$

$\wedge\ \mathrm{ADS}(\mathsf{AVs\_new}, \mathsf{SN}) \wedge \mathrm{ADS}(\mathsf{AVs\_new}_1, \mathsf{SN}_1)$

$\Rightarrow \mathsf{AVs\_new} = \mathsf{AVs\_new}_1 \wedge \mathsf{SN} = \mathsf{SN}_1$

$\wedge\ \mathrm{UAR}_{\mathrm{S}}(\mathsf{chall}, \mathsf{SN}) \wedge \mathrm{UAR}_{\mathrm{S}}(\mathsf{chall}_1, \mathsf{SN}_1)$

$\Rightarrow \mathsf{chall} = \mathsf{chall}_1 \wedge \mathsf{SN} = \mathsf{SN}_1$

$\wedge\ \mathrm{UAR}_{\mathrm{R}}(\mathsf{chall}, \mathsf{SN}) \wedge \mathrm{UAR}_{\mathrm{R}}(\mathsf{chall}_1, \mathsf{SN}_1)$

$\Rightarrow \mathsf{chall} = \mathsf{chall}_1 \wedge \mathsf{SN} = \mathsf{SN}_1$

$\wedge\ \mathrm{UAS}_{\mathrm{S}}(\mathsf{resp}, \mathsf{SN}) \wedge \mathrm{UAS}_{\mathrm{S}}(\mathsf{resp}_1, \mathsf{SN}_1)$

$\Rightarrow \mathsf{resp} = \mathsf{resp}_1 \wedge \mathsf{SN} = \mathsf{SN}_1$

$\wedge\ \mathrm{UAS}_{\mathrm{R}}(\mathsf{resp}, \mathsf{SN}) \wedge \mathrm{UAS}_{\mathrm{R}}(\mathsf{resp}_1, \mathsf{SN}_1)$

$\Rightarrow \mathsf{resp} = \mathsf{resp}_1 \wedge \mathsf{SN} = \mathsf{SN}_1$

$\wedge\ \mathrm{UAJ}_{\mathrm{S}}(\mathsf{RejResp}, \mathsf{SN}) \wedge \mathrm{UAJ}_{\mathrm{S}}(\mathsf{RejResp}_1, \mathsf{SN}_1)$

$\Rightarrow \mathsf{RejResp} = \mathsf{RejResp}_1 \wedge \mathsf{SN} = \mathsf{SN}_1$

$\wedge\ \mathrm{UAJ}_{\mathrm{R}}(\mathsf{RejResp}, \mathsf{SN}) \wedge \mathrm{UAJ}_{\mathrm{R}}(\mathsf{RejResp}_1, \mathsf{SN}_1)$

$\Rightarrow \mathsf{RejResp} = \mathsf{RejResp}_1 \wedge \mathsf{SN} = \mathsf{SN}_1$

$\wedge\ \mathrm{UASF}_{\mathrm{S}}(\mathsf{SynResp}, \mathsf{SN}) \wedge \mathrm{UASF}_{\mathrm{S}}(\mathsf{SynResp}_1, \mathsf{SN}_1)$

$\Rightarrow \mathsf{SynResp} = \mathsf{SynResp}_1 \wedge \mathsf{SN} = \mathsf{SN}_1$

$\wedge\ \mathrm{UASF}_{\mathrm{R}}(\mathsf{SynResp}, \mathsf{SN}) \wedge \mathrm{UASF}_{\mathrm{R}}(\mathsf{SynResp}_1, \mathsf{SN}_1)$

$\Rightarrow \mathsf{SynResp} = \mathsf{SynResp}_1 \wedge \mathsf{SN} = \mathsf{SN}_1$

$\wedge\ \mathrm{UAS}_{\mathrm{S}} \Rightarrow \neg\mathrm{UAJ}_{\mathrm{S}} \wedge\ \neg\mathrm{UASF}_{\mathrm{S}}$

$\wedge\ \mathrm{UAJ}_{\mathrm{S}} \Rightarrow \neg\mathrm{UAS}_{\mathrm{S}} \wedge\ \neg\mathrm{UASF}_{\mathrm{S}}$

$\wedge\ \mathrm{UASF}_{\mathrm{S}} \Rightarrow \neg\mathrm{UAS}_{\mathrm{S}} \wedge\ \neg\mathrm{UAJ}_{\mathrm{S}}\ )$

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:

$\mathcal{A}^{CurrSN}_{normal} :\Leftrightarrow$

$\wedge\ \exists_{\mathsf{SN}}\ curr\_SN = \{\mathsf{SN}\}$

$\Box(\ \wedge\ curr\_SN\updownarrow \Rightarrow \mathrm{LUR} \vee\ \textsc{CanLoc}$

$\wedge\ \mathrm{LUR}(\mathsf{SN}, \mathsf{SN}_1) \wedge\ \textsc{CanLoc}(\mathsf{SN}_2) \Rightarrow$

$curr\_SN' = (curr\_SN \setminus \{\mathsf{SN}_2\}) \cup \{\mathsf{SN}_1\}$

$\wedge\ \mathrm{LUR}(\mathsf{SN}, \mathsf{SN}_1) \wedge\ \neg\textsc{CanLoc} \Rightarrow$

$curr\_SN' = curr\_SN \cup \{\mathsf{SN}_1\}$

$\wedge\ \textsc{CanLoc}(\mathsf{SN}) \wedge\ \neg\mathrm{LUR} \Rightarrow$

$curr\_SN' = (curr\_SN \setminus \{\mathsf{SN}\})\ )$

The intuition is that, under ideal[2] behavior, a LUR($\mathsf{SN}, \mathsf{SN}_1$) implies a CanLoc($\mathsf{SN}$). In this case, in the set of current SNs the old SN, SN, is replaced by the new one, SN_1: $curr\_SN' = (curr\_SN \setminus \{\mathsf{SN}\}) \cup \{\mathsf{SN}_1\}$. Thus $curr\_SN$ would always be a singleton. We will allow *in normal conditions* that a CanLoc(SN) occurs without a LUR($\mathsf{SN}, \mathsf{SN}_1$), 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($\mathsf{SN}, \mathsf{SN}_1$) has not been followed by a CanLoc(SN).

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

$$
\begin{aligned}
&\mathcal{N}^{LU}_{normal} :\Leftrightarrow \\
&\quad \wedge\ curr\_SN' = \emptyset \vee\ \exists_{SN_0}\ curr\_SN' = \{\mathsf{SN}_0\} \\
&\quad \wedge\ \mathrm{LUR}(\mathsf{SN}, \mathsf{SN}_1) \Rightarrow \mathrm{MvAV}(\mathsf{SN}, \mathsf{SN}_1) \vee DB'(\mathsf{SN}_1) = \epsilon \\
&\quad \wedge\ \mathrm{MvAV}(\mathsf{SN}, \mathsf{SN}_1) \Rightarrow \mathrm{LUR}(\mathsf{SN}, \mathsf{SN}_1) \\
&\quad \wedge\ \mathrm{ADS} \Rightarrow curr\_SN' = curr\_SN \\
&\quad \wedge\ \mathrm{MvAV}(\mathsf{SN}, \mathsf{SN}_1) \Rightarrow \wedge\ DB'(\mathsf{SN}_1) = DB(\mathsf{SN}) \\
&\qquad\qquad\qquad\qquad\ \wedge\ DB'(\mathsf{SN}) = \epsilon \\
&\qquad\qquad\qquad\qquad\ \wedge \forall_{\mathsf{SN}_2}\ (\mathsf{SN}_2 \neq \mathsf{SN} \wedge \mathsf{SN}_2 \neq \mathsf{SN}_1 \Rightarrow \\
&\qquad\qquad\qquad\qquad\qquad\qquad DB'(\mathsf{SN}_2) = DB'(\mathsf{SN}_2))
\end{aligned}
$$

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, $\exists_{SN_0}\ curr\_SN' = \{\mathsf{SN}_0\}$), 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 $DB'(\mathsf{SN}_1) = \epsilon$. If MvAV happens, then $DB'(\mathsf{SN}_1) = DB(\mathsf{SN})$. 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:

$$
\begin{aligned}
&\mathcal{A}^{LastSN}_{normal} :\Leftrightarrow \\
&\quad \wedge\ curr\_SN = \{last\_SN\} \\
&\quad \wedge \Box(\ \wedge\ last\_SN\updownarrow \Rightarrow curr\_SN\updownarrow \\
&\qquad\ \wedge\ \forall_{\mathsf{SN}_0}\ (curr\_SN\updownarrow \wedge\ curr\_SN' = \{\mathsf{SN}_0\} \Rightarrow last\_SN' = \mathsf{SN}_0) \\
&\qquad\ \wedge\ (curr\_SN\updownarrow \wedge\ \forall_{\mathsf{SN}_0}\ curr\_SN' \neq \{\mathsf{SN}_0\}) \Rightarrow last\_SN' = last\_SN\ )
\end{aligned}
$$

Notice that we in the context of $\Box\mathcal{N}^{LU}_{normal}$ the predicate $\forall_{\mathsf{SN}_0}\ curr\_SN' \neq \{\mathsf{SN}_0\}$ 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

[2]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., $\mathrm{ADR}_0 \Rightarrow DB(\mathsf{SN}) = \epsilon$.

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 ($DB(\mathsf{SN})$).

When the **SN** sends a USER AUTHENT. REQUEST SEND, it updates the database of AVs by deleting the first AV in the list $DB(\mathsf{SN})$. (This AV is now the "current AV", and the values are used to compare with the expected response, $\mathrm{UAS}_\mathrm{R}(\mathsf{resp},\mathsf{SN})$). 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}^{SN}_{normal} :\Leftrightarrow \\
&\quad \wedge\ \mathrm{ADR}_0(\mathsf{SN}) \Rightarrow DB(\mathsf{SN}) = \epsilon \wedge \mathsf{SN} \in curr\_SN \\
&\quad \wedge\ \mathrm{ADR}_1(\mathsf{SynResp},\mathsf{Rand},\mathsf{SN}) \Rightarrow \\
&\qquad\qquad \wedge\ \mathrm{UASF}_\mathrm{R}(\mathsf{SynResp},\mathsf{SN}) \\
&\qquad\qquad \wedge\ \mathsf{SN} \in curr\_SN \\
&\qquad\qquad \wedge\ \exists_{\mathsf{AV}}\ (\mathrm{UAR}_\mathrm{S}(\underline{Chall}(\mathsf{AV}),\mathsf{SN}) \wedge\ \mathsf{Rand} = \underline{Rand}(\mathsf{AV})) \\
&\quad \wedge\ \mathrm{ADS}(\mathsf{AVs\_new},\mathsf{SN}) \Rightarrow \wedge\ \mathsf{AVs\_new} \neq \epsilon \Rightarrow DB'(\mathsf{SN}) = \mathsf{AVs\_new} \\
&\qquad\qquad\qquad\qquad\qquad \wedge\ \mathsf{AVs\_new} = \epsilon \Rightarrow DB'(\mathsf{SN}) = DB(\mathsf{SN}) \\
&\quad \wedge\ \mathrm{UAR}_\mathrm{S}(\mathsf{chall},\mathsf{SN}) \Rightarrow \wedge\ \mathsf{SN} \in curr\_SN \wedge DB(\mathsf{SN}) \neq \epsilon \\
&\qquad\qquad\qquad\qquad\qquad\quad \wedge\ DB'(\mathsf{SN}) = Tail(DB(\mathsf{SN})) \\
&\qquad\qquad\qquad\qquad\qquad\quad \wedge\ \mathsf{chall} = \underline{Chall}(Head(DB(\mathsf{SN}))) \\
&\quad \wedge\ \mathrm{UAR}_\mathrm{S}(\mathsf{chall},\mathsf{SN}) \Rightarrow \vee\ \exists_{\mathsf{resp}}\ \mathrm{UAS}_\mathrm{R}(\mathsf{resp},\mathsf{SN}) \\
&\qquad\qquad\qquad\qquad\qquad\quad \vee\ \exists_{\mathsf{RejResp}}\ \mathrm{UAJ}_\mathrm{R}(\mathsf{RejResp},\mathsf{SN}) \\
&\qquad\qquad\qquad\qquad\qquad\quad \vee\ \exists_{\mathsf{SynResp}}\ \mathrm{UASF}_\mathrm{R}(\mathsf{SynResp},\mathsf{SN}) \\
&\qquad\qquad\qquad\qquad\qquad\quad \vee\ \mathrm{TIO} \\
&\quad \wedge\ DB(\mathsf{SN})\Updownarrow \Rightarrow \vee\ \exists_{\mathsf{AVs\_new},\mathsf{SN}}\ \ \mathrm{ADS}(\mathsf{AVs\_new},\mathsf{SN}) \wedge \mathsf{AVs\_new} \neq \epsilon \\
&\qquad\qquad\qquad\qquad\quad \vee\ \exists_{\mathsf{chall},\mathsf{SN}}\ \ \mathrm{UAR}_\mathrm{S}(\mathsf{chall},\mathsf{SN}) \\
&\qquad\qquad\qquad\qquad\quad \vee\ \exists_{\mathsf{SN}_1}\ \ \mathrm{MVAV}(\mathsf{SN},\mathsf{SN}_1) \\
&\qquad\qquad\qquad\qquad\quad \vee\ \exists_{\mathsf{SN}_1}\ \ \mathrm{MVAV}(\mathsf{SN}_1,\mathsf{SN}) \\
&\quad \wedge\ [\ \wedge \mathrm{UASF}_\mathrm{R}(\mathsf{SynResp},\mathsf{SN}) \\
&\qquad \wedge \mathsf{SN} \in curr\_SN \\
&\qquad \wedge \mathrm{UAR}_\mathrm{S}(\underline{Chall}(\mathsf{AV}),\mathsf{SN}) \\
&\qquad \wedge \mathsf{Rand} = \underline{Rand}(\mathsf{AV})\ ] \Rightarrow \mathrm{ADR}_1(\mathsf{SynResp},\mathsf{Rand},\mathsf{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 $\mathcal{N}^{HE}_{normal}$ as the sequential composition of two "micro-steps":

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

Notice, by the way, that if $\mathcal{N}^{HE}_{normal} \wedge (\mathrm{ADR}_0 \vee \mathrm{ADR}_1)$ the value of seq_{HE} uniquely determines the value of seq_{he}.

Recall that $\mathsf{AVs\_new}$ is a word (or ordered sequence) of AVs. We write $\mathsf{AV}_1 \ll_{\mathsf{AVs\_new}} \mathsf{AV}_2$ iff both AV_1 and AV_1 appear in $\mathsf{AVs\_new}$ and AV_1 appears (anywhere) *left* of AV_2.
Here we write $\ll$ instead of $\ll_{\mathsf{AVs\_new}}$.

$$
\begin{aligned}
&\mathcal{N}^{HE}_{normal} :\Leftrightarrow \exists_{seq_{he}} \; [\\
&\quad \wedge \; \mathrm{ADS}(\mathsf{AVs\_new}, \mathsf{SN}) \Rightarrow \vee \; \mathrm{ADR}_0(\mathsf{SN}) \\
&\qquad\qquad\qquad\qquad \vee \; \exists_{\mathsf{SynResp},\mathsf{Rand}} \; \mathrm{ADR}_1(\mathsf{SynResp}, \mathsf{Rand}, \mathsf{SN}) \\
&\quad \wedge \; \mathrm{ADR}_0 \vee \; \mathrm{ADR}_1 \Rightarrow \mathrm{ADS} \\
&\quad \wedge \; \mathrm{ADR}_0 \Rightarrow seq_{he} = seq_{HE} \\
&\quad \wedge \; \mathrm{ADR}_1(\mathsf{SynResp}, \mathsf{Rand}, \mathsf{SN}) \Rightarrow \\
&\qquad \wedge \; [verif(\mathsf{SynResp}, \mathsf{Rand}) \wedge \\
&\qquad\qquad \neg synchr_1(\underline{Seq}_{MS}(\mathsf{SynResp}), seq_{HE})] \Rightarrow seq_{he} = \underline{Seq}_{MS}(\mathsf{SynResp}) \\
&\qquad \wedge \; [\neg verif(\mathsf{SynResp}, \mathsf{Rand}) \vee \\
&\qquad\qquad synchr_1(\underline{Seq}_{MS}(\mathsf{SynResp}), seq_{HE})] \Rightarrow seq_{he} = seq_{HE} \\
&\quad \wedge \; \mathrm{ADS}(\mathsf{AVs\_new}, \mathsf{SN}) \Rightarrow \\
&\qquad\qquad \wedge \; \mathsf{AVs\_new} \neq \epsilon \\
&\qquad\qquad \wedge \; \mathsf{AV} \in \mathsf{AVs\_new} \Rightarrow cons(\mathsf{AV}) \\
&\qquad\qquad \wedge \; \forall_{\mathsf{AV}_1,\mathsf{AV}_2 \in \mathsf{AVs\_new}} \; (\mathsf{AV}_1 \prec \mathsf{AV}_2 \Leftrightarrow \mathsf{AV}_1 \ll \mathsf{AV}_2) \\
&\qquad\qquad \wedge \; \forall_{\mathsf{AV} \in \mathsf{AVs\_new}} \; (seq_{he} < \underline{Seq}(\mathsf{AV}) \leq seq'_{HE}) \\
&\qquad\qquad \wedge \; \forall_{i \in \mathbb{N}} \; \exists_{\mathsf{AV} \in \mathsf{AVs\_new}} \; (seq_{he} < i \leq seq'_{HE} \Rightarrow \underline{Seq}(\mathsf{AV}) = i) \\
&\qquad\qquad \wedge \; seq'_{HE} - seq_{he} \leq \mathsf{N} \\
&\quad \wedge \; seq_{HE}\updownarrow \Rightarrow \exists_{\mathsf{AVs\_new},\mathsf{SN}} \; \mathrm{ADS}(\mathsf{AVs\_new}, \mathsf{SN}) \wedge \mathsf{AVs\_new} \neq \epsilon \;]
\end{aligned}
$$

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: $synchr(seq_{MS}, seq_{HE} + \mathsf{N})$. The definition of $synchr_1(x, y)$ is therefore:

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

This specification says nothing about where do the AVs in $\mathsf{AVs\_new}$ 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 $\mathrm{ADR}_1(\mathsf{SynResp}, \mathsf{Rand}, \mathsf{SN})$ with $verif(\mathsf{SynResp}, \mathsf{Rand}) \wedge \neg synchr_1(\underline{Seq}_{MS}(\mathsf{SynResp}), seq_{HE})$, the the parameter $\mathsf{AVs\_new}$ in $\mathrm{ADS}(\mathsf{AVs\_new}, \mathsf{SN})$ 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 $\mathsf{AVs\_new}$ is never ϵ. Later, for the incorrect system, we will weaken this assumption.

4.4 The Communication Channel

Recall from Convention 3.2 that for instance $\mathrm{UAR_S}(\mathsf{SN})$ means $\exists_{\mathsf{resp}}\ \mathrm{UAR_S}(\mathsf{resp},\mathsf{SN})$, 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.

$$\begin{aligned} Comm(\mathsf{SN}) :\Leftrightarrow & \vee \mathrm{UAR_S}(\mathsf{SN}) \vee \mathrm{UAR_S}(\mathsf{SN}) \\ & \vee \mathrm{UAS_S}(\mathsf{SN}) \vee \mathrm{UAS_S}(\mathsf{SN}) \\ & \vee \mathrm{UAJ_S}(\mathsf{SN}) \vee \mathrm{UAJ_S}(\mathsf{SN}) \\ & \vee \mathrm{UASF_S}(\mathsf{SN}) \vee \mathrm{UASF_S}(\mathsf{SN}) \end{aligned}$$

We will assume that the MS communicates at the same time with only one SN: $Comm(\mathsf{SN}) \wedge Comm(\mathsf{SN}_1) \Rightarrow \mathsf{SN} = \mathsf{SN}_1$

The message User Authent. Request may be received correctly (OKR), or it may be corrupted during the transmission ($CorrR$), or it may get lost ($LossR$):

$$\begin{aligned} OKR :\Leftrightarrow \exists_{\mathsf{chall},\mathsf{SN}} & \wedge \mathrm{UAR_S}(\mathsf{chall},\mathsf{SN}) \\ & \wedge \mathrm{UAR_R}(\mathsf{chall},\mathsf{SN}) \\ CorrR :\Leftrightarrow \exists_{\mathsf{chall},\mathsf{SN}} & \wedge \mathrm{UAR_S}(\mathsf{chall},\mathsf{SN}) \\ & \wedge \exists_{\mathsf{chall}_1}\ \neg cons(\mathsf{chall}_1) \wedge \mathrm{UAR_R}(\mathsf{chall}_1,\mathsf{SN}) \\ LossR :\Leftrightarrow \exists_{\mathsf{chall},\mathsf{SN}} & \wedge \mathrm{UAR_S}(\mathsf{chall},\mathsf{SN}) \\ & \wedge \neg \mathrm{UAR_R}(\mathsf{SN}) \end{aligned}$$

We will assume that during each step of the normal system, $\mathrm{UAR_S} \Rightarrow OKR \vee CorrR \vee LossR$. In other words, our assumption is that the challenge chall in $\mathrm{UAR_S}$(chall,SN) can not be replaced during the communication by another challenge chall_1 (in $\mathrm{UAR_R}(\mathsf{chall}_1,\mathsf{SN})$) 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 $\mathrm{UAR_S}(\underline{Chall}(\mathsf{AV}),\mathsf{SN})$ 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 $\mathcal{N}^{Ass_3}_{critical}$, discussed later in Assumption 6.3.

$$\begin{aligned} OKS :\Leftrightarrow \vee\ & \exists_{\mathsf{resp},\mathsf{SN}} \wedge \mathrm{UAS_S}(\mathsf{resp},\mathsf{SN}) \\ & \qquad \wedge \mathrm{UAS_R}(\mathsf{resp},\mathsf{SN}) \\ \vee\ & \exists_{\mathsf{RejResp},\mathsf{SN}} \wedge \mathrm{UAJ_S}(\mathsf{RejResp},\mathsf{SN}) \\ & \qquad \wedge \mathrm{UAJ_R}(\mathsf{RejResp},\mathsf{SN}) \\ \vee\ & \exists_{\mathsf{SynResp},\mathsf{SN}} \wedge \mathrm{UASF_S}(\mathsf{SynResp},\mathsf{SN}) \\ & \qquad \wedge \mathrm{UASF_R}(\mathsf{SynResp},\mathsf{SN}) \end{aligned}$$

$$
\begin{aligned}
CorrS :\Leftrightarrow \vee\ & \exists_{\mathsf{resp,SN}}\ \wedge \mathrm{UAS_S}(\mathsf{resp},\mathsf{SN}) \\
& \qquad \wedge \exists_{\mathsf{resp}_1}\ \mathsf{resp}_1 \neq \mathsf{resp} \wedge \mathrm{UAS_R}(\mathsf{resp}_1,\mathsf{SN}) \\
\vee\ & \exists_{\mathsf{RejResp,SN}} \\
& \qquad \wedge \mathrm{UAJ_S}(\mathsf{RejResp},\mathsf{SN}) \\
& \qquad \wedge \exists_{\mathsf{RejResp}_1}\ \mathsf{RejResp}_1 \neq \mathsf{RejResp} \wedge \mathrm{UAJ_R}(\mathsf{RejResp}_1,\mathsf{SN}) \\
\vee\ & \exists_{\mathsf{SynResp,SN}} \\
& \qquad \wedge \mathrm{UASF_S}(\mathsf{SynResp},\mathsf{SN}) \\
& \qquad \wedge \exists_{\mathsf{SynResp}_1}\ \wedge \mathsf{SynResp}_1 \neq \mathsf{SynResp} \\
& \qquad\qquad \wedge \mathrm{UASF_R}(\mathsf{SynResp}_1,\mathsf{SN})
\end{aligned}
$$

$$
\begin{aligned}
LossS :\Leftrightarrow \wedge\ & \vee\ \mathrm{UAS_S}(\mathsf{SN}) \\
& \vee\ \mathrm{UAJ_S}(\mathsf{SN}) \\
& \vee\ \mathrm{UASF_S}(\mathsf{SN}) \\
& \wedge \neg\mathrm{UAS_R}(\mathsf{SN}) \\
& \wedge \neg\mathrm{UAJ_R}(\mathsf{SN}) \\
& \wedge \neg\mathrm{UASF_R}(\mathsf{SN})
\end{aligned}
$$

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

Assumption 4.1.

$$
\begin{aligned}
& \mathcal{N}^{Ass}_{normal} :\Leftrightarrow \\
& \quad [\wedge \mathrm{UAR_S}(\mathsf{chall},\mathsf{SN}) \wedge\ \mathrm{UASF_R}(\mathsf{SynResp},\mathsf{SN}) \\
& \quad\ \wedge\ \neg\mathrm{UASF_S}(\mathsf{SynResp},\mathsf{SN})] \\
& \qquad \Rightarrow \neg \mathit{verif}(\mathsf{SynResp}, \underline{\mathit{Rand}}(\mathsf{AV}))
\end{aligned}
$$

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:

$$
\begin{aligned}
& \mathcal{N}^{CC}_{normal} :\Leftrightarrow \\
& \quad \wedge\ \forall_{\mathsf{SN,SN}_1}\ \mathit{Comm}(\mathsf{SN}) \wedge \mathit{Comm}(\mathsf{SN}_1) \Rightarrow \mathsf{SN} = \mathsf{SN}_1 \\
& \quad \wedge\ \mathrm{UAR_S} \Rightarrow \mathit{OKR} \vee \mathit{CorrR} \vee \mathit{LossR} \\
& \quad \wedge\ \mathrm{UAR_R} \Rightarrow \mathit{OKR} \vee \mathit{CorrR} \vee \mathit{LossR} \\
& \quad \wedge\ \mathrm{UAS_S} \vee \mathrm{UAJ_S} \vee \mathrm{UASF_S} \Rightarrow \mathit{OKS} \vee \mathit{CorrS} \vee \mathit{LossS} \\
& \quad \wedge\ \mathrm{UAS_R} \vee \mathrm{UAJ_R} \vee \mathrm{UASF_R} \Rightarrow \mathit{OKS} \vee \mathit{CorrS} \vee \mathit{LossS} \\
& \quad \wedge\ \mathcal{N}^{Ass}_{normal}
\end{aligned}
$$

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 $SQNmax/\Delta$:*

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

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,

it sends a USER AUTHENT. RESPONSE: The only reason for updating the variable *la_chall* is the one given above:

$$
\begin{aligned}
&\mathcal{N}^{MS}_{normal} :\Leftrightarrow \\
&\quad \wedge\ \mathrm{UAR_R}(\mathsf{chall},\mathsf{SN}) \Rightarrow \wedge\ \ cons(\mathsf{chall}) \wedge synchr(seq_{MS}, \underline{Seq}(\mathsf{chall})) \\
&\qquad\qquad\qquad\qquad \Rightarrow \mathrm{UAS_S}(\underline{Resp}(\mathsf{chall}),\mathsf{SN})) \\
&\qquad\qquad\qquad \wedge\ \neg cons(\mathsf{chall}) \\
&\qquad\qquad\qquad\qquad \Rightarrow \mathrm{UAJ_S}(\underline{RejResp}(\mathsf{chall}),\mathsf{SN}) \\
&\qquad\qquad\qquad \wedge\ \ cons(\mathsf{chall}) \wedge \neg synchr(seq_{MS}, \underline{Seq}(\mathsf{chall})) \\
&\qquad\qquad\qquad\qquad \Rightarrow \mathrm{UASF_S}(\underline{SynResp}(la\_chall,\mathsf{chall}),\mathsf{SN})) \\
&\quad \wedge\ \mathrm{UAS_S} \vee \mathrm{UAJ_S} \vee \mathrm{UASF_S} \Rightarrow \mathrm{UAR_R} \\
&\quad \wedge\ \mathrm{UAR_R}(\mathsf{chall},\mathsf{SN}) \wedge\ \mathrm{UAS_S} \Rightarrow la\_chall' = \mathsf{chall} \\
&\quad \wedge\ \mathrm{UAR_R}(\mathsf{chall},\mathsf{SN}) \wedge\ \neg\mathrm{UAS_S} \Rightarrow la\_chall' = la\_chall \\
&\quad \wedge\ \ la\_chall\updownarrow \Rightarrow \mathrm{UAS_S}(\mathsf{resp},\mathsf{SN}) \\
&\quad \wedge\ \ Lifetime'
\end{aligned}
$$

5 Definition of Normal Behavior

Definition 5.1. *An* AV *is called* generated *(by the home environment) if the home environment has sent this* AV *in an Authentication Data Response. Formally, the variable Gen, of type* $\wp(\mathcal{AV})$ *is defined by the temporal formula:*

$$
\begin{aligned}
&\mathcal{A}^{Gen}_{normal} :\Leftrightarrow \\
&\quad \wedge\ \ Gen = \emptyset \\
&\quad \wedge\ \ \Box(\ \wedge Gen\updownarrow \Rightarrow \exists_{\mathsf{AVs\_new},\mathsf{SN}}\ \ \mathrm{ADS}(\mathsf{AVs\_new},\mathsf{SN}) \wedge \mathsf{AVs\_new} \neq \epsilon \\
&\qquad\quad \wedge \exists_{\mathsf{AVs\_new},\mathsf{SN}}\ \ \mathrm{ADS}(\mathsf{AVs\_new},\mathsf{SN}) \Rightarrow Gen' = Gen \cup \mathsf{AVs\_new}\ )
\end{aligned}
$$

Definition 5.2. *An* AV *copy is* lost *if it is either:*

1. *lost or corrupted in the communication Channel from the SN to the USIM, or*
2. *lost or intentionally discarded during a Location Update*

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

$$
\begin{aligned}
&\mathcal{A}^{Lost}_{normal} :\Leftrightarrow \\
&\quad \wedge\ \ Lost = \emptyset \\
&\quad \wedge\ \ \Box(\ \wedge\ Lost\updownarrow \Rightarrow \vee\ \mathrm{UAR_s} \wedge (CorrR \vee LossR) \\
&\qquad\qquad\qquad\qquad \vee\ \mathrm{LUR}(\mathsf{SN},\mathsf{SN_1}) \wedge\ \neg\mathrm{MvAV}(\mathsf{SN},\mathsf{SN_1}) \\
&\qquad\quad \wedge\ \mathrm{UAR_s}(\underline{Chall}(\mathsf{AV}),\mathsf{SN}) \wedge (CorrR \vee LossR) \\
&\qquad\qquad\qquad\qquad \Rightarrow Lost' = Lost\ \cup\ \{\mathsf{AV}\} \\
&\qquad\quad \wedge\ \mathrm{LUR}(\mathsf{SN},\mathsf{SN_1}) \wedge\ \neg\mathrm{MvAV}(\mathsf{SN},\mathsf{SN_1}) \Rightarrow \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad Lost' = Lost\ \cup\ DB(\mathsf{SN})\ )
\end{aligned}
$$

It is not necessary to explicitly model losses of AVs (1.) inside of an SN or during the (2.) communication between the home environment and the serving network or

(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:*

$$\begin{aligned}
&\mathcal{A}^{Used}_{normal} :\Leftrightarrow \\
&\quad \wedge\ Used = \emptyset \\
&\quad \wedge\ \ \Box(\ \wedge Used\!\downarrow \Rightarrow \mathrm{UAR_S} \\
&\qquad\qquad \wedge \mathrm{UAR_S}(\underline{Chall}(\mathsf{AV}), \mathsf{SN}) \Rightarrow Used' = Used\ \cup\ \{\mathsf{AV}\}
\end{aligned}$$

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:*

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

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* $synchr(seq_{MS}, seq_{HE}+1)$. *In this case we call the system* weakly-synchronous*:*

$$WeakSynchr :\Leftrightarrow synchr(seq_{MS}, seq_{HE}+1)$$

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

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}*).*

$$StrongSynchr :\Leftrightarrow WeakSynchr \wedge \forall_{\mathsf{AV} \in DB(last\_SN)}\ \ synchr(seq_{MS}, \underline{Seq}(\mathsf{AV}))$$

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.*

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

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

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

6 Transitions of the incorrect System

6.1 Events: more Transitions

Message or Event	*Trans. Pred.*	*Param. Name*	*Param. Type*	*Var. Name*
ERROR HE	XHE	Failure	$\mathcal{FAIL}$	$n_{\mathtt{XHE}}$
ERROR SN	XSN	SN Failure	$\mathcal{SN}$ $\mathcal{FAIL}$	$n_{\mathtt{XSN}}$
ERROR LU	XLU	SN Failure	$\mathcal{SN}$ $\mathcal{FAIL}$	$n_{\mathtt{XLU}}$

Table 3: Failure Events

Definition 6.1. (The Failure Events)

$$\mathrm{XHE}(\mathsf{Failure}) :\Leftrightarrow n_{\mathtt{XHE}}(\mathsf{Failure})\updownarrow$$
$$\mathrm{XSN}(\mathsf{SN}, \mathsf{Failure}) :\Leftrightarrow n_{\mathtt{XSN}}(\mathsf{SN}, \mathsf{Failure})\updownarrow$$
$$\mathrm{XLU}(\mathsf{SN}, \mathsf{Failure}) :\Leftrightarrow n_{\mathtt{XLU}}(\mathsf{SN}, \mathsf{Failure})\updownarrow$$

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:

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

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

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

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:

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

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.

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

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

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

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

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

The other difference to $\mathcal{N}^{LU}_{normal}$ is that if the race condition happens, then while ADS is performed, $(\mathrm{ADS} \wedge \neg\exists_{\mathsf{SN}}\ \mathrm{XLU}(\mathsf{SN}, \mathsf{Race}))$, 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}^{SN}_{critical} :\Leftrightarrow \\
&\quad \wedge\ \mathrm{ADR}_0(\mathsf{SN}) \Rightarrow DB(\mathsf{SN}) = \epsilon \wedge \mathsf{SN} \in curr\_SN \\
&\quad \wedge\ \mathrm{ADR}_1(\mathsf{SynResp}, \mathsf{Rand}, \mathsf{SN}) \Rightarrow \\
&\qquad\qquad \wedge\ \mathrm{UASF}_{\mathrm{R}}(\mathsf{SynResp}, \mathsf{SN}) \\
&\qquad\qquad \wedge\ \mathsf{SN} \in curr\_SN \\
&\qquad\qquad \wedge\ \exists_{\mathsf{AV}}\ (\mathrm{UAR}_{\mathrm{S}}(\underline{Chall}(\mathsf{AV}), \mathsf{SN}) \wedge\ \mathsf{Rand} = \underline{Rand}(\mathsf{AV})) \\
&\quad \wedge\ \mathrm{ADS}(\mathsf{AVs\_new}, \mathsf{SN}) \Rightarrow \wedge\ \mathsf{AVs\_new} \neq \epsilon \Rightarrow DB'(\mathsf{SN}) = \mathsf{AVs\_new} \\
&\qquad\qquad\qquad\qquad\qquad \wedge\ \mathsf{AVs\_new} = \epsilon \Rightarrow DB'(\mathsf{SN}) = DB(\mathsf{SN}) \\
&\quad \wedge\ \mathrm{UAR}_{\mathrm{S}}(\mathsf{chall}, \mathsf{SN}) \wedge\ \neg\mathrm{XSN}(\mathsf{SN}, \mathsf{Loss}) \Rightarrow \\
&\qquad\qquad \wedge\ \mathsf{SN} \in curr\_SN \wedge DB(\mathsf{SN}) \neq \epsilon \\
&\qquad\qquad \wedge\ DB'(\mathsf{SN}) = Tail(DB(\mathsf{SN})) \\
&\qquad\qquad \wedge\ \mathsf{chall} = \underline{Chall}(Head(DB(\mathsf{SN}))) \\
&\quad \wedge\ \mathrm{XSN}(\mathsf{SN}, \mathsf{Crash}) \Rightarrow \mathrm{UAR}_{\mathrm{S}} \\
&\quad \wedge\ \mathrm{UAR}_{\mathrm{S}}(\mathsf{chall}, \mathsf{SN}) \wedge\ \mathrm{XSN}(\mathsf{SN}, \mathsf{Crash}) \Rightarrow \\
&\qquad\qquad \exists_{\mathsf{AV} \in Used}\ \mathsf{chall} = \underline{Chall}(\mathsf{AV}) \\
&\quad \wedge\ \mathrm{XSN}(\mathsf{SN}, \mathsf{DB}) \Rightarrow Set(DB'(\mathsf{SN})) = Set(DB(\mathsf{SN})) \\
&\quad \wedge\ \mathrm{XSN}(\mathsf{SN}, \mathsf{Loss}) \Rightarrow \wedge\ Set(DB'(\mathsf{SN})) \subseteq Set(DB(\mathsf{SN})) \\
&\qquad\qquad\qquad\qquad \wedge\ \mathsf{AV}_1 \ll_{DB'(\mathsf{SN})} \mathsf{AV}_2 \Rightarrow \mathsf{AV}_1 \ll_{DB(\mathsf{SN})} \mathsf{AV}_2 \\
&\quad \wedge\ DB(\mathsf{SN})\!\Updownarrow \Rightarrow \vee\ \exists_{\mathsf{SN}_1}\ \mathrm{LUR}(\mathsf{SN}_1, \mathsf{SN}) \wedge \neg\mathrm{XLU}(\mathsf{SN}_1, \mathsf{DB}) \\
&\qquad\qquad\qquad \vee\ \mathrm{XSN}(\mathsf{SN}, \mathsf{DB}) \vee \mathrm{XSN}(\mathsf{SN}, \mathsf{Loss}) \\
&\qquad\qquad\qquad \vee\ \exists_{\mathsf{AVs\_new}, \mathsf{SN}}\ \ \mathrm{ADS}(\mathsf{AVs\_new}, \mathsf{SN}) \wedge \mathsf{AVs\_new} \neq \epsilon \\
&\qquad\qquad\qquad \vee\ \exists_{\mathsf{chall}, \mathsf{SN}}\ \ \mathrm{UAR}_{\mathrm{S}}(\mathsf{chall}, \mathsf{SN}) \wedge \neg\mathrm{XSN}(\mathsf{SN}, \mathsf{Loss}) \\
&\qquad\qquad\qquad \vee\ \exists_{\mathsf{SN}_1}\ \ \mathrm{MvAV}(\mathsf{SN}, \mathsf{SN}_1) \\
&\qquad\qquad\qquad \vee\ \exists_{\mathsf{SN}_1}\ \ \mathrm{MvAV}(\mathsf{SN}_1, \mathsf{SN}) \\
&\quad \wedge\ [\ \wedge\ \mathrm{UASF}_{\mathrm{R}}(\mathsf{SynResp}, \mathsf{SN}) \\
&\qquad \wedge\ \mathsf{SN} \in curr\_SN \\
&\qquad \wedge\ \mathrm{UAR}_{\mathrm{S}}(\underline{Chall}(\mathsf{AV}), \mathsf{SN}) \\
&\qquad \wedge\ \mathsf{Rand} = \underline{Rand}(\mathsf{AV})\ ] \Rightarrow \mathrm{ADR}_1(\mathsf{SynResp}, \mathsf{Rand}, \mathsf{SN})
\end{aligned}
$$

6.4 The Home Environment

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

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

then a CanLoc(SN) can be produced, instead of the ADS expected by the serving network. But we insist that $(\mathrm{ADR}_0 \vee\ \mathrm{ADR}_1) \Rightarrow \mathrm{ADS}$. We model the described situation as follows: first send a ADS(AVs_new,SN) with $\mathsf{AVs\_new} = \epsilon$ and then send a CanLoc. This sequence is equivalent to just sending one CanLoc. Notice that our specification does not constrain at all the occurrences of 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}^{HE}_{critical} :\Leftrightarrow \exists_{seq_{he}} \; [\\
&\wedge \; \mathrm{ADS}(\mathsf{AVs\_new}, \mathsf{SN}) \Rightarrow \vee \; \mathrm{ADR}_0(\mathsf{SN}) \\
&\qquad\qquad\qquad\qquad \vee \; \exists_{\mathsf{SynResp},\mathsf{Rand}} \; \mathrm{ADR}_1(\mathsf{SynResp}, \mathsf{Rand}, \mathsf{SN}) \\
&\wedge \; (\mathrm{ADR}_0 \vee \; \mathrm{ADR}_1) \Rightarrow \mathrm{ADS} \\
&\wedge \; \mathrm{ADR}_0 \Rightarrow seq_{he} = seq_{HE} \\
&\wedge \; \mathrm{ADR}_1(\mathsf{SynResp}, \mathsf{Rand}, \mathsf{SN}) \Rightarrow \\
&\quad \wedge \; [\mathit{verif}(\mathsf{SynResp}, \mathsf{Rand}) \wedge \\
&\qquad \neg \mathit{synchr}_1(\underline{\mathit{Seq}}_{MS}(\mathsf{SynResp}), seq_{HE})] \Rightarrow seq_{he} = \underline{\mathit{Seq}}_{MS}(\mathsf{SynResp}) \\
&\quad \wedge \; [\neg \mathit{verif}(\mathsf{SynResp}, \mathsf{Rand}) \vee \\
&\qquad \mathit{synchr}_1(\underline{\mathit{Seq}}_{MS}(\mathsf{SynResp}), seq_{HE})] \Rightarrow seq_{he} = seq_{HE} \\
&\wedge \; \mathrm{ADS}(\mathsf{AVs\_new}, \mathsf{SN}) \Rightarrow \\
&\qquad\qquad \wedge \neg \mathrm{XLU}(\mathsf{SN}, \mathsf{Race}) \Rightarrow \mathsf{AVs\_new} \neq \epsilon \\
&\qquad\qquad \wedge \mathsf{AV} \in \mathsf{AVs\_new} \Rightarrow \mathit{cons}(\mathsf{AV}) \\
&\qquad\qquad \wedge \forall_{\mathsf{AV}_1, \mathsf{AV}_2 \in \mathsf{AVs\_new}} \; (\mathsf{AV}_1 \prec \mathsf{AV}_2 \Leftrightarrow \mathsf{AV}_1 \ll \mathsf{AV}_2) \\
&\qquad\qquad \wedge \forall_{\mathsf{AV} \in \mathsf{AVs\_new}} \; (seq_{he} < \underline{\mathit{Seq}}(\mathsf{AV}) \leq seq'_{HE}) \\
&\qquad\qquad \wedge \forall_{i \in \mathbb{N}} \; \exists_{\mathsf{AV} \in \mathsf{AVs\_new}} \; (seq_{he} < i \leq seq'_{HE} \Rightarrow \underline{\mathit{Seq}}(\mathsf{AV}) = i) \\
&\qquad\qquad \wedge seq'_{HE} - seq_{he} \leq \mathsf{N} \\
&\wedge \; \mathrm{XHE}(\mathsf{DB}) \Rightarrow seq_{HE}{}' < seq_{HE} \\
&\wedge \; seq_{HE}{}^{\updownarrow} \Rightarrow \vee \; \exists_{\mathsf{AVs\_new},\mathsf{SN}} \; \mathrm{ADS}(\mathsf{AVs\_new}, \mathsf{SN}) \wedge \mathsf{AVs\_new} \neq \epsilon \\
&\qquad\qquad \vee \mathrm{XHE}(\mathsf{DB}) \\
&\qquad\qquad \vee \mathrm{XHE}(\mathsf{Steal}) \\
&\qquad\qquad \vee \mathrm{XHE}(\mathsf{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 ($ATTR$).

$$
\begin{aligned}
\mathit{ATTR} :\Leftrightarrow \exists_{\mathsf{chall},\mathsf{SN}} \; &\wedge \mathrm{UAR}_{\mathrm{S}}(\mathsf{chall}, \mathsf{SN}) \\
&\wedge \exists_{\mathsf{chall}_1 \neq \mathsf{chall}} \; \mathit{cons}(\mathsf{chall}_1) \wedge \mathrm{UAR}_{\mathrm{R}}(\mathsf{chall}_1, \mathsf{SN})
\end{aligned}
$$

Notice that the following is a tautology:

$$
\begin{aligned}
\mathrm{UAR}_{\mathrm{S}}(\mathsf{chall}, \mathsf{SN}) \Rightarrow &\vee \mathrm{UAR}_{\mathrm{R}}(\mathsf{chall}, \mathsf{SN}) \\
&\vee \exists_{\mathsf{chall}_1} \; \neg \mathit{cons}(\mathsf{chall}_1) \wedge \mathrm{UAR}_{\mathrm{R}}(\mathsf{chall}_1, \mathsf{SN}) \\
&\vee \neg \mathrm{UAR}_{\mathrm{R}}(\mathsf{SN}) \\
&\vee \exists_{\mathsf{chall}_1 \neq \mathsf{chall}} \; \mathit{cons}(\mathsf{chall}_1) \wedge \mathrm{UAR}_{\mathrm{R}}(\mathsf{chall}_1, \mathsf{SN})
\end{aligned}
$$

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

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

$$\begin{aligned} ATTRi :\Leftrightarrow \exists_{\mathsf{chall,SN}} \ & \wedge \mathrm{UAR_R}(\mathsf{chall}, \mathsf{SN}) \wedge cons(\mathsf{chall}) \\ & \wedge \neg \mathrm{UAR_S} \end{aligned}$$

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 $\mathcal{N}^{CC}_{normal}$, if a message was received, then a corresponding message was as also sent (perhaps with different parameter values, they can be corrupted). For instance, if $\mathrm{UAS_R}$ happens, then either OKR or $CorrR$ or $LossR$ happens, and in any case, $\mathrm{UAS_S}$ 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):

$$\begin{aligned} ATTSi(\mathsf{SN}) :\Leftrightarrow \wedge \ & \exists_{\mathsf{chall}} \ \mathrm{UAR_S}(\mathsf{chall}, \mathsf{SN}) \\ & \wedge \neg \exists_{\mathsf{resp}} \ \mathrm{UAS_S}(\mathsf{resp}, \mathsf{SN}) \\ & \wedge \neg \exists_{\mathsf{RejResp}} \ \mathrm{UAJ_S}(\mathsf{RejResp}, \mathsf{SN}) \\ & \wedge \neg \exists_{\mathsf{SynResp}} \ \mathrm{UASF_S}(\mathsf{SynResp}, \mathsf{SN}) \\ & \wedge \ \vee \ \exists_{\mathsf{resp}} \ \mathrm{UAS_R}(\mathsf{resp}, \mathsf{SN}) \\ & \quad \vee \ \exists_{\mathsf{RejResp}} \ \mathrm{UAJ_R}(\mathsf{RejResp}, \mathsf{SN}) \\ & \quad \vee \ \exists_{\mathsf{SynResp}} \ \mathrm{UASF_R}(\mathsf{SynResp}, \mathsf{SN}) \end{aligned}$$

$$ATTSi :\Leftrightarrow \exists_{\mathsf{SN}} \ ATTSi(\mathsf{SN})$$

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.

$$\begin{aligned} & \mathcal{N}^{Ass_1}_{critical} :\Leftrightarrow \\ & \quad [\ \mathrm{UAR_R}(\mathsf{chall}, \mathsf{SN}) \wedge \neg \mathrm{UAR_S}(\mathsf{chall}, \mathsf{SN}) \wedge cons(\mathsf{chall})] \\ & \qquad \Rightarrow \exists_{\mathsf{AV} \in Stolen \ \cup \ Used} \ \mathsf{chall} = \underline{Chall}(\mathsf{AV}) \end{aligned}$$

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

Assumption 6.2.

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

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

Assumption 6.3.

$$\begin{aligned}&\mathcal{N}^{Ass_3}_{critical} :\Leftrightarrow \\ &\quad [\ \wedge\ \mathrm{UAR_S}(\mathsf{chall},\mathsf{SN}) \wedge\ \mathrm{UASF_R}(\mathsf{SynResp},\mathsf{SN}) \\ &\quad\ \wedge\ \neg\mathrm{UASF_S}(\mathsf{SynResp},\mathsf{SN}) \wedge\ \mathit{verif}(\mathsf{SynResp},\underline{\mathit{Rand}}(\mathsf{AV}))] \\ &\qquad \Rightarrow (\mathsf{AV} \in \mathit{Stolen}\ \cup\ \mathit{Used})\ \wedge\ \mathsf{SynResp} = \underline{\mathit{SynResp}}(\mathsf{AV})\end{aligned}$$

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

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

6.6 The Mobile Station

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

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

7 Definition of Incorrect Behavior

Definition 7.1. *The stealing of AVs generates a* clone *(not a "copy") of an AV. Stolen*[3]*, the set of clones, is defined by the formulas:*

$$\begin{aligned}&\mathit{Stolen} := \mathit{Stolen}_{HE} \cup \mathit{Stolen}_{SN} \\ &\mathit{Stolen}_{HE} = \emptyset \wedge \Box(\wedge\ \mathit{Stolen}_{HE}^{\updownarrow} \Rightarrow \mathrm{XHE}(\mathsf{Steal}) \\ &\qquad\qquad \wedge\ \mathrm{XHE}(\mathsf{Steal}) \Rightarrow \mathit{Stolen}_{HE}{}' \supseteq \mathit{Stolen}_{HE}) \\ &\mathit{Stolen}_{SN} = \emptyset \wedge \Box(\wedge\ \mathit{Stolen}_{SN}^{\updownarrow} \Rightarrow \mathrm{XSN}(\mathsf{SN},\mathsf{Steal}) \\ &\qquad\qquad \wedge\ \mathrm{XSN}(\mathsf{SN},\mathsf{Steal}) \Rightarrow \\ &\qquad\qquad\qquad \mathit{Stolen}_{SN} \subseteq \mathit{Stolen}_{SN}{}' \subseteq \mathit{Stolen}_{SN} \cup \mathit{DB}(\mathsf{SN}))\end{aligned}$$

Definition 7.2. *As before (Def. 5.5), if curr_SN is a sigleton or empty, an unused* AV *copy is* usable *if its location is last_SN, 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). If curr_SN contains more than one elemant, then an unused* AV *is* usable *if its location is contained in curr_SN, that is, it is an element of* $\cup_{\mathsf{SN}\in \mathit{curr\_SN}} \mathit{DB}(\mathsf{SN})$.

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 loose 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 loosing 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)*

[3]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:*

$$
\begin{aligned}
&\mathcal{A}^{Lost}_{critical} :\Leftrightarrow \\
&\quad \wedge\ Lost = \emptyset \\
&\quad \wedge\ \Box(\ \wedge\ Lost\updownarrow \Rightarrow \vee\ \mathrm{UAR_s} \wedge \neg OKR \\
&\qquad\qquad\qquad\qquad \vee\ \mathrm{LUR}(\mathsf{SN}, \mathsf{SN}_1) \wedge\ \neg \mathrm{MvAV}(\mathsf{SN}, \mathsf{SN}_1) \\
&\qquad\qquad\qquad\qquad \vee\ \exists_{\mathsf{SN}}\ \mathrm{XSN}(\mathsf{SN}, \mathsf{Loss}) \\
&\qquad\qquad\qquad\qquad \vee\ \exists_{\mathsf{SN}}\ \mathrm{XSN}(\mathsf{SN}, \mathsf{DB}) \\
&\qquad \wedge\ \mathrm{UAR_s}(\underline{Chall}(\mathsf{AV}), \mathsf{SN}) \wedge \neg OKR \\
&\qquad\qquad\qquad\qquad \Rightarrow Lost' = Lost\ \cup\ \{\mathsf{AV}\} \\
&\qquad \wedge\ \mathrm{LUR}(\mathsf{SN}, \mathsf{SN}_1) \wedge\ \neg \mathrm{MvAV}(\mathsf{SN}, \mathsf{SN}_1) \Rightarrow \\
&\qquad\qquad\qquad\qquad\qquad\qquad Lost' = Lost\ \cup\ DB(\mathsf{SN}) \\
&\qquad \wedge\ \mathrm{XSN}(\mathsf{SN}, \mathsf{Loss}) \Rightarrow Lost' = Lost\ \cup\ (DB(\mathsf{SN}) \setminus DB'(\mathsf{SN})) \\
&\qquad \wedge\ \mathrm{XSN}(\mathsf{SN}, \mathsf{DB}) \Rightarrow Lost' = Lost\ \cup\ DB(\mathsf{SN})\ )
\end{aligned}
$$

This definition of *Lost* is only one of several possible choices. We could say that if XSN(SN,DB) happens, not all AVs in $DB(\mathsf{SN})$ are lost, only those AV for which there is an AV_1 in $DB'(\mathsf{SN})$, such that AV_1 is left of AV (it will be used earlier than AV) but AV_1 has a larger sequence number than AV:

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

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:*

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

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

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

Definition 7.6. *An* AV *clone is called* synchronous *(with respect to* seq_{MS}*) if* $synchr(seq_{MS}, \underline{Seq}(\mathsf{AV}))$

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:*

$$Perfect :\Leftrightarrow StrongSynchr \wedge Stolen \subseteq Obsolete$$

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

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:*

$$\mathcal{N}^{Step}_{critical} :\Leftrightarrow \mathcal{N}^{LU}_{critical} \wedge \mathcal{N}^{SN}_{critical} \wedge \mathcal{N}^{HE}_{critical} \wedge \mathcal{N}^{CC}_{critical} \wedge \mathcal{N}^{MS}_{critical}$$

$$\mathcal{A}_{critical} :\Leftrightarrow$$

$$\mathsf{Init} \wedge \mathcal{A}^{interleave}_{critical} \wedge \mathcal{A}^{CurrSN}_{critical} \wedge \mathcal{A}^{LastSN}_{critical} \wedge \mathcal{A}^{Gen}_{critical} \wedge \mathcal{A}^{Lost}_{critical} \wedge \mathcal{A}^{Used}_{critical}$$
$$\wedge \Box(\mathcal{N}^{Step}_{critical})$$

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

- *For any* Δ *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.*

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

After the system has been behvaving 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* $\underline{1}$*:* $\Box(x\updownarrow \Rightarrow x' = \underline{1})$*. Then we say that the system* behaves normally when x *if during the time that* $x{=}\underline{1}$ *for any* $(\Delta - \mathsf{N} - 1)$ *AVs in sequence, at least 1* AV *is not lost, and on each transition step, the formulas* $\mathcal{N}^{LU}_{normal}$*,* $\mathcal{N}^{SN}_{normal}$*,* $\mathcal{N}^{HE}_{normal}$*,* $\mathcal{N}^{CC}_{normal}$*, and* $\mathcal{N}^{MS}_{normal}$ *hold:*

$$\mathcal{A}^{x}_{normal} :\Leftrightarrow$$

$$\mathsf{Init} \wedge \mathcal{A}^{interleave}_{critical} \wedge \mathcal{A}^{CurrSN}_{critical} \wedge \mathcal{A}^{LastSN}_{critical} \wedge \mathcal{A}^{Gen}_{critical} \wedge \mathcal{A}^{Lost}_{critical} \wedge \mathcal{A}^{Used}_{critical}$$
$$\wedge \forall_{Lost_0:\wp(\mathcal{AV})} \ [\ (\neg x \wedge x' \Rightarrow \mathit{Lost}' = \mathit{Lost}_0)$$
$$\Rightarrow \Box\{ \ x \Rightarrow \mathcal{N}^{Step}_{normal} \wedge \mathit{Interr}_{\Delta-\mathsf{N}-1}(\mathit{Lost}' \setminus \mathit{Lost}_0) \ \} \ ]$$

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:

$$\begin{aligned}\mathrm{UASF}_{successful} :\Leftrightarrow \exists_{\mathsf{chall,SN,SynResp}} \\ &\wedge\ \mathrm{UAR_S}(\mathsf{chall},\mathsf{SN}) \wedge \mathrm{UAR_R}(\mathsf{chall},\mathsf{SN}) \\ &\wedge\ \mathrm{UASF_S}(\mathsf{SynResp},\mathsf{SN}) \wedge \mathrm{UASF_R}(\mathsf{SynResp},\mathsf{SN}) \\ &\wedge\ \neg\exists_{\mathsf{SN}}\ \mathrm{XLU}(\mathsf{SN},\mathsf{Race})\end{aligned}$$

8 Theorems

Lemma 8.1.

$$\begin{aligned}&\mathcal{A}^{interleave}_{normal} \wedge \mathcal{N}^{MS}_{normal} \Rightarrow \\ &\quad\wedge\ \mathrm{UAS_S}(\mathsf{resp},\mathsf{SN}) \Rightarrow \exists_{\mathsf{chall}}\ \wedge\ \mathrm{UAR_R}(\mathsf{chall},\mathsf{SN}) \\ &\qquad\qquad\qquad\qquad\qquad\wedge\ \mathit{cons}(\mathsf{chall}) \\ &\qquad\qquad\qquad\qquad\qquad\wedge\ \mathit{synchr}(\mathit{seq}_{MS}, \underline{\mathit{Seq}}(\mathsf{chall})) \\ &\qquad\qquad\qquad\qquad\qquad\wedge\ \mathsf{resp} = \underline{\mathit{Resp}}(\mathsf{chall}) \\ &\quad\wedge\ \mathrm{UAJ_S}(\mathsf{RejResp},\mathsf{SN}) \Rightarrow \exists_{\mathsf{chall}}\ \mathrm{UAR_R}(\mathsf{chall},\mathsf{SN}) \wedge \neg\mathit{cons}(\mathsf{chall}) \\ &\quad\wedge\ \mathrm{UASF_S}(\mathsf{SynResp},\mathsf{SN}) \Rightarrow \exists_{\mathsf{chall}}\ \wedge\ \mathrm{UAR_R}(\mathsf{chall},\mathsf{SN}) \\ &\qquad\qquad\qquad\qquad\qquad\qquad\wedge\ \mathit{cons}(\mathsf{chall}) \\ &\qquad\qquad\qquad\qquad\qquad\qquad\wedge\ \neg\mathit{synchr}(\mathit{seq}_{MS}, \underline{\mathit{Seq}}(\mathsf{chall})) \\ &\qquad\qquad\qquad\qquad\qquad\qquad\wedge\ \mathsf{SynResp} = \underline{\mathit{SynResp}}(\mathit{la\_chall}, \mathsf{chall})\end{aligned}$$

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

$$\begin{aligned}&\mathcal{A}^{interleave}_{normal} \wedge \mathcal{N}^{MS}_{normal} \wedge\ \mathrm{UAJ_S}(\mathsf{RejResp},\mathsf{SN}) \Rightarrow \\ &\qquad\qquad\qquad\qquad\qquad\exists_{\mathsf{chall}}\ \mathrm{UAR_R}(\mathsf{chall},\mathsf{SN}) \wedge \neg\mathit{cons}(\mathsf{chall})\end{aligned}$$

$$\begin{aligned}&\mathcal{A}^{interleave}_{normal} \wedge \mathcal{N}^{MS}_{normal} \wedge\ \mathrm{UAJ_S}(\mathsf{RejResp},\mathsf{SN}) \\ &\qquad\Rightarrow \mathrm{UAJ_S} && \text{(Def of } \mathrm{UAJ_S}\text{)} \\ &\qquad\Rightarrow \mathrm{UAS_S} \vee \mathrm{UAJ_S} \vee \mathrm{UASF_S} && \text{(Conjunction Rules)} \\ &\qquad\Rightarrow \mathrm{UAR_R} && \text{(Def of } \mathcal{N}^{MS}_{normal}\text{)} \\ &\qquad\Rightarrow \exists_{\mathsf{chall,SN_1}}\ \mathrm{UAR_R}(\mathsf{chall},\mathsf{SN_1}) && \text{(Def of } \mathrm{UAR_S}\text{)} \\ &\qquad\Rightarrow \mathrm{UAR_R}(\mathsf{chall},\mathsf{SN_1}) && \text{(Skolemisation: Introd.} \\ &\qquad\qquad && \quad\text{of fresh variables)}\end{aligned}$$

$$
\begin{array}{ll}
[\ \text{Assume} \quad cons(\mathsf{chall}) \wedge synchr(seq_{MS}, \underline{Seq}(\mathsf{chall})) & \\
\quad \Rightarrow \mathrm{UAS_S}(\underline{Resp}(\mathsf{chall}), \mathsf{SN}_1)) & (\text{Def of } \mathcal{N}^{MS}_{normal}) \\
\quad \Rightarrow \mathrm{UAS_S} & (\text{Def of } \mathrm{UAS_S}) \\
\quad \Rightarrow \neg \mathrm{UAJ_S} & (\text{Def of } \mathcal{A}^{interleave}_{normal}) \\
\quad \Rightarrow \text{Contradiction} & (\mathrm{UAJ_S})\ ]
\end{array}
$$

$$
\begin{array}{ll}
\Rightarrow \neg(cons(\mathsf{chall}) \wedge synchr(seq_{MS}, \underline{Seq}(\mathsf{chall}))) & \text{(Assumption false)} \\
\Rightarrow \neg cons(\mathsf{chall}) \vee \neg synchr(seq_{MS}, \underline{Seq}(\mathsf{chall}))) & \text{(De Morgan)}
\end{array}
$$

$$
\begin{array}{ll}
[\ \text{Assume} \quad cons(\mathsf{chall}) \wedge \neg synchr(seq_{MS}, \underline{Seq}(\mathsf{chall})) & \\
\quad \Rightarrow \mathrm{UASF_S}(\underline{SynResp}(la\_chall, \mathsf{chall}), \mathsf{SN}) & (\text{Def of } \mathcal{N}^{MS}_{normal}) \\
\quad \Rightarrow \mathrm{UASF_S} & (\text{Def of } \mathrm{UAS_S}) \\
\quad \Rightarrow \neg \mathrm{UAJ_S} & (\text{Def of } \mathcal{A}^{interleave}_{normal}) \\
\quad \Rightarrow \text{Contradiction} & (\mathrm{UAJ_S})\ ]
\end{array}
$$

$$
\begin{array}{ll}
\Rightarrow \neg(cons(\mathsf{chall}) \wedge \neg synchr(seq_{MS}, \underline{Seq}(\mathsf{chall}))) & \text{(Assumption false)} \\
\Rightarrow \neg cons(\mathsf{chall}) \vee synchr(seq_{MS}, \underline{Seq}(\mathsf{chall}))) & \text{(De Morgan)} \\
\Rightarrow \neg cons(\mathsf{chall}) & \text{(Resolution)} \\
\Rightarrow \mathrm{UAJ_S}(\underline{RejResp}(\mathsf{chall}), \mathsf{SN}_1) & (\text{Def of } \mathcal{N}^{MS}_{normal} \\
 & \quad \text{and} \mathrm{UAR_R}(\mathsf{chall}, \mathsf{SN}_1)) \\
\Rightarrow \mathsf{RejResp} = \underline{RejResp}(\mathsf{chall}) \wedge \mathsf{SN} = \mathsf{SN}_1 & (\mathrm{UAJ_S}(\underline{RejResp}(\mathsf{chall}), \mathsf{SN}_1) \\
 & \quad \mathrm{UAJ_S}(\mathsf{RejResp}, \mathsf{SN})) \\
 & \quad \text{and Def of } \mathcal{A}^{interleave}_{normal}) \\
\Rightarrow \mathrm{UAR_R}(\mathsf{chall}, \mathsf{SN}) & (\mathsf{SN} = \mathsf{SN}_1) \\
\Rightarrow \mathrm{UAR_R}(\mathsf{chall}, \mathsf{SN}) \wedge \neg cons(\mathsf{chall}) & \text{Conjunction} \\
\Rightarrow \exists_{\mathsf{chall}}\ \mathrm{UAR_R}(\mathsf{chall}, \mathsf{SN}) \wedge \neg cons(\mathsf{chall}) & \text{Introd of Ex.}
\end{array}
$$

Lemma 8.2.

$$
\begin{array}{l}
\mathcal{A}^{CurrSN}_{normal} \wedge \mathcal{A}^{LastSN}_{normal} \Rightarrow \\
\qquad \Box(\ (\mathcal{N}^{LU}_{normal} \wedge curr\_SN' \neq \emptyset) \Rightarrow curr\_SN' = \{last\_SN\}\ )
\end{array}
$$

Lemma 8.3.

$$
\begin{array}{l}
\mathcal{A}_{normal} \Rightarrow \Box(\ \wedge\ DB(last\_SN) \neq \epsilon \Rightarrow seq_{HE} = \underline{Seq}(\max(DB(last\_SN))) \\
\qquad\qquad \wedge\ seq_{MS} = \underline{Seq}(\max(Accepted)) \\
\qquad\qquad \wedge\ \mathrm{UAR_S}(\underline{Chall}(\mathsf{AV}), \mathsf{SN}) \Rightarrow \mathsf{AV} = \min(DB(last\_SN))\ )
\end{array}
$$

PROOF: 1. The first goal is to prove

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

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

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

letting $i = seq'_{HE}$ we have

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

Choose AV_0 with $\underline{Seq}(\mathsf{AV}_0) = seq'_{HE}$. Now, from

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

it follows that $\forall_{\mathsf{AV}\in\mathsf{AVs\_new}}\ (\underline{Seq}(\mathsf{AV}) \leq \underline{Seq}(\mathsf{AV}_0))$, which may be written as $\mathsf{AV}_0 = \max(\mathsf{AVs\_new})$.

Using the definition of $\mathcal{N}^{SN}_{normal}$, we have that ADS($\mathsf{AVs\_new}$,$\mathsf{SN}$) implies $DB'(\mathsf{SN}) = \mathsf{AVs\_new}$ and therefore $\mathsf{AV}_0 = \max(DB'(\mathsf{SN}))$. Then

$$\underline{Seq}(\mathsf{AV}_0) = \underline{Seq}(\max(DB'(\mathsf{SN}))) = seq_{HE}{}'$$

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

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

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

Since

$$last\_SN\updownarrow \Rightarrow curr\_SN\updownarrow \wedge (\exists_{\mathsf{SN}_1}\ curr\_SN' = \{\mathsf{SN}_1\} \vee curr\_SN' = \emptyset)$$

but $curr\_SN' = \emptyset$ implies $last\_SN' = last\_SN$. Choosing a new fresh variable, we conclude that $curr\_SN\updownarrow \wedge curr\_SN' = \{\mathsf{SN}_1\}$.
From $\mathcal{A}^{CurrSN}_{normal} \wedge \mathcal{A}^{LastSN}_{normal}$ we obtain that LUR $\vee$ CanLoc and from $\mathcal{N}^{LU}_{normal}$ 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 $DB\updownarrow \wedge last\_SN' = last\_SN \wedge seq_{HE}{}' = seq_{HE}$, 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 $\mathrm{UAR_S}$ or MvAV happen ($\mathcal{N}^{SN}_{normal}$). 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 $\mathcal{N}^{MS}_{normal}$.

3. The third goal is that in each transition,

$$\mathrm{UAR_S}(\underline{Chall}(\mathsf{AV}), \mathsf{SN}) \Rightarrow \mathsf{AV} = \min(DB(last\_SN)).$$

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

Lemma 8.4.

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

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:*

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

A simple consequence of the last two lemmas is:

Lemma 8.6.

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

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

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

Lemma 8.8.

$$\mathcal{A}_{normal} \Rightarrow \Box(\ \mathrm{UAR_S}(\mathsf{chall}, \mathsf{SN}) \Rightarrow \underline{Seq}(\mathsf{chall}) - seq_{MS} < \Delta\ )$$

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

Lemma 8.9.

$$\begin{aligned}
&\mathcal{N}^{CC}_{normal} \wedge \mathrm{UAR_S}(\underline{Chall}(\mathsf{AV}), \mathsf{SN}) \Rightarrow \\
&\quad \vee\ \wedge Accepted' = Accepted \cup \{\mathsf{AV}\} \\
&\qquad \wedge {seq_{MS}}' = \underline{Seq}(\mathsf{AV}) \\
&\qquad \wedge Lost' = Lost \\
&\quad \vee\ \wedge Accepted' = Accepted \cup \{\mathsf{AV}\} \\
&\qquad \wedge {seq_{MS}}' = \underline{Seq}(\mathsf{AV}) \\
&\qquad \wedge Lost' = Lost \cup \{\mathsf{AV}\} \\
&\quad \vee\ \wedge Accepted' = Accepted \\
&\qquad \wedge {seq_{MS}}' = seq_{MS} \\
&\qquad \wedge Lost' = Lost \cup \{\mathsf{AV}\}
\end{aligned}$$

Lemma 8.10.

$$\begin{aligned}
&\mathcal{N}^{CC}_{normal} \wedge \mathrm{UAR_S}(\underline{Chall}(\mathsf{AV}), \mathsf{SN}) \wedge \mathrm{UAS_R}(\underline{Resp}(\mathsf{AV}), \mathsf{SN}) \Rightarrow \\
&\quad \wedge Accepted' = Accepted \cup \{\mathsf{AV}\} \\
&\quad \wedge {seq_{MS}}' = \underline{Seq}(\mathsf{AV}) \\
&\quad \wedge Lost' = Lost
\end{aligned}$$

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*[4]*:*

$$\mathcal{A}_{normal} \Rightarrow \Box(Perfect) \wedge \Box(\neg \mathrm{UASF_S})$$

PROOF: First let us see why *Perfect* is an invariant, i.e. $\mathcal{A}_{normal} \Rightarrow \Box(Perfect)$. It is clear that initially, *Perfect* holds, i.e. Init $\Rightarrow$ *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:*

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

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:*

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

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 Architechture*

[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 numger management scheme protecting agains USIM lockout.*

[4]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.