Chapter 3

Relaxing Trust Requirement in UMTS

Identity Privacy is considered a standard security feature in any mobile telecommunication system. UMTS is no exception and this requirement is clearly spelt out in 3GPP TS 33.102 [19]. However, there is a security vulnerability in UMTS, due to which the identity privacy of a subscriber gets compromised. In addition, in UMTS it is assumed that there is no threat from the SNs, even if they belong to third party operators. All of this may be attributed to the existing trust model adopted in UMTS.

In this chapter, we propose a new security extension called End to End User Identity Confidentiality (E2EUIC) for the AKA protocol used in UMTS. The extension is based on our proposed trust model (Chapter 2, Section 2.4), and has the potential to improve the subscriber’s identity privacy while relaxing trust requirement for roaming between mobile operators. While designing the extension, we question the trustworthiness of the SNs themselves, on which the existing mechanism to protect identity privacy is based. Our extension relaxes HN to SN and hence, UE to SN trust relationship requirement with respect to the subscriber's identity privacy. We propose to replace the transmission of IMSI with a Dynamic Mobile Subscriber Identity (DMSI). Unlike IMSI, the value of DMSI is not static, thus enhancing identity privacy/confidentiality.
3.1 Introduction

Like all other mobile systems proposed by 3GPP, a unique IMSI is assigned to identify the subscriber in UMTS. The identity privacy of the subscriber gets compromised if his/her IMSI gets exposed. Towards this end, 3GPP endeavours to limit the transmission of IMSI to the wired part of the network, as the wireless link is too open for various kinds of attacks [17][23].

In order to limit transmission of IMSI over the wireless link, the SN assigns a local temporary identity called Temporary Mobile Subscriber Identity (TMSI) to the UE through a ciphered channel. The value of this temporary identity is short lived and is refreshed frequently by the SN. The SN keeps the association between a TMSI and its corresponding IMSI in its local database. Whenever the UE needs to present its identity, it transmits the TMSI instead of its IMSI. The SN can easily correlate this TMSI to the corresponding IMSI through the TMSI to IMSI mapping maintained in its local database. If the subscriber roams into a new SN and it produces a TMSI obtained from the old SN, the association between the produced TMSI and the corresponding IMSI is obtained by the new SN from the old SN.

In UMTS, there are circumstances when a TMSI fails to identify the UE, forcing transmission of IMSI in clear-text over the wireless link. Such situations make identity privacy of the subscriber vulnerable [30]. Moreover, there is full trust relationship among the agents in the wired network and the IMSI is exchanged freely among them. Such trust requirement limits interoperability between mobile operators as it complicates roaming agreements.

3.2 Security Architecture of UMTS

The security architecture of UMTS (Figure 3.1) involves three primary participants, namely: the UE, the SN and the HN [31]. The UE is any device that is used by the subscriber to communicate. It can be a hand-held telephone, a laptop computer, or any other device that is fitted with a Universal Subscriber Identity Module (USIM) [32]. Every UE has to be registered with a HN (with
Chapter 3. Relaxing Trust Requirement in UMTS

Figure 3.1: Security architecture of UMTS.

their security credentials stored at the HN's database). The HN contains key security elements like the Home Location Register (HLR) and the Authentication Centre (AuC). The HLR stores permanent sensitive information of the subscribers such as identity, service profile, activity status, etc., whereas the AuC is a protected database that stores association between subscriber identities and long-term keys. The HN extends its services to its roaming subscribers through the SNs. The SN contains elements like the Visitor Location Register (VLR) and the Mobile Switching Center (MSC). The VLR stores temporary information about the subscribers visiting the SN and maintains temporary to permanent identity associations, whereas the MSC offer circuit-switching domain services. The UE directly communicates with a Base Transceiver Station (BTS) or NodeB (through the Uu reference point/interface). One or more NodeBs are connected with a Radio Network Controller (RNC) (through the lubis reference point/interface). The RNC manages the radio resources and is the interface between the UE and the core network (through the lucs reference point/interface). Communication between the UE and the SN happens over radio link, whereas communication between the SN and the HN happens through wired link. While the radio link is considered to be vulnerable, it is assumed that the wired links are adequately secure.
3.3 UMTS-AKA

UMTS-AKA is the AKA protocol used in UMTS for access security [19]. It is carried out in two stages [23][17], which are as follows:

- In the first stage, the UE presents its identity to the SN. The SN, with the help of this identity, obtains the security credentials of the UE in the form of a set of Authentication Vectors (AVs) from the HN.

- In the second stage, the SN utilises one of these AVs to perform mutual authentication with the UE through a challenge response mechanism. In this phase, a Cipher Key (CK) and an Integrity Key (IK) are established between the UE and the SN, so that communication over the otherwise vulnerable radio link (between the UE and the SN) can happen in a secured and reliable way.

In order to facilitate the authentication mechanism, each UE shares with its HN a long term secret key $K_i$ and certain cryptographic algorithms, viz., $f_0$, $f_1$, $f_2$, $f_3$, $f_4$, $f_5$, $f_6$ and $f_9$. Where, $f_0$ is the random challenge generating function, $f_1$ is the network authentication function, $f_2$ is the user challenge response authentication function, $f_3$ is the cipher key derivation function, $f_4$ is the integrity key derivation function, $f_5$ is the anonymity key derivation function, $f_6$ is the confidentiality key stream generating function and $f_9$ is the integrity stamp generating function. A set of example algorithms for these functions called MILENAGE are proposed in [33]. In order to assure freshness of authentication data, two counters, viz., $SQN_{UE}$ and $SQN_{HN}$ are maintained at the UE and the HN respectively. Detailed functionality of both the stages of UMTS-AKA are described in the following two subsections:

3.3.1 Distribution of Authentication Data

1. The UE presents its identity to the SN by transmitting it through the radio channel.

2. In case, the presented identity is a temporary identity, the SN locates the
corresponding IMSI using the TMSI-IMSI mapping maintained in its local database. If the SN already has unused authentication data stored in its local database against this IMSI, the remaining steps of this stage are skipped. Otherwise, the SN sends an authentication data request to the HN along with the UE’s IMSI.

3. Upon receipt of the request, the HN generates an ordered array of $M$ authentication vectors denoted by $AV[1..M]$. Each $AV$ is a quintet, consisting of five elements, viz.: a Random Number ($RAND$), an Expected Response ($XRES$), a Cipher Key ($CK$), an Integrity Key ($IK$), and an Authentication Token ($AUTN$). An $AV$ in $AV[1..M]$ is generated according to the following steps (Figure 3.2):

(a) The HN generates a Random Number $RAND$ using the function $f_0$, and a Sequence Number $SQN$ from the counter $SQN_{HN}$.

(b) The HN then calculates the following values:

\[
XRES = f_{2K_1}(RAND) 
\]
\[
CK = f_{3K_1}(RAND) 
\]
\[
IK = f_{4K_1}(RAND) 
\]
\[
AK = f_{5K_1}(RAND) 
\]
\[
MAC = f_{1K_1}(SQN \| RAND \| AMF) 
\]
Chapter 3. Relaxing Trust Requirement in UMTS

Figure 3.3: Content of AV.

Where, $AK$: Anonymity Key, $MAC$: Message Authentication Code, $AMF$: Authentication and Key Management Field, and ‘||’ denote concatenation. $AK$ is used to conceal the sequence number, as the later may expose the location of the user. If no concealment is needed, $AK$ is set to zero.

(c) After this, HN assembles the Authentication Token:

$$AUTN = SQN \oplus AK \| AMF \| MAC$$

(3.3.6)

and the Authentication Vector:

$$AV = (RAND, XRES, CK, IK, AUTN)$$

(3.3.7)

where ‘$\oplus$’ is bit wise Exclusive OR operation (Figure 3.3).

(d) The HN increments $SQN_{HN}$ by 1.

4. Finally, the HN sends $AV[1..M]$ back to the SN.

3.3.2 Authentication and Key Agreement

1. The SN selects the first unused $AV$ from the received $AV[1..M]$. It then extracts $RAND$ and $AUTN$ from the selected $AV$ and sends it to the UE as a challenge.

2. Upon receipt of $RAND$ and $AUTN$, the UE calculates $AK$ using Equation 3.3.4. Using the calculated $AK$, the sequence number $SQN$ is then
Figure 3.4: UMTS authentication and key agreement.

retrieved from \textit{AUTN} (Equation 3.3.6) and compared with \textit{SQN}_{UE} in order to verify freshness of the challenge. The UE then computes \textit{MAC} using Equation 3.3.5 and compares this value with the \textit{MAC} included in \textit{AUTN} (Equation 3.3.6). If they are different, the UE rejects the connection procedure, otherwise it accepts it. Finally, the UE computes the following:

\[ RES = f_{2K_i}(RAND) \]  
(3.3.8)

and sends \textit{RES} back to the \textit{SN}.

3. Upon receipt of the \textit{RES}, the \textit{SN} compares it with \textit{XRES} (\textit{XRES} is a constituent of the selected \textit{AV}, Equation 3.3.7). If these values match, the authentication process is considered successful. \textit{CK} and \textit{IK}, calculated at either end (using Equation 3.3.2 and Equation 3.3.3) are used to secure further communications between the \textit{SN} and the UE.

The UMTS-AKA procedure is schematically expressed in Figure 3.4.

### 3.4 Identity Privacy in UMTS

To achieve identity privacy in UMTS, temporary identities (i.e., \textit{TMSIs}) are used. A \textit{TMSI} is assigned to the UE by the \textit{SN} only after a secure channel is established between them. The channel has to be secured using \textit{CK} and \textit{IK} generated during the previous successful UMTS-AKA. The \textit{TMSI}, when
available, is used instead of the IMSI to identify the subscriber over the radio access path, for paging requests, location update requests, attach requests, service requests, connection re-establishment requests and detach requests. A TMSI only has local significance in the location area or the routing area in which the user is roaming. Outside that area, it should be appended with an appropriate Location Area Identification (LAI) or Routing Area Identification (RAI) in order to avoid ambiguities. The association between a TMSI and its IMSI is maintained at the SN. To avoid user traceability, which may lead to the compromise of identity privacy, the user should not be identified by means of the same TMSI for a long period. The allocation of a new TMSI is initiated by the SN. The SN generates a new TMSI (say TMSI_n) and stores the association of TMSI_n and the IMSI in its database. TMSI_n should be unpredictable. The SN then sends this new TMSI_n and (if necessary) the new location area identity (say LAI_n) to the user through a ciphered channel. Upon receipt, the UE stores TMSI_n and automatically removes the association with any previously allocated TMSI. The UE sends an acknowledgement back to the SN. Upon receipt of the acknowledgement, the SN removes the association (if there was any) between the old temporary identity TMSI_o and the IMSI from its database. If the SN does not receive an acknowledgement from the UE (informing it of the successful allocation of a temporary identity), the SN shall maintain both the TMSI_n to IMSI and TMSI_o to IMSI associations.

When the subscriber roams into a new region, he/she presents his/her identity to the new SN (say SN_n) by transmitting the TMSI that was allocated to it by the old SN (say SN_o) along with the identity of SN_o. SN_n obtains the association between the TMSI and the IMSI from SN_o, and uses this IMSI to request necessary authentication data from the HN.

### 3.5 Motivation

In spite of the security mechanism used for identity privacy in UMTS (Section 3.4), there are situations in UMTS-AKA when the identity privacy of a subscriber
becomes vulnerable. Such situations, which are also the motivation of the work behind this chapter, are as follows:

- **The UE is switched on for the first time and has not yet received a TMSI:** In such a situation, the UE is forced to present its identity by transmitting its IMSI in clear-text through the radio link (Message 1, Figure 3.5).

- **The SN cannot map a presented TMSI to its corresponding IMSI:** In such a situation (that may arise due to reasons like database failure, etc.), the SN has a provision to request the UE for its IMSI. Such a request requires the UE to transmit its IMSI in clear-text through the radio link.

- **A new SN cannot contact the old SN for the TMSI-to-IMSI mapping of a roaming subscriber:** When a subscriber moves into the region of a new SN (say SN_n), the UE will present its identity to SN_n through the TMSI allocated to it by the previous SN (say SN_o) along with the LAI of SN_o (Message 7, Figure 3.5). In order to request for a new set of AV from the HN, SN_n will need to have knowledge of the IMSI. Normally, this will be obtained by presenting the TMSI to SN_o. However, in case SN_o cannot be contacted, SN_n will be forced to ask the UE for its IMSI. The later
Chapter 3. Relaxing Trust Requirement in UMTS

will then have to be transmitted in clear-text over the radio link by the UE (Message 10, Figure 3.5). This vulnerability can in fact be exploited by an attacker who can masquerade as a new SN.

- UMTS-AKA assumes full trust relationship within the wired intermediary service network components, and hence IMSI is transmitted freely between the SN and the HN. But, in practice the HN operator does not own all the SNs through which it provides services to its roaming subscribers, and as such trusting the SNs with the IMSI may end up compromising the identity privacy of the subscriber.

3.6 UMTS-AKA-with-E2EUIC

In this section, we propose a new security extension called End to End User Identity Confidentiality (E2EUIC) [34]. This extension, which is based on our trust model proposed in Chapter 2, Section 2.4, when adopted to the AKA protocol used in a 3GPP mobile system, has the potential to enhance identity privacy and relax trust requirement for roaming. Here, we present this extension with reference to UMTS-AKA (the authentication and key agreement protocol used for access security in UMTS). E2EUIC not only takes care of identity privacy over the wireless network, but goes one step ahead to ensure the same over the wired part as well. It enables mutual authentication without requiring the SNs to have access to the IMSI of the UE. This ensures that even hostile SNs that are placed in between the UE and the HN will not be able to compromise identity privacy, thereby relaxing the need for the HN to trust the SN with respect to identity privacy during roaming agreements, specially when the HN and the SN belongs to two different operators. Thus, we call the proposed extension as End to End User Identity Confidentiality. E2EUIC achieves enhanced identity privacy without forcing any change in the intermediary network. It does not require the UE to transmit its IMSI at any stage of the protocol flow. We propose to replace transmission of the IMSI with a Dynamic Mobile Subscriber Identity (DMSI). A fresh DMSI is created as and when its need arises, and
its value is derived from the most recent random number received as a challenge during a successful UMTS-AKA procedure. As a result, transmission of a $DMSI$ does not compromise the permanent identity of the user. The extension can be introduced in the existing system as an optional service, with the subscriber requiring to collect a new USIM in place of his/her existing USIM, or can be introduced on a rolling basis as new USIMs are issued.

In order to enable the UE to create a $DMSI$, a new random number called Random number for Identity Confidentiality ($RIC$) is introduced in the security extension. The $DMSI$ is a function of this $RIC$, details of which is explained later in this section.

The HN maintains a pool of $RIC$s in its local database (i.e., HLR/AuC), some of which are in-use (i.e., already assigned to different UEs) and some of which are not-in-use (unassigned) at an instant of time. During every successful run of the UMTS-AKA protocol, a not-in-use $RIC$, randomly selected from the pool, is securely transferred to the UE. The HN uses this $RIC$ to uniquely identify the UE for an epoch (explained later in this section) of time. The selected $RIC$ has to be sufficiently random, such that there is no correlation with a previously selected $RIC$. A mapping between the selected $RIC$ and the $IMSI$ of the UE is maintained at the HN's local database (HLR/AuC), so that the HN can uniquely identify the subscriber/UE through this mapping at a later instant (for purposes like billing, generation of AVs, etc.). Thus, whenever the UE needs to present its permanent identity, it assembles a $DMSI$ with the most recently received $RIC$ and transmits it instead of the $IMSI$. The HN in turn, extracts the $RIC$ from the received $DMSI$ and identifies the subscriber by referring to the $RIC$ to $IMSI$ mapping maintained in its database (i.e., HLR/AuC).

In some exceptional situations like failure of an ongoing UMTS-AKA or due to an active attack by an adversary, the UE may not receive the next $RIC$ (from the HN) after it has already used the most recently received $RIC$ to create and transmit a $DMSI$. In such a situation, if the need to transmit a $DMSI$ arises again, the UE can reuse the most recently received $RIC$ to create the next $DMSI$. This can continue, as long as the UE does not receive a fresh $RIC$ from
the HN (during a successful UMTS-AKA). Even though such a mechanism, in the worst case, may allow an adversary to link two or more failed UMTS-AKA of the same UE, an adversary cannot gain anything from this in terms of compromised identity privacy. Moreover, it is a much better option than transmitting the IMSI itself.

To securely transfer a not-in-use RIC to the UE during a run of the AKA protocol (Section 3.3), a fresh not-in-use RIC (RIC\textsubscript{Fresh}) is selected at the HN. RIC\textsubscript{Fresh} is then embedded into the RAND part of each AV in AV\textsubscript{[1..M]} (Equation 3.3.7). For embedding, the long term secret key Ki and an embedding algorithm are used. The resultant random number after embedding a RIC into a RAND is called an Embedded RAND (ERAND). Thus, during a run of the UMTS-AKA protocol with E2EUIC extension, an ERAND (which is of the same size as the RAND, i.e., 128 bit) is now send as a challenge to the UE instead of a RAND. The UE, having knowledge of the long term shared key Ki, can easily extract RIC\textsubscript{Fresh} from the received ERAND. The rest of the AKA procedure continues in the same way as in UMTS-AKA, the sole difference being the use of ERAND in all purposes where the RAND was used earlier. The intermediary networks does not have to bother about this difference, as the size of ERAND and RAND are same, and they can continue to operate as before.

The mechanism proposed in the above paragraph cannot provide a RIC to the UE that is required for identity presentation during the first UMTS-AKA-with-E2EUIC in the USIM's life time. This is because, the first identity presentation precedes all AKAs. Thus, an alternate mechanism is used for this purpose. This mechanism is carried out before distribution of the USIM, i.e., before a subscriber procures the USIM from the mobile operator. According to this mechanism, an ERAND (Say ERAND\textsubscript{First}) that has a unique RIC (say RIC\textsubscript{First}) embedded into it is stored into the USIM's flash memory. RIC\textsubscript{First} is meant to be used only for the first successful authentication in the USIM's life time.

We propose the size of RIC to be of 32 bits. Choice of 32 bits for RIC is inspired by the size of TMSI used in UMTS-AKA [5]. With this size, a SN is
easily able to allocate unique $TMSIs$ to all the UEs under its service area. A 32 bit $RIC$ provides a pool of $2^{32} = 4.29$ billion (approx) unique $RIC$ values. However, size of the $RIC$ may even be determined by the operator depending on the anticipated subscriber base of the HN (provided it is lesser than 128 bits). A $RIC$ of size $b$ bits, provide a pool of $n$ unique $RIC$ values. Where,

$$n = 2^b \quad (3.6.1)$$

The HN needs to store in its local database multiple ($m$) $RIC$s against the $IMSI$ of a particular UE (Figure 3.6). Multiple $RIC$s are stored in order to deal with the following exceptional situations:

- **An AV or a sequence of AVs gets lost in transit or does not get utilised**: In such situations, the most recently selected $RIC$ that is stored at the HN’s local database against the $IMSI$ of the UE will not reach the UE. Due to which, the UE will continue to consider the $RIC$ that it received during the last successful authentication as the current $RIC$, using which it may create and transmit the next $DMSI$. Thus, a history of $m - 1$ most recent $RIC$s
Chapter 3. Relaxing Trust Requirement in UMTS

generated against the UE needs to be maintained against its IMSI at the HN’s database (in the fields say RIC_{New}, RIC_{Prev}, RIC_{Old}, etc.); the value of \( m \) is to be decided by the mobile operator depending on the probability of AVs getting lost in transition. This ensures that a mapping between the RIC that is currently stored at the UE and the corresponding IMSI is always maintained at the HN. However, like any other critical information such as the subscriber’s security credentials, billing details, etc., in case of the RICs maintained in the HN’s database also, it is the responsibility of the operator to have a robust backup mechanism against database crash.

Whenever a new RIC (i.e., RIC_{Fresh}) is generated at the HN, the oldest RIC (i.e., RIC_{Old}) is discarded (returned to the pool of not-in-use RICs) and the values stored in the \( m-2 \) other RIC fields (i.e., RIC_{New}, RIC_{Prev}, etc.) are shifted to their next older fields (i.e., RIC_{Prev} is shifted to RIC_{Old}, RIC_{New} is shifted to RIC_{Prev}, etc.). These adjustments are done to make space for RIC_{Fresh} in the HN’s local database.

- **The RIC contained in the DMSI that is being used by the SN to identify the UE gets deleted from the HN’s database:**

When the subscriber enters the service area of a new SN, the very first identity that it uses to identify itself to this SN is a DMSI rather than a TMSI or the IMSI. This DMSI is used by the SN to uniquely identify the UE for purposes like billing and collection of AVs. The SN uses this DMSI as long as it does not receive the next DMSI (during a successful authentication) from the UE or till the subscriber does not leave its service area - which ever happens earlier. When the next DMSI is received from the UE, the SN discards the previous DMSI and starts using the newly received DMSI to uniquely identify the UE.

If a roaming subscriber continuously stays with the SN through several authentications (requiring the SN to collect more than \( m \) ordered array of AVs (i.e., AV[1..M]) from the HN), a time will come when the RIC contained in the DMSI that is being used by the SN to uniquely identify the UE gets removed (returned to the pool of not-in-use RICs) from the
HN’s database. Thus, in order to deal with this situation, an additional field called $RIC_{\text{InUse}}$ is maintained at the HN’s database against the IMSI of the UE. While the $m - 1$ other RIC values stored against the IMSI in the fields: $RIC_{\text{New}}, RIC_{\text{Prev}}, RIC_{\text{Old}},$ etc., keeps changing, the RIC value stored against $RIC_{\text{InUse}}$ changes only when a new DMSI is selected by the SN to uniquely identify the UE.

If $s$ is the maximum number of subscribers that a mobile operator wants the proposed extension to handle, then

$$s = \frac{n}{m} \quad (3.6.2)$$

where, $n$ (Equation 3.6.1) is the total number of possible RICs in the entire pool and $m$ is the number of RICs maintained against each IMSI in the HN’s database (i.e., HLR/AuC). We propose the value of $m$ to be 4. However, an operator may choose to have a different value for $m$ depending on its anticipated subscriber base. With $m = 4$, a 32 bit RIC will enable the HSS to have at the most $2^{30} = 1.073$ billion (approx) subscribers, which is 5.73 times more than the 187.302 million (approx) subscriber base of the largest mobile operator in India as of June, 2012 [35]).

Before distribution of an USIM, it has to be initialised. During this, all the $m$ RIC fields (maintained in the HN’s database against the IMSI of the USIM) are assigned randomly selected not-in-use RICs. Out of all the assigned RIC values, the value that is assigned against the $RIC_{\text{New}}$ field is chosen as $RIC_{\text{First}}$ and is later transferred to the UE as already explained earlier in this section.

In order to verify the freshness of a received DMSI and to prevent replay attacks, the HSS maintains an additional field called $SEQ_{\text{HN}}$ against every IMSI in its database. $SEQ_{\text{HN}}$ is used to store the sequence number of the most recent DMSI received from the UE.

In order to quickly locate a RIC in the HN’s database, a database index called $RIC$-index is maintained at the HN (Figure 3.6). The $RIC$-Index contains all the $n$ possible RICs sorted according to their values. Each entry in the $RIC$-Index contains a pointer against it, which is called an IMSI-Pointer. This pointer
either points to an IMSI in the HN's database or is null, depending on whether that particular RIC is allocated to an UE or is unallocated at a particular instance of time. The collection of all the RICs in the RIC-Index having a null value against it, forms the pool of not-in-use RICs, whereas the rest of the RICs in the RIC-Index that points to some IMSI, represents the RICs that are in-use. The total number of entries in the RIC-Index is fixed at $n$, irrespective of the number of RICs that are currently in-use in the HN's database. Even though such an index would require more disk space (Section 6.5), compared to an index whose size grows and shrinks according to the number of RICs that are in-use at a particular instance in the HN's database, it relieves the HN of computational overhead involved during frequent insertions and deletions in the index.

### 3.6.1 The Protocol Flow

E2EUIC, is implemented in the USIM of the UE and in the HLR/AuC of the HN. In UMTS-AKA-with-E2EUIC, a DMSI is transmitted instead of the IMSI (Figure 3.7). The role of the TMSI remains same as in the original UMTS-AKA. A fresh DMSI is created in the USIM only when its need arises, i.e., during the first AKA in the USIM's life time and when the UE receives a request for the permanent identity from the SN. The current value of the DMSI depends on the most recently received RIC by the UE. A DMSI is a concatenation of the Mobile Country Code (MCC), the Mobile Network Code (MNC), the most recent RIC received by the UE, and an Encrypted RIC called ERIC. Since DMSI is calculated using short-lived RIC values, knowledge of the former does not compromise the actual identity of the UE. The protocol flow during the first UMTS-AKA-with-E2EUIC in the life time of a USIM is as follows (a list of the notations used and their brief description are presented in Table 3.1).
Chapter 3. Relaxing Trust Requirement in UMTS

Figure 3.7: Protocol flow of UMTS-AKA-with-E2EUIC.

The First UMTS-AKA-with-E2EUIC

(1.1) The \( RIC \) (i.e., \( RIC_{\text{First}} \)) stored in \( ERAND_{\text{First}} \) is extracted using an operator specific function \( f_{\text{Extract}} \).

\[
RIC_{\text{First}} = f_{\text{Extract}_{K_i}}(ERAND_{\text{First}}) \tag{3.6.3}
\]

An example algorithm for \( f_{\text{Extract}} \) is presented in Section 3.7. For identity presentation, the UE creates a \( DMSI \) (say \( DMSI_1 \)) using \( RIC_{\text{First}} \) as follows:

\[
DMSI_1 = MCC || MNC || RIC_{\text{First}} || ERIC \tag{3.6.4}
\]

where, \( ERIC \) is created by encrypting a padded \( RIC \) (say \( RIC_{\text{padded}} \)) with the Advanced Encryption Standard (AES) algorithm, taking the long term secret key \( K_i \) as parameter. Thus,

\[
ERIC = f_{\text{Encrypt}_{K_i}}(RIC_{\text{padded}}) \tag{3.6.5}
\]

35
### Chapter 3. Relaxing Trust Requirement in UMTS

#### Table 3.1: Notations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_{IC\text{Fresh}}$</td>
<td>A fresh not-in-use RIC.</td>
</tr>
<tr>
<td>$R_{IC\text{First}}$</td>
<td>A not-in-use RIC embedded in the USIM before the USIM is distributed.</td>
</tr>
<tr>
<td>$R_{IC\text{New}}$</td>
<td>A RIC maintained against the IMSI at the HN's database.</td>
</tr>
<tr>
<td>$R_{IC\text{Prev}}$</td>
<td>A RIC maintained against the IMSI at the HN's database.</td>
</tr>
<tr>
<td>$R_{IC\text{Old}}$</td>
<td>A RIC maintained against the IMSI at the HN's database.</td>
</tr>
<tr>
<td>$R_{IC\text{InUse}}$</td>
<td>A RIC maintained at the HN's database against the IMSI, a copy of which is currently being used at the SN to identify the UE.</td>
</tr>
<tr>
<td>$SEQ_{UE}$</td>
<td>Value of a DMSI counter maintained at the UE.</td>
</tr>
<tr>
<td>$SEQ_{HN}$</td>
<td>Sequence number maintained at the HN to check freshness of a received DMSI.</td>
</tr>
<tr>
<td>$R$</td>
<td>128-(32+b) bit random number.</td>
</tr>
<tr>
<td>$R_{IC\text{Padded}}$</td>
<td>$RIC | SEQ_{UE} | R$; where, $|$ indicates concatenation.</td>
</tr>
<tr>
<td>$ERAND_{UE}$</td>
<td>Variable maintained in the USIM's flash memory to store the most recently received ERAND.</td>
</tr>
<tr>
<td>$DMSI_{SN}$</td>
<td>Variable maintained in the SN to uniquely identify the UE as long as it does not receive a new DMSI (during a successful AKA).</td>
</tr>
<tr>
<td>$TTL_{DMSI}$</td>
<td>Time to live for DMSI.</td>
</tr>
<tr>
<td>$I_{Embed}$</td>
<td>Embeds a RIC into a RAND to find an ERAND.</td>
</tr>
<tr>
<td>$I_{Extract}$</td>
<td>Used to extract an embedded RIC from an ERAND.</td>
</tr>
<tr>
<td>$I_{Encrypt}$</td>
<td>An AES standard encryption algorithm that encrypts $RIC_{Padded}$ to find an ERIC.</td>
</tr>
<tr>
<td>$I_{Decrypt}$</td>
<td>An AES standard decryption algorithm that decrypts an ERIC to find $RIC_{Padded}$.</td>
</tr>
<tr>
<td>$I_{PRNG}$</td>
<td>A pseudorandom number generator.</td>
</tr>
<tr>
<td>$m$</td>
<td>Total number of $RIC$s maintained at the HN's database against a particular IMSI.</td>
</tr>
<tr>
<td>$b$</td>
<td>Number of bits in a $RIC$.</td>
</tr>
<tr>
<td>$n$</td>
<td>Total number of $RIC$s in the pool of $RIC$s maintained at the HN.</td>
</tr>
<tr>
<td>$s$</td>
<td>Maximum number of subscribers that a mobile operator wants to accommodate.</td>
</tr>
</tbody>
</table>
where,

\[ RIC_{\text{padded}} = RIC_{\text{First}} \| SEQ_{UE} \| R \]  \hspace{1cm} (3.6.6)

\( SEQ_{UE} \) is the value of a 32 bit counter that is maintained at the UE (USIM’s flash memory); whenever a new \( DMSI \) is created for identity presentation, \( SEQ_{UE} \)'s value is incremented by one. \( R \) is a \( 128 - (32 + b) \) bit random number. The inclusion of \( SEQ_{UE} \) ensures freshness of the \( DMSI \)s, whereas the inclusion of \( R \) completes the block size of 128 bits that is necessary to be fed into the AES cipher. In addition, \( R \) introduces significant amount of randomness to harden cryptanalysis of the cipher text.

Finally, \( DMSI_1 \) is transmitted to the SN.

In this step, we selected AES algorithm for the purpose of creating ERIC, due to the following reasons:

- AES (originally called Rijndael) is fast in both software and hardware \([36]\)
- The example algorithm set presented by 3GPP in \([33]\) is based on Rijndael that eventually got selected as AES standard \([37]\); Rijndael as being then one of the five remaining AES candidates, was well studied.
- Until May 2009, the only successful published attacks against the full AES were side-channel attacks on some specific implementations \([38]\).

Depending on availability of better functions, this choice could be replaced by any suitable 128-bit keyed function employing a 128 bit key.

(1.2) The SN temporarily stores \( DMSI_1 \) in its local database, as it would require this \( DMSI \) to uniquely identify the UE for a period of time (till it receives the next \( DMSI \) from the UE during a successful UMTS-AKA) if this run of the UMTS-AKA eventually succeeds. The SN identifies the HN of the UE by inspecting the \( MCC \) and \( MNC \) portion of \( DMSI_1 \). A request for a fresh set of \( AV \) is then sent along with \( DMSI_1 \) to the HN.
Chapter 3. Relaxing Trust Requirement in UMTS

(1.3) On receiving the request, the HN separates \( RIC_{First} \) from \( DMSI_1 \). It then uses the \( RIC-Index \) to locate \( RIC_{First} \) and hence the \( IMSI \) and the key \( K_i \) of the UE in the HN's database (Figure 3.6). After this, the \( ERIC \) part of the \( DMSI \) is decrypted using AES and the key \( K_i \). Thus,

\[
RIC_{Padded} = f_{Decrypt_K_i}(ERIC) \tag{3.6.7}
\]

The \( RIC \) contained in \( RIC_{Padded} \) is then compared with the \( RIC \) part of \( DMSI_1 \) (i.e., \( RIC_{First} \)); success of this comparison ensures that a malicious agent did not create \( DMSI_1 \). The \( SEQ_{UE} \) part of \( RIC_{Padded} \) is then compared with the value stored against \( SEQ_{HN} \) field in the HN’s database (HLR/AuC). If \( SEQ_{UE} > SEQ_{HN} \), the request is proven as fresh (not a replay of any previous request). Failure of any of these two comparisons, leads to rejection of the request. If the request for \( AV \) is found to be fresh and from a genuine source (from the above two comparisons), the following are performed:

(a) \( SEQ_{UE} \) is assigned to \( SEQ_{HN} \).

\[
SEQ_{HN} = SEQ_{UE} \tag{3.6.8}
\]

(b) If \( RIC_{First} \) is stored in any of the \( RIC \) fields other than \( RIC_{InUse} \) in the HN’s/HSS’s database, \( RIC_{First} \) is moved from its current location (say \( L \)) to \( RIC_{InUse} \) and the \( RIC \) stored earlier against \( RIC_{InUse} \) is moved to \( L \). In other words, \( L \) and \( RIC_{InUse} \) swaps their values. For example: if \( RIC_{First} \) is found in \( RIC_{New} \) then:

\[
temp = RIC_{InUse} \tag{3.6.9}
\]

\[
RIC_{InUse} = RIC_{New} \tag{3.6.10}
\]

\[
RIC_{New} = temp \tag{3.6.11}
\]

This is done to ensure that a mapping between \( RIC_{First} \) and the \( IMSI \) of the UE is maintained in the HN’s database, as long as \( DMSI_1 \) is used by the SN to uniquely identify the subscriber; in other words, to ensure that \( RIC_{First} \) doesn’t get removed from the
HN's database, while $DMSI_1$ is still being used by the SN to uniquely identify the subscriber. The value of $RIC_{InUse}$ does not change till a new genuine request for authentication data along with a new $DMSI$ (with a $RIC$ value that is different from the one which is stored in $RIC_{InUse}$) does not reach the HN. $RIC$ values stored against all the $m - 1$ other $RIC$ fields, viz., $RIC_{Old}$, $RIC_{Prev}$, $RIC_{New}$, etc., eventually gets removed from the HN's database after generation of $m$ $AV[1..M]$s.

(c) A fresh array of AVs (say $AV_{i[1..M]}$) is generated using the procedure used in UMTS-AKA (Equation 3.3.7).

After this, HN selects a fresh not-in-use $RIC$ (say $RIC_{Fresh}$) from the pool of $RIC$s ($RIC$-Index). In order to select $RIC_{Fresh}$, a $b$ bit random number (say $RN$) is generated using a standard Pseudo Random Number Generator (PRNG). For this, we propose to use National Institute of Standards and Technology (NIST) recommended random number generator based on ANSI X9.31 Appendix A.2.4 Using AES [39], which appears in the list of approved random number generators for Federal Information Processing Standards Publication (FIPS PUB) 140-2 [40]. With a 128 bit key, this PRNG generates a 128 bit random number, the $b$ most significant bits of which is selected as $RN$.

$$RN = f_{PRNG}(seed) \quad (3.6.12)$$

This $RN$ is then searched for in the $RIC$-Index. If the $IMSI$-Pointer against $RN$ in the $RIC$-Index is found to be null, $RN$ is selected as $RIC_{Fresh}$ and the null value is replaced with the address of the record in the HSS's database where the $IMSI$ is stored.

$$RIC_{Fresh} = RN \quad (3.6.13)$$

$$RN.IMSI$-Pointer = Address \; of \; IMSI \quad (3.6.14)$$

The oldest $RIC$ value (i.e., $RIC_{Old}$) stored against the $IMSI$ is then returned to the pool of not-in-use $RIC$ by searching for it in the $RIC$-Index.
and by setting the \textit{IMSI-Pointer} against it to \textit{null}.

\[ RIC_{\text{Old}}.\text{IMSI-Pointer} = \text{null} \quad (3.6.15) \]

In case the \textit{IMSI-Pointer} against \textit{RN} in the \textit{RIC-Index} is not \textit{null}, it may be inferred that there is a collision, and \textit{RN} is currently in-use. For collision resolution, a \textit{b bit} variable called \textit{Variable for Collision Resolution (VCR)} is used (Figure 3.6). The \textit{VCR} contains a not-in-use \textit{RIC}; an indication of this fact is specified in the \textit{RIC-Index} by setting the \textit{IMSI-Pointer} against the value in \textit{VCR} to the address of \textit{VCR}. At the very outset, during initialisation of the HSS's database, a \textit{b bit} random number (say \textit{RN}$_0$) is stored in the \textit{VCR} and the \textit{IMSI-Pointer} against it in the \textit{RIC-Index} is set to the address of \textit{VCR}.

\[ RN_0 = f_{\text{PRNG}}(\text{seed}) \quad (3.6.16) \]

\[ VCR = RN_0 \quad (3.6.17) \]

\[ RN_0.\text{IMSI-Pointer} = \text{Address of VCR} \quad (3.6.18) \]

Whenever there is a collision, the \textit{b bit} value stored in the \textit{VCR} is selected as \textit{RIC$_{\text{Fresh}}$}. \textit{VCR} is then searched for in the \textit{RIC-Index} and the \textit{IMSI-Pointer} against it in the \textit{RIC-Index} is made to point to the record in the HSS's database where the \textit{IMSI} is stored.

\[ RIC_{\text{Fresh}} = VCR \quad (3.6.19) \]

\[ VCR.\text{IMSI-Pointer} = \text{Address of IMSI} \quad (3.6.20) \]

In order to replace the \textit{RIC} stored in the \textit{VCR} with a fresh \textit{RIC}, the oldest \textit{RIC} (i.e., \textit{RIC$_{\text{Old}}$}) stored against the \textit{IMSI} is copied into \textit{VCR}. \textit{RIC$_{\text{Old}}$} is then searched for in the \textit{RIC-Index} and the \textit{IMSI-Pointer} against it is set to the address of \textit{VCR}.

\[ VCR = RIC_{\text{Old}} \quad (3.6.21) \]

\[ RIC_{\text{Old}}.\text{IMSI-Pointer} = \text{Address of VCR} \quad (3.6.22) \]

Software heuristics for generating empirically strong random number sequences rely on entropy gathering by measuring unpredictable external
Chapter 3. Relaxing Trust Requirement in UMTS

The above procedure used to refresh the VCR introduces ample entropy to make the selection procedure of RIC sufficiently random, because it is impossible to predict which IMSI’s RICOld value will refresh the VCR during the next AKA at the HN. It solely depends on the call timing and usage pattern of all the active subscribers registered with the HN.

RICFresh is then embedded into the RAND part of all the $M$ AVs of $AV_1[1..M]$ using an operator specific function $f_{Embed}$. We propose an example algorithm for $f_{Embed}$ in Section 3.7. We call the resultant number after embedding RICFresh into a RAND as an ERAND. Thus, all the $M$ ERANDs for $AV_1[1..M]$ are derived as follows:

$$ERAND_{1x} = f_{Embed}(RIC_{Fresh}, RAND_{1x}) \quad (3.6.23)$$

where, $x = 1, 2, 3, ..., M$. Therefore, each AV quintet of $AV_1[1..M]$ will now have an ERAND in it, instead of a RAND.

$$AV_{1x} = (ERAND_{1x}, XRES_{1x}, CK_{1x}, IK_{1x}, AUTN_{1x}) \quad (3.6.24)$$

where, $x = 1, 2, 3, ..., M$. From now on, an ERAND is used for all purposes where a RAND is used in UMTS-AKA (this will not have any impact on the protocol flow, as the size of RAND and ERAND are same (128 bit)). A copy of RICFresh is also stored at the HN’s database against the IMSI of the subscriber. For this purpose, RICOld is replaced by RICPrev, RICPrev is replaced RICNew and so on. And finally, RICNew is replaced by RICFresh. An entry in the RIC-Index against the IMSI-Pointer of RICFresh is also made accordingly. Thus,

$$RIC_{Old} = RIC_{Prev} \quad (3.6.25)$$

$$RIC_{Prev} = RIC_{New} \quad (3.6.26)$$

$$RIC_{New} = RIC_{Fresh} \quad (3.6.27)$$

$$RIC_{Fresh}, IMSI-Pointer = address \_ of \{IMSI\} \quad (3.6.28)$$

Finally, HN sends $AV_1[1..M]$ along with $DMSI_1$ back to the SN.
Chapter 3. Relaxing Trust Requirement in UMTS

(1.4) On receipt, SN continues the AKA procedure by extracting the ERAND and AUTN part of the first unused AV of \( AV_{1}[1..M] \) (i.e., \( ERAND_{1} \) and \( AUTN_{1} \)). \( ERAND_{1} \) and \( AUTN_{1} \) are then transmitted as a challenge to the UE.

(1.5) The UE and the SN completes the remaining part of the UMTS-AKA-with-E2EUIC extension, following the same steps as in UMTS-AKA. On successful completion of the mutual authentication process, the following additional steps are carried out by the UE and the SN:

(a) The UE saves the recent \( ERAND_{1} \) that it received from the SN in a field (say \( ERAND_{UE} \)) in the USIM’s flash memory.

(b) The SN stores \( DMSI_{1} \) in a variable say \( DMSI_{SN} \) in its local database. The value of this variable does not change till the SN does not receive a new \( DMSI \) during a successful UMTS-AKA-with-E2EUIC. The SN uses \( DMSI_{SN} = DMSI_{1} \) to uniquely identify the UE as long as it does not receive a new \( DMSI \). The same field that the SN uses to store the IMSI can be used as \( DMSI_{SN} \).

(c) The remaining AVs in \( AV_{1}[1..M] \) are stored against \( DMSI_{SN} \) at SN’s database for future authentications.

(1.6) At the end of the AKA procedure, a pair of Cipher Key (\( CK \)) and an Integrity Key (\( IK \)) is established between the UE and the SN, following the same procedure as in UMTS-AKA (Equation 3.3.2 and Equation 3.3.3). A secure and reliable channel is then created between the UE and the SN using these two keys. After this, a Temporary Mobile Subscriber Identity (say \( TMSI_{1} \)) generated by the SN is securely communicated to the UE through this channel. The UE stores the received \( TMSI_{1} \) in the USIM’s flash memory (in a field say \( TMSI_{UE} \)) for identity presentation during the next authentication. A mapping between \( TMSI_{1} \) and \( DMSI_{SN} \) is also maintained in the SN’s local database. If the UE uses \( TMSI_{1} \) to identify itself in the next authentication, the \( TMSI_{1} \)-to-\( DMSI_{SN} \) mapping helps to SN to locate/acquire an AV that is needed for the authentication.
Subsequent Authentications

During all subsequent communications and the corresponding mutual authentications involved therein, the UE may present its identity in two different ways. Both these ways are listed below in their order of preference:

(i) **By transmitting a TMSI received in the previous AKA:**

In this method, the UE transmits the most recently received TMSI (i.e., the TMSI stored in $TMSI_{UE}$). When the SN receives a TMSI, it locates the corresponding $DMSI_{SN}$ using the $TMSI$-to-$DMSI_{SN}$ mapping maintained in its database. It then initiates the authentication procedure using an unused AV stored against $DMSI_{SN}$ in its local database (if there is any). If there is no unused AV, the SN will have to acquire a fresh set of AV (i.e., $AV[1..M]$) from the HN by presenting its $DMSI_{SN}$.

Out of the two methods, this is the preferred choice for the UE, because, it reduces communication latency during authentication. Specifically, whenever there is an unused AV at the SN, the authentication happens locally between the UE and the SN, without needing the SN to communicate with the HN. The protocol flow for subsequent authentications through the transmission of TMSI is as follows:

(2.1) The UE extracts $TMSI_{UE}$ from its memory (USIM's flash memory) and transmits it to the SN.

(2.2) Through this TMSI, the SN identifies the corresponding $DMSI_{SN}$ and hence the authentication vectors (i.e., $AV[1..M]$) that are stored against $DMSI_{SN}$. If there is no unused AV in $AV[1..M]$, SN sends a request for a fresh set of AVs along with the $DMSI_{SN}$ to the HN. In case there is an unused AV in $AV[1..M]$, the next step (i.e., step 2.3) is skipped.

(2.3) After receiving the request, the HN separates the RIC part of $DMSI_{SN}$. The IMSI-Pointer against this RIC leads to the record in the HN's database that contain details related with the corresponding IMSI of the UE. The remaining portion of this step proceeds in the same
manner as in step 1.3.

(2.4) The remaining part of the protocol flow is same as steps 1.4 through 1.6.

(ii) By transmitting a fresh DMSI:

In this method, the UE transmits a fresh DMSI that is created using the RIC extracted (using $f_{Extract}$) from the most recent ERAND received by the UE. This method of identity presentation is performed only during the following situations:

- **The SN cannot identify the UE with its current TMSI**: This may happen if the TMSI-to-DMSI$_{SN}$ mapping is lost from the SN’s database.

- **The subscriber moves from an old SN (say SN$_o$) to the service area of a new SN (say SN$_n$)**: In UMTS-AKA, the first identity presentation under the service area of SN$_n$ happens through transmission of a TMSI (say TMSI$_o$) allotted to the UE by SN$_o$ and the Location Area Identity (LAI) of SN$_o$. Unlike UMTS-AKA, in UMTS-AKA-with-E2EUIC, the first identity presentation under the service area of SN$_n$ happens through transmission of a fresh DMSI. This, makes the following two messages of UMTS-AKA redundant:
  
  (a) transmission of TMSI$_o$ from SN$_n$ to SN$_o$.
  
  (b) transmission of IMSI and TMSI$_o$ from SN$_o$ to SN$_n$.

The above two messages enables SN$_n$ to learn the IMSI of the subscriber from SN$_o$. SN$_n$ uses the received IMSI to collect AVs from the HN. In UMTS-AKA-with-E2EUIC, SN$_n$ can directly collect AVs from the HN (without communicating SN$_o$) by presenting the received DMSI to the HN, thereby improving communication latency.

- **Time To Live for DMSI (TTL$_{DMSI}$) has expired**: The SN uses the same DMSI stored in DMSI$_{SN}$ to uniquely identify the subscriber as long the subscriber continues to stay within the SN’s service area or till the SN does not receive another fresh DMSI from the UE, which ever happens earlier. However, this will allow a SN with malicious in-
Chapter 3. Relaxing Trust Requirement in UMTS

tention to link two or more connections of the same subscriber through the value of $DMSI_{SN}$, though it will not be possible to exactly know which particular subscriber it is. Thus, in order to prevent the SN from linking several communications of the same subscriber, it becomes important to limit the lifetime of the $DMSI$ stored in $DMSI_{SN}$. For this purpose, a field called Time To Live for DMSI ($TTL_{DMSI}$) is introduced in the USIM’s flash memory. Maximum value of $TTL_{DMSI}$ is to be decided by the operator (operator specific). Immediately after transmitting a freshly generated $DMSI$, the UE resets $TTL_{DMSI}$ to its maximum value. The value of $TTL_{DMSI}$ is decremented by one with the tick of every second. Next time whenever the UE has to transmit its identity, firstly the value of $TTL_{DMSI}$ is checked. If $TTL_{DMSI}$ is found to be greater than zero, the UE transmits a $TMSI$. Otherwise, if $TTL_{DMSI}$ is found to be equal to zero, a fresh $DMSI$ is computed and transmitted. This forces the SN to periodically refresh $DMSI_{SN}$ even if the UE chooses to stay with the same SN for a long duration.

The E2EUIC protocol flow for subsequent authentications through the transmission of a $DMSI$ is same as that of the first mutual authentication in UMTS-AKA-with-E2EUIC (i.e., steps 1.1 through 1.6). The only difference is that the $RIC$ contained in the most recently received $ERAND$ is used in this case by the UE to create a $DMSI$, rather than $RIC_{first}$.

3.6.2 Strengths

Some of the strong points of UMTS-AKA-with-E2EUIC are as follows:

- End to end user identity privacy: Knowledge of $IMSI$ is confined only to the UE and the HN; it is never transmitted at any stage of the protocol flow and at any portion of the path between the UE and the HN.

- Relaxed trust requirement: UE-to-SN as well as HN-to-SN trust relationship requirement with respect to permanent identity is relaxed. Such trust
relaxation simplifies roaming agreements between operators.

- Reduced number of message exchanges: If the extension is adopted, two protocol messages of UMTS-AKA becomes redundant. This will improve communication latency during authentication and key agreement.

- Fast database access: The RIC-Index makes searching through the HN's database faster.

- Minimal impact on the SN: For the extension to be adopted, most of the modifications are performed at the USIM and at the HN, very negligible amount of adjustment, of that of treating a received DMSI as an IMSI, is required at the SN. This, makes the extension easier to adopt for the operators, since an operator has to do the necessary modifications only at the HN's database and at the USIMs of the subscribers.

3.7 Example Algorithms for $f_{\text{Embed}}$ and $f_{\text{Extract}}$

In this section, we present an example algorithm to implement the cryptographic functions $f_{\text{Embed}}$ and $f_{\text{Extract}}$, which otherwise is operator specific [42]. If found appropriate, an operator may choose to use this algorithm, otherwise it may have its own implementation. $f_{\text{Embed}}$ embeds a 32 bit RIC into a 128 bit RAND using the secret key $K_i$ to produce a 128 bit ERAND (Equation 3.6.23), where as $f_{\text{Extract}}$ extracts the embedded RIC from an ERAND using $K_i$ as parameter (Equation 3.6.3).

The first step of the algorithm is to generate the following two sets of 32 element integer arrays from the secret key $K_i$:

$$A_{\text{pos}} = \{X_1, X_2, \ldots, X_{32}\} \quad (3.7.1)$$

$$A_{\text{XOR}} = \{Y_1, Y_2, \ldots, Y_{32}\} \quad (3.7.2)$$

such that

$$X_i \neq X_j; X_i \neq Y_j; Y_i \neq Y_j$$
Chapter 3. Relaxing Trust Requirement in UMTS

Figure 3.8: Generation of $A_{pos}$ and $A_{XOR}$.

where

\[ i, j \in \{1, 2, 3, ..., 32\} \]

and

\[ X_i, X_j, Y_i, Y_j \in \{1, 2, ..., 128\} \]

The elements of $A_{XOR}$ are used to locate 32 unique bit positions in the 128 bit $RAND$, the bit values in these positions are used to mask the 32 bits of $RIC$. The masking is achieved by performing bit-wise XOR operation between the 32 bits determined by $A_{XOR}$ and the 32 bits of $RIC$. The elements of $A_{pos}$ determine 32 other unique bit positions of $RAND$ that will be replaced by the masked bits of $RIC$. Finally, the resultant 128 bit number generated after inserting the masked bits is encrypted using AES cipher to find $ERAND$. $A_{pos}$ and $A_{XOR}$ are generated as follows:

To generate $A_{pos}$, the bits of $K_i(k_1, k_2, \ldots k_{128})$ are grouped into collection of seven, starting from left to right.

\[
(k_1 k_2 k_3 k_4 k_5 k_6 k_7), (k_8 k_9 k_{10} k_{11} k_{12} k_{13} k_{14})\ldots\quad (3.7.3)
\]

Each group yields a decimal number in the range of 1 to 128 (Figure 3.8). 7 bit groupings are used since $2^7$ provide 128 possibilities, corresponding to the 128 bit positions of RAND. At the end of the first scan, 18 such groups will be formed $((k_1 k_2 k_3 k_4 k_5 k_6 k_7), (k_8 k_9 k_{10} k_{11} k_{12} k_{13} k_{14})$, \ldots $k_{120} k_{121} k_{122} k_{123} k_{124} k_{125} k_{126})$ leaving the two least significant bits $k_{127}$ and $k_{128}$ ungrouped. Thus, there will be
18 decimal numbers in the range of 1 to 128 after the first scan, which may or may not be unique. Only the unique numbers are used to populate $A_{pos}$. In the next scan, the left out two least significant bits are grouped with the five most significant bits of $K_{i}$ to form the next group of seven, successive groups are created starting from the 6th most significant bit.

\[ (k_{127}k_{128}k_{1}k_{2}k_{3}k_{4}k_{5}), (k_{6}k_{7}k_{8}k_{9}k_{10}k_{11}k_{12}), \ldots \]  

This scanning process is continued in cyclic manner, till 32 unique decimal numbers are available to populate $A_{pos}$.

To generate $A_{XOR}$, a similar process is followed, except that the scan in this case proceeds from right to left. Every time an integer is generated, it is checked with the elements of $A_{pos}$, as well as the filled in elements of $A_{XOR}$ for uniqueness; only the unique ones being selected to populate $A_{XOR}$.

Since the key $K_{i}$ is a long time shared key, $A_{pos}$ and $A_{XOR}$ needs to be generated only once for every UE. If a key $K_{i}$ cannot be used to generate 64 unique integers by the above process, we refer it to be an unusable key. An unusable key should not be allocated to an E2EUIC enabled USIM.

In the next step, the XOR operations are performed (Figure 3.9). Let us denote a particular 32 bit $RIC$ and a particular 128 bit $RAND$ as follows:

\[
RIC_{i} = (p_{1}, p_{2}, \ldots, p_{32}) \quad (3.7.5)
\]

\[
RAND_{i} = (q_{1}, q_{2}, \ldots, q_{128}) \quad (3.7.6)
\]
Chapter 3. Relaxing Trust Requirement in UMTS

The decimal number stored against $X_1$ (in $A_{pos}$) determines the bit position in $RAND_t$ where $p_1$ is to be inserted. And, the decimal number stored against $Y_1$ (in $A_{XOR}$) determines the bit value in $RAND_t$ that has to be XORed with $p_1$ before it is inserted. For example, if $X_1 = 20$, $Y_1 = 96$ and $p_1 = 1$ then a XOR operation is performed between $p_1$ and the 96th bit of $RAND_t$. The resultant bit is then inserted into the 20th bit of $RAND_t$.

$$q_{X_1} = q_{Y_1} \oplus p_1 \quad (3.7.7)$$

where $\oplus$ is bit wise XOR operation. Thus, in order to insert all the 32 bits of $RIC_s$ into $RAND_t$, the following is to be performed:

$$q_{X_i} = q_{Y_i} \oplus p_i \quad (3.7.8)$$

where $i=1,2,3,...32$. The resultant 128 bit sequence after inserting all the 32 bits of $RIC_s$ into $RAND_t$ is called a Transformed $RAND$ (say $TRANS_D_t$ in this case). $TRANS_D_t$ may be represented as follows:

$$TRANS_D_t = (u_1, u_2, .., u_{128}) \quad (3.7.9)$$

Finally, an $ERAND$ is produced by encrypting $TRANS_D_t$ using the block cipher AES ($f_{AES}$) with the secret key $K_i$ as parameter.

$$ERAND_t = f_{AES_{K_i}}(TRANS_D_t) \quad (3.7.10)$$

At the UE's end, $RIC_s$ may be extracted back from $ERAND_t$ using a similar process. First of all, $TRANS_D_t$ is extracted from $ERAND_t$ using $f_{AES}$.

$$TRANS_D_t = f_{AES_{K_i}}(ERAND_t) \quad (3.7.11)$$

Since, we have the property that

$$A = B \oplus C \implies C = A \oplus B \quad (3.7.12)$$

thus, each bit of $RIC_s$ can be extracted from $TRANS_D_t$ as follows:

$$p_i = u_{Y_i} \oplus u_{X_i} \quad (3.7.13)$$

where $i=1,2,3,...32$. 

49
Chapter 3. Relaxing Trust Requirement in UMTS

Table 3.2: Percentage of unusable keys.

<table>
<thead>
<tr>
<th>Iterations</th>
<th>Unusable keys</th>
<th>Time (ms)</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>96</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
<td>376</td>
<td>0.100</td>
</tr>
<tr>
<td>10000</td>
<td>15</td>
<td>1282</td>
<td>0.150</td>
</tr>
<tr>
<td>100000</td>
<td>129</td>
<td>9476</td>
<td>0.129</td>
</tr>
<tr>
<td>1000000</td>
<td>1465</td>
<td>90016</td>
<td>0.146</td>
</tr>
</tbody>
</table>

A single RIC (i.e., RIC_{Fresh}) is embedded into the same bit positions (determined by A_{Pos}) of all the RANDs in AV[1..M]. The XOR operations are performed in order to mask the bit values in these positions, and to ensure that no two random numbers of AV[1..M] have any predictable pattern in them. The final encryption through AES is carried out to further shuffle all the 128 bits so that it becomes extremely hard for an adversary to predict a RIC that is embedded into an ERAND.

3.7.1 Usability of a Key

If 64 unique decimal numbers in the range of 1 to 128 can be extracted from a key $K_i$, we call this key an usable key for the example algorithm proposed in this section. Otherwise, we call it an unusable key. Thus, in order to use the example algorithm proposed in this section, the usability of a key $K_i$ should be properly verified before assigning it to any USIM.

We wrote a program in Java to test the usability of a series of 128 bit random numbers (keys), generated using SecureRandom class of java. Every single iteration in the program was made to generate a 128 bit random number and to check its usability, on the basis of whether 64 unique decimal numbers in the range 1 to 128 can be generated. We executed the program for varied number of iterations and recorded the number of unusable keys vis-a-vis the total number.
of keys generated, and the amount of time taken for each run. Our findings are listed in Table 3.2.

As evident from the results, the number of unusable keys generated is extremely small compared to the total number of keys. In our experiments, the percentage of unusable key vis-a-vis total number of keys never exceeded 0.15%. It also illustrates that the time consumed for such evaluation of keys is not very large. Moreover, in practice the evaluation of the usability of a key will not be done in real time.

### 3.7.2 Test for Randomness

In UMTS-AKA-with-E2EUIC, `ERAND` plays a pivotal role. Thus, randomness of `ERAND` is a vital issue. While randomness of `RAND` and `RIC` depends on operator specific random number generators and the call pattern of all the registered users respectively, the randomness of `ERAND` that depends on the algorithm used to implement $f_{Embed}f_{Extract}$, needs to be verified. In this subsection, we use a statistical test suite called the Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications provided by National Institute of Standards and Technology (NIST), to analyse the randomness of `ERANDs` that are generated using the example algorithm proposed in this section.

The NIST Test Suite is a statistical package that provides a list of tests to test the randomness of random numbers produced by cryptographic random or pseudo-random number generators. These tests focus on a variety non-randomness that could exist in a number. For our analysis, we have used seven of these tests (selected arbitrarily), namely, the frequency (mono-bit) test, frequency test within a block, the runs test, tests for the longest-run-of-ones in a block, the linear complexity test, the approximate entropy test and the cumulative sums (cu-sums) test. Details about all these tests are presented in [43].

As stated in NIST's Special Publication 800-22 [43], a statistical test is formulated to test a specific null hypothesis ($H_0$). Here, the null hypothesis is that a random number being tested is random. Each test is based on a calculated test
statistic value, which is a function of the random number being tested. The test statistic is used to calculate a p-value. If the p-value for a test is equal to 1, then a random number is considered to have perfect randomness. A p-value of zero indicates that a random number is completely non-random. A significance level (\( \alpha \)) is chosen for the tests. If p-value \( \geq \alpha \), then the null hypothesis is accepted; i.e., the random number being tested is considered to be random. If p-value < \( \alpha \), then the null hypothesis is rejected; i.e., the random number being tested is considered to be non-random. Typically, \( \alpha \) is chosen in the range [0.001, 0.01].

For statistical analysis of ERAND, we use the strategy adopted by NIST that consists of five stages [43]:

1. **Selection of a Generator:**
   We have used Sun's SHA1 Pseudo Random Number Generator (PRNG) - 'SecureRandom.class', available in 'java.security' package to generate the random numbers used in the analysis.

2. **Binary Sequence Generation:**
   We wrote a program in Java called 'GeneratePRN.java' that generates 1000 random numbers (128-bit) and stores them in a file called 'RAND.txt'. Each of the random numbers in this file represents a RAND. We wrote another program called 'EmbedRAND.java' that generates a fresh 32 bit random number (to represent RIC) and embeds it into each 128 bit random number of 'RAND.txt' (using our proposed example algorithm for \( f_{Embed} / f_{Extract} \)). The resultant 1000 ERANDs (128-bit) are written into another file called 'ERAND.txt'. Each of the random numbers in 'ERAND.txt' represents an ERAND.

3. **Execute the Statistical Test Suite:**
   Using the test suite, each of the selected seven tests were performed on the random numbers written in the file 'ERAND.txt' (by passing 'ERAND.txt' as parameter to the test suite).

4. **Empirical Results:**
   Each statistical test generates empirical results that consists of test statistics
and p-values against each of the random numbers stored in 'ERAND.txt'. An output file is generated by the test suite with the empirical results for each of the statistical tests written in it. Based on these results, a conclusion regarding quality of the sequence of random numbers generated by the proposed example algorithm can be made.

5. Interpretation of Empirical Results:
NIST has adopted two approaches for interpretation of empirical results. In the event that either of these approaches fail, the corresponding null hypothesis must be rejected. Here, the null hypothesis ($H_0$) is that the sequence of random numbers (stored in the file 'ERAND.txt') being tested is random. In the following portion of this subsection, we carry out both these approaches to analyse randomness of the sequence of random numbers stored in 'ERAND.txt'.

(i) Proportion of Random Numbers Passing a Test: The proportion of random numbers passing a statistical test is the ratio between the number of random numbers whose $p$-values $\geq \alpha$ and the total number of random numbers (say $t$) present in the sequence of random numbers being tested. For example, if a statistical test tests 1000 random numbers (i.e., $t = 1000$), with the significance level $\alpha = 0.01$, and with 996 random numbers having $P$-values $\geq 0.01$, then the proportion of random numbers passing the test is $\frac{996}{1000} = 0.9960$. The proportion of random numbers passing a statistical test for all the seven selected tests performed on 'ERAND.txt', considering $\alpha = 0.01$, are calculated and is listed in Table 3.3.

For a fixed significance level ($\alpha$), a certain proportion of $P$-values generated by a particular test are expected to fail. For example, if the significance level is chosen to be 0.01 (i.e., $\alpha = 0.01$), then about 1% of the random numbers are expected to fail. Taking this into consideration, NIST has determined the following range of acceptable proportions, in
Table 3.3: Proportion of random numbers that pass a test.

<table>
<thead>
<tr>
<th>Statistical Test</th>
<th>P-values (\geq \alpha)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Frequency (Mono-bit) Test</td>
<td>995</td>
</tr>
<tr>
<td>2. Frequency Test within a Block</td>
<td>996</td>
</tr>
<tr>
<td>3. Runs Test</td>
<td>994</td>
</tr>
<tr>
<td>4. Tests for the Longest-Run-of-Ones</td>
<td>988</td>
</tr>
<tr>
<td>5. Linear Complexity Test</td>
<td>981</td>
</tr>
<tr>
<td>6. Approximate Entropy Test</td>
<td>995</td>
</tr>
<tr>
<td>7. Cumulative Sums (Cu-sums) Test</td>
<td>998</td>
</tr>
</tbody>
</table>

its Special Publication 800-22 [43]:

\[
A = \hat{p} \pm \sqrt{\frac{\hat{p}(1-\hat{p})}{t}}
\]

(3.7.14)

where, \(\hat{p} = 1 - \alpha\) and \(t\) is the sample size. If the proportion of random numbers that pass a test falls outside of this interval, then there is evidence that the sequence of random numbers being tested is non-random.

For analysis of the sequence of random numbers stored in ‘ERAND.txt’, if \(\alpha = 0.01\) and \(t = 1000\), the range of acceptable proportion is:

\[
A = 0.99 \pm \sqrt{\frac{0.99(0.01)}{1000}}
\]

(3.7.15)

\[
= 0.99 \pm 0.0094392
\]

Fig. 3.10 provides a graphical representation of the proportion of successful random numbers against each statistical test performed on the random numbers stored in ‘ERAND.txt’. Since the proportions for all the tests lie within the range of acceptable proportions (i.e., between 0.9805608 and 0.9994392), the sequence of random numbers present in the file ‘ERAND.txt’ can be considered to be random.

(ii) **Uniform Distribution of P-values**: The distribution of *P-values* is examined to ensure uniformity. In this method (proposed by NIST in [43]), the interval between 0 and 1 is divided into 10 sub-intervals (C1, C2, ..., C10), and the *P-values* obtained by performing a particular statistical test on a sequence of random numbers, which lie within each
Proportion of Random Numbers Passing a Test

1. The Frequency (Monobit) Test
2. Frequency Test within a Block (Block Size 16 bits)
3. The Runs Test
4. Tests for the Longest-Run-of Ones in a Block
5. The Linear Complexity Test
6. The Approximate Entropy Test
7. The Cumulative Sums (Cusums) Test

Figure 3.10: P-value plot

Sub-interval are counted. Uniformity is then determined via an application of a $\chi^2$ test and the determination of a $P$-value corresponding to the Goodness-of-Fit Distributional Test on these $P$-values (i.e., a $P$-value of the $P$-values). This is accomplished by computing:

$$\chi^2 = \sum_{i=1}^{10} \frac{(F_i - \frac{t}{10})^2}{\frac{t}{10}}$$  \hfill (3.7.16)

where $F_i$ is the number of $P$-values in sub-interval $i$, and $t$ is the sample size. A $P$-value is calculated such that:

$$P\text{-value}_T = \text{igamc} \left( \frac{9}{2}, \frac{\chi^2}{2} \right)$$  \hfill (3.7.17)

where igamc is the complementary incomplete gamma function. If $p\text{-value}_T \geq 0.0001$, then the sequence of random numbers can be considered to be uniformly distributed. The value of $\chi^2$ and $P\text{-value}_T$ (calculated from the empirical results) for each of the tests performed on ‘ERAND txt’ is summarised in Table 3.4. Since the value of $P\text{-value}_T$ for all the tests
Table 3.4: $P-value_T$ for the statistical tests.

<table>
<thead>
<tr>
<th>Statistical Test</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
<th>C6</th>
<th>C7</th>
<th>C8</th>
<th>C9</th>
<th>C10</th>
<th>$x^2$</th>
<th>$P-value_T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency Test</td>
<td>134</td>
<td>93</td>
<td>86</td>
<td>105</td>
<td>107</td>
<td>100</td>
<td>96</td>
<td>103</td>
<td>100</td>
<td>76</td>
<td>20.76</td>
<td>0.013760</td>
</tr>
<tr>
<td>Frequency Test within a Block</td>
<td>93</td>
<td>114</td>
<td>129</td>
<td>93</td>
<td>89</td>
<td>98</td>
<td>110</td>
<td>101</td>
<td>93</td>
<td>80</td>
<td>18.10</td>
<td>0.034010</td>
</tr>
<tr>
<td>Runs Test</td>
<td>105</td>
<td>106</td>
<td>125</td>
<td>110</td>
<td>92</td>
<td>94</td>
<td>64</td>
<td>121</td>
<td>105</td>
<td>78</td>
<td>31.32</td>
<td>0.000261</td>
</tr>
<tr>
<td>Test for Longest Run of Ones</td>
<td>97</td>
<td>112</td>
<td>88</td>
<td>90</td>
<td>100</td>
<td>120</td>
<td>78</td>
<td>131</td>
<td>86</td>
<td>98</td>
<td>24.42</td>
<td>0.003684</td>
</tr>
<tr>
<td>Linear Complexity Test</td>
<td>107</td>
<td>84</td>
<td>85</td>
<td>93</td>
<td>77</td>
<td>91</td>
<td>109</td>
<td>105</td>
<td>130</td>
<td>119</td>
<td>25.56</td>
<td>0.002410</td>
</tr>
<tr>
<td>Approximate Entropy Test</td>
<td>113</td>
<td>103</td>
<td>79</td>
<td>103</td>
<td>100</td>
<td>123</td>
<td>86</td>
<td>106</td>
<td>86</td>
<td>101</td>
<td>15.86</td>
<td>0.069863</td>
</tr>
<tr>
<td>Cumulative Sums Test</td>
<td>80</td>
<td>102</td>
<td>94</td>
<td>89</td>
<td>138</td>
<td>88</td>
<td>81</td>
<td>98</td>
<td>128</td>
<td>102</td>
<td>33.02</td>
<td>0.000132</td>
</tr>
</tbody>
</table>
are greater than 0.0001, we consider the sequence of random numbers stored in 'ERAND.txt' to be uniformly distributed.

Since the sequence of random numbers stored in 'ERAND.txt' passes both the approaches specified by NIST for interpretation of empirical results, we can accept the null hypothesis and may conclude that the example algorithm proposed for generation of \textit{ERANDs} produces a sequence of \textit{ERANDs} that are sufficiently random for cryptographic use.

### 3.8 Summary

In this chapter, the AKA protocol used for access security in UMTS (i.e., UMTS-AKA) is analysed. It is found that due to the existing trust model used for roaming in UMTS, there are vulnerabilities in UMTS-AKA because of which the identity privacy of the subscriber gets compromised. To improve this situation, a security extension called E2EUIC with respect to UMTS-AKA is proposed. The extension is based on our trust model that is proposed in Chapter 2. This is a novel solution that prevents transmission of \textit{IMSI} across the entire network (wired as well as wireless); without necessitating any change in the intermediate network. Unlike many solutions which look at restricting \textit{IMSI} transmission only over radio link, E2EUIC takes a comprehensive end-to-end view of the problem. A couple of example algorithms needed for implementation of E2EUIC, which otherwise are operator specific, were also proposed. The usability of a series of randomly generated keys in these example algorithms were then analysed using programs written in Java. In addition, the randomness of random numbers that are generated by the example algorithms were verified using a statistical test suite provided by NIST.