The 5G Key-Establishment Stack: In-Depth Formal Verification and Experimentation

Rhys Miller
rhys.miller@surrey.ac.uk
Surrey Centre for Cyber Security
University of Surrey
Guildford, UK

Ioana Boureanu
i.boureanu@surrey.ac.uk
Surrey Centre for Cyber Security
University of Surrey
Guildford, UK

Stephan Wesemeyer
s.wesemeyer@surrey.ac.uk
Surrey Centre for Cyber Security
University of Surrey
Guildford, UK

Christopher J P Newton
c.newton@surrey.ac.uk
Surrey Centre for Cyber Security
University of Surrey
Guildford, UK

ABSTRACT

We formally analyse the security of each 5G authenticated key-establisment (AKE) procedures: the *5G registration*, the *5G authentication and key agreement (AKA)* and *5G handovers*. We also study the security of their composition, which we call the *5GAKE_stack*. Our security analysis focuses on aspects of multi-party AKEs that occur in the *5GAKE_stack*. We also look at the consequences this AKE (in)security has over critical mobile-networks' objects such as the Protocol Data Unit (PDU) sessions, which are used to bill subscribers and ensure quality of service as per their contracts/plans.

In our assessments, we augment the standard Dolev-Yao model with different levels of trust and threat by considering honest, honest-but-curious, as well as completely rogue radio nodes. We formally prove security in the first case, and insecurity in the latter two as well as making formal recommendations on this. We carry out our formal analysis using the Tamarin-Prover.

Lastly, we also present an emulator of the *5GAKE_stack*. This can be a useful "5G API"-like tool for the community to experiment with the *5GAKE_stack*, since the 5G networks are not yet fully deployed and mobile networks are proprietary and closed "loops".

CCS CONCEPTS

• **Security and privacy → Formal security models**.

KEYWORDS

Formal Verification, Security, Mobile Communications

ACM Reference Format:
Rhys Miller, Ioana Boureanu, Stephan Wesemeyer, and Christopher J P Newton. 2022. The 5G Key-Establishment Stack: In-Depth Formal Verification and Experimentation. In *Proceedings of the 2022 ACM Asia Conference on Computer and Communications Security (ASIA CCS '22), May 30–June 3, 2022, Nagasaki, Japan.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3488932.3517421

ASIA CCS '22, May 30–June 3, 2022, Nagasaki, Japan.

ACM ISBN 978-1-4503-9140-5/22/05…$15.00
https://doi.org/10.1145/3488932.3517421

1 INTRODUCTION

In 2021, 5th Generation Mobile Networks (5G) technologies are at a vital point in their development. In the UK for instance, in 2018, a government-focused consultation [27] forecast full-country 5G-coverage by 2027; also, they estimated that the UK would invest around £6.85 billion in the role-out of digital infrastructure, including 5G-deployment, between 2018 and 2021. Also, 5G – as opposed to 4th Generation Mobile Networks (4G) – is able to offer many more and much refined services, as well as subscribers' plans of numerous sorts (e.g., Netflix bandwidth included, content-delivery acceleration for video streaming, etc.). With this comes an increased interest in illicit gains: e.g., to maliciously obtain a better Quality of Service (QoS) without paying for it. Moreover, the radio nodes in 5G are usually accessed and manipulated via software remotely [26], much more so than in 4G where this was already shown to be a risk [29], thus increasing the attack surface of the mobile network considerably.

Given these added threat vectors, it is of paramount importance to offer new as well as varied security analyses for the different cryptographic protocols[1] used in the 5G network. In this work, we do this by focussing on the 5G security procedures that re-establish the authentication and channel-securing keys for the mobile device thus ensuring it can be cryptographically identified to the different parts of the network and communicate securely with them.

Our Work on 5G AKEs

The Stack of Inter-dependent AKE Protocols in 5G. The AKE protocols in 5G are: the *Registration procedure (REG)* and *AKA*, the *XN handover procedure (XN)*, and the *N2 handover procedure (N2)* [11]. REG (incl. AKA) and handovers (XN, N2) operate over the same user-identifiers and largely the same cryptographic material. Yet, these procedures are executed by different entities in the 5G network, with some more trustworthy/secure than others. Moreover, the key-updates carried out by each of these procedures affect different levels of the same 5G key-hierarchy, which is shared by all

[1]The 3rd Generation Partnership Project (3GPP) specifications [1–5, 7–9, 11] refer to protocols as "procedures".

these procedures and which is the basis for the user's authentication and secure communication. Finally, the REG, AKA, XN and N2, all carry back and forth long-term data linked to the user's voice/data connections and billing information. So, it is important that these procedures be as secure as they can be, be considered in silo, but also as a composed AKE primitive, and under varying trust and threat models.

When we consider these 5G AKE procedures (i.e., REG, AKA, XN and N2) all together, we refer to the composed procedures as the 5GAKE_stack.

In-depth Security Analysis of the 5G AKE Protocols, in Varying Threat Models. In this paper, we formally analyse the procedures in the *5GAKE_stack*, in a stand-alone manner and as a composed AKE protocol, applying various threat models. We find several security issues in the case where the radio-part of the network (a.k.a. the Radio Access Network (RAN)) contains rogue nodes; in some cases, these nodes need not even be fully malicious; "honest-but-curious" is enough to lower the security of these procedures.

Towards an API for the 5GAKE_stack. On a different note, the arguably numerous and non-trivial 5G specifications have evolved significantly since 2016 when their 3GPP standardisation process began. In these specifications, the fully-capable 5G mode is called "stand-alone", but this is not yet widely deployed. There are not many devices on the market that are activated in stand-alone mode. At the moment, it is therefore hard to see the full 5G security procedures "at work". To this end, another contribution in this paper is an emulator (and its source code) of these 5G security procedures (e.g., REG, AKA, handovers) such that other researchers can use this as they would an Application Programming Interface (API) for 5G experimentation.

Organisation of the paper. Section 2 presents a background on the 5G Registration, and the 5G handover procedures. Section 3 shows our envisaged threat models. In Section 4, our formal security verification of the full *5GAKE_stack* and of the sub-procedures of the *5GAKE_stack* is discussed. Our *5GAKE_stack* emulator is discussed within Section 5. Lastly, in Section 6, we discuss related work and in Section 7, we conclude.

2 BACKGROUND

In Section 2.1, we present an overview of the 5G network[2]. In Section 2.2, we give an outline of the *5GAKE_stack*, focusing on the aspects needed for our analyses and implementation.

5G Specifications. The 3GPP specifications used to create our formal models follow Release 16 [6], and are: TS 23.316 [9], TS 23.501 [7], TS 23.502 [11], TS 24.501 [4], TS 29.244 [3], TS 33.501 [8], TS 33.512 [1], TS 33.515 [2], TS 38.415 [5], and TS 38.423 [10].

2.1 An Overview of the 5G Network

In Figure 1, we give a simplified representation of the 5G network architecture.

Users. In 5G networks, a *User Equipment (UE)* is subscribed to and receives service from an *operator*, whose main backend infrastructure is called the *core network*. The subscribers and the core will share several long-term cryptographic secrets, which we denote succinctly as K.

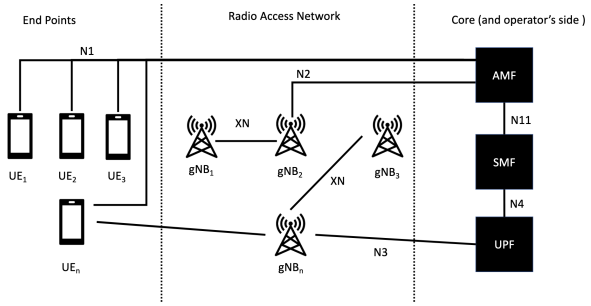

Figure 1: Main 5G Entities & Interfaces: An Overview

Radio Access Nodes. At any point, a user is provided mobile service through a radio "base-station" denoted *Next Generation NodeB (gNB)*, which in turn can communicate with the UE's core network and other gNBs. The combination of these nodes forms the RAN.

The Operator & Its Core. We only mention briefly some sub-parts of the core. The Access and Mobility Management Function (AMF) receives all connection and session related information from the UE but is only responsible for handling connection and mobility management tasks while the Session Management Function (SMF) deals with session management.

On the side of the operator, there is also the implementation of the User Plane Function (UPF). A UPF is a packet gateway which connects the operator's internal network to an external network, e.g. the internet. So when a UE makes a HTTP request, its serving gNB forwards the request to the UE's UPF. The UPF then routes the UE-issued HTTP traffic in and out of the edge of the network. It also supports/implements different UE-related functions including rate-limiting based on QoS agreements and billing.

Communication Interfaces. Interfaces are tagged alphanumerically in Figure 1 (e.g., N1, N4, etc). At any point, a given UE is served by a single gNB. They can communicate over an un-encrypted channel, at the so-called *Non Access Stratum (NAS) level*: for instance, when the gNB proxies procedures between the UE and the core, e.g., to re-establish new cryptographic keys used to (re-)secure the channel between the gNB and the UE. Alternatively, the two can communicate securely at the *Access Stratum (AS) level* over which the actual mobile messages are sent/received. Analysing the security of re-establishing the keys to secure the AS level is one aspect of this work.

All interfaces between the gNBs themselves, between the gNBs and the *core* and between the different parts of the core (including UPFs) are secure, i.e., they are assumed to provide confidentiality, integrity and authentication. The security of these interfaces does not form part of this work. We are only concerned with the security of UE-to-gNB and UE-to-Core.

[2]We assume no roaming, i.e., the subscriber is served only by the network managed by their operator.

2.2 An Overview of the *5GAKE_stack*

2.2.1 Securing UE-to-gNB Interfaces via the 5GAKE_stack. Actual mobile/application messages, called *access-stratum messages*, are messages between any one UE and a gNB. The two entities (using the core) first establish a secure channel by using the well-known AKA as part of a procedure called *Registration (REG)*. Messages over this channel are secured with what is called *access-stratum (AS) keys* (K_{AS}). These K_{AS} are derived from a key called K_{gNB}, which is (re-)established during AKA, but also during handovers, as we explain next.

Occasionally, a gNB that has so far provided a user with mobile signal changes (for instance because the user has moved physically out of its reach). At this point, a *handover* is initiated which requires the refreshing of the key K_{gNB}. This key has been shared by the UE and one gNB, referred to as *source node (s-gNB)*, and the new K_{gNB} key will now be shared by the UE and a new serving node, called *target node (t-gNB)*. There are two handover procedures: XN, and N2. There are several flavours of N2.

As Figure 1 shows, not all gNBs are connected amongst themselves via an XN interface (XN), but all are connected via a N2 interface (N2) to the core. This implies that all gNBs can execute N2 for a handover, but not all can use XN.

With the refreshing of K_{gNB}, the different procedures in the *5GAKE_stack*, in fact, refresh different series of keys respectively. This key hierarchy is summarised in Figure 2, where different boxes show the procedures controlling their respective parts of the hierarchy. On the edges of Figure 2, one can see that these keys are shared by different entities (e.g., K_{gNB} and the AS Keys – UE and gNBs; the rest – UE and core). Indeed, these keys serve different purposes and are either used: a) as "seed" keys for other keys (e.g., NH); b) for authentication (e.g., K_{SEAF}); or c) for securing actual interfaces (e.g., K_{AMF} to secure $N1$).

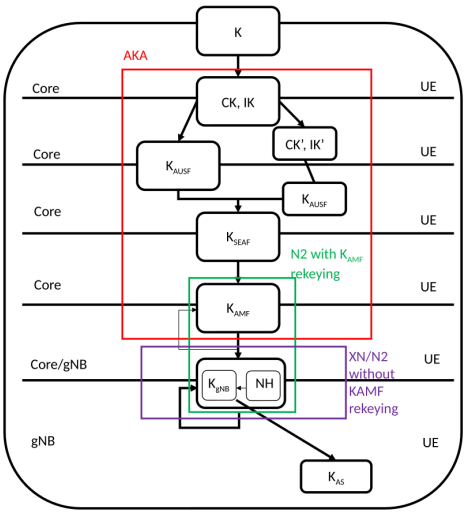

Figure 2: 5G Key Hierarchy [7, 11]: Refreshment by Procedure & Sharing across Parties

2.2.2 The Registration & AKA Procedures. The Registration procedure (REG), which contains AKA as a sub-procedure, is executed when a UE joins the network or (re)gains signal. The Registration procedure is described mainly in [7, 11].

The REG procedure is run between a UE and the *core*, and it is passively proxied by a node gNB_0. During REG , the *core* authenticates the UE, and –with the aim to secure the channel between said UE and gNB_0– the procedure refreshes a list of keys as shown in Figure 2. The lowest-level key regenerated by the UE and the *core* at the end of AKA is K_{AMF}. This K_{AMF} is used by the *core* and the UE to generate a new *"security key" denoted* K_{gNB}. The *core* sends K_{gNB} to gNB_0, and this radio-node uses K_{gNB} to derive the access-stratum keys K_{AS} to communicate securely with the UE.

In Table 1, we show the short-term keys re-established at the end of each REG execution, and their distribution across parties.

	UE	gNB_0	Core
K_{AMF}	✓		✓
K_{gNB}	✓	✓	✓

Table 1: Keys Refreshed/Shared at the End of Registration

2.2.3 The Handover Procedures: XN and N2. The handover procedures are described mainly in [7, 11]. A handover procedure is executed every time the UE swaps from being served by one gNB, called *source node s-gNB* to being served by another gNB, called *target node t-gNB*. This generally happens to enable the UE to maintain a good signal/connection, as it physically moves away from one radio node and closer to another.

Another aim of a handover is to facilitate secure communication between the UE and the new gNB, i.e., the t-gNB. In line with the REG procedure, a fresh "security key" K^*_{gNB} needs to be sent to the target node t-gNB as part of the handover which is then used to establish the AS keys. There are two different handover procedures supporting this:

(1) *The N2 procedure.* In this case, the *core* generates and sends a new security key K^*_{gNB} to the t-gNB. This is similar to the Registration procedure. The serving s-gNB asks the core to start the process, but the s-gNB is passive with respect to K_{gNB}'s regeneration.
(2) *The XN procedure [8, 10].* In this case, the serving s-gNB, which already has a current security-key K_{gNB} shared with the UE, is an active proxy in the key-exchange procedure: it generates and sends to the t-gNB a new security key K^*_{gNB}.

XN Key-Derivations & Backwards Security. In XN, the source node s-gNB has a current security key K_{gNB} and computes a new security key K^*_{gNB} for the target node t-gNB. Since the AS keys are computed out of the K_{gNB} keys and (retrievable/known data), it means that the source node s-gNB can compute the AS keys that will decrypt the channel between the UE and the t-gNB. This is a well-known fact [8]: i.e., XN does not have backward security [19]. XN's lack of backward security can or cannot be "healed" [19] depending on how K^*_{gNB} is derived. To this end, we now describe XN's computation of K^*_{gNB} which can be done either by *horizontal key derivation (HKD)* or *vertical key derivation (VKD)*.

• *XN's Horizontal Key Derivation.* In this case, the source node derives the new K^1_{gNB} from the previous K^0_{gNB}. We use XN^{hkd} to refer to XN executed with such an establishment of K^1_{gNB}.

	UE	s-gNB	t-gNB	Core
K_{gNB}	✓	✓		?
K^*_{gNB}	✓	✓	✓	
current NH		✓		✓
next NH			✓	✓

Table 2: Keys Refreshed/Shared at the End of XN with HKD

	UE	s-gNB	t-gNB	Core
K_{gNB}	✓	✓		?
K^*_{gNB}	✓	✓	✓	
current NH	✓	✓		✓
next NH			✓	✓

Table 3: Keys Refreshed/Shared at the End of XN with VKD

Clearly, XN's lack of backward security can be prolonged via a series of XN^{hkd} executions. Consider that XN^{hkd} was executed in n times in a row. So, the security key K^n_{gNB} for the n-th t-gNB depends on the K^{n-1}_{gNB} of the $n-1$-th s-gNB and other recoverable/public data. So, a rogue gNB which was source node n handovers ago can retrieve iteratively all the security keys $K^1_{gNB}, \ldots, K^n_{gNB}$ and therefore decrypt the UE-AS traffic with all the nodes that were t-gNB in these n handovers.

• *XN's Vertical Key Derivation.* In this case, the source node derives the new K^1_{gNB} using a unique key called *Next-Hop Key (NH)* which it received from the *core* at the end of the handover in which it acted as the t-gNB for the current, to-be-handed-over UE. This NH is derived by the *core* from K_{AMF}, which –as we explained– is refreshed (at least) at the end of each Registration procedure execution. We use XN^{vkd} to refer to XN executed with such an establishment of K^1_{gNB}.

Clearly, in a sequence of XN executions, as soon as a vertical key derivation takes place, the chain of serial loss of backward security is broken. Note that, provided we trust the gNBs to follow the protocol and not be malicious, vertical derivation should (always) take place when the s-gNB holds an unused NH[3] for the UE currently being handed over.

To summarise, the main keys which are re-established and/or shared across parties at the end of a XN handover procedure execution are given in Tables 2 and 3. In Tables 2 and 3, we have a placed a question mark in the K_{gNB} column for the *core*, as it is possible for the *core* to have this key in the case where the XN was preceded by a Registration procedure.

N2 Key-Derivations: Trust & Backwards Security. The N2 protocol requires less trust in the gNBs, since the s-gNB is not actively calculating the security key K^*_{gNB} for the t-gNB. In N2, the derivation of K^*_{gNB} is in fact a vertical key derivation based on a new NH locally computed by the *core* and sent to the t-gNB.

The *core* can be configured to re-fresh the keys further up the key hierarchy than NH: e.g., by recomputing K_{AMF}; this is referred to as *"N2 with K_{AMF} rekeying"*. When we write simply N2, we generally assume implicitly it is without K_{AMF} rekeying.

The choice between one type of handover or another is down to the interfaces between the s-gNB and the t-gNB: if there is an XN between them, then the XN handover procedure is executed; otherwise, N2 will be executed, intermediated by the *core*.

5GAKE_stack*: Summary on Key Derivations & Backwards Security.* Tables 1 to 3 show the end-state of executing the *5GAKE_stack* procedure with respect to key refreshment and redistribution. We provide further details in our appendices: e.g., see Figure 6 for REG and Figure 7 for XN.

Finally, Figure 3 gives a summative representation of the backward security and "post-compromise healing" [19] in REG, XN, and/or N2.

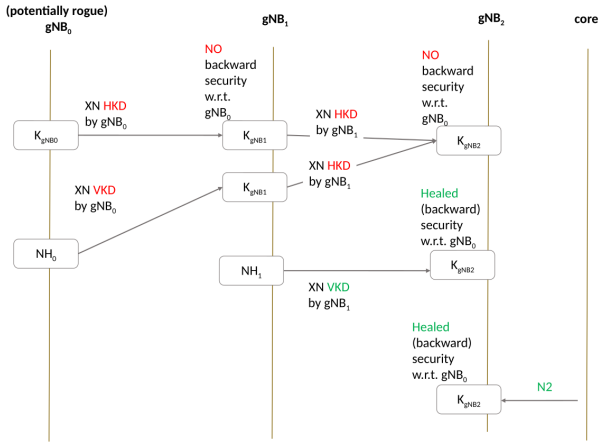

Figure 3: Overview of Backward Security with respect to K^*_{gNB}: Loss and Healing via 5G Handovers

2.2.4 Protocol Data Units Sessions. A Protocol Data Units (PDU) is a networking and billing-related data-structure used by the 5G network to book-keep the data connections and the associate usage that the UEs have. A PDU session will contain several long-term details, such as what a given user is entitled to by its contract (e.g., Quality of Service parameters), as well as real-life details such as consumption data, speed of connection and network information (e.g., the UPF used to route the connection out to the internet), all in relation to the current connection. In this work, we refer to the long-term details of a PDU session as the *"PDU session type"*. An active UE has what is called a *PDU session-list* (PDUSL) and several PDU sessions can be open at once, each with its own type.

In any procedure of the *5GAKE_stack*, the latest PDU session list of the UE is updated by the *core* e.g., the core might assign new UPFs[4] for the different PDU sessions, and will provide the user's plan details such as QoS descriptions as part of an updated PDU session list, *PDUSL'*.

The *core* sends the updated PDU session list PDUSL' to the gNB that will serve the UE as well the relevant UPFs. This PDUSL update and its distribution by the end of any *5GAKE_stack* procedure is also shown in Table 4.

[3]The NH could have been used by the current s-gNB, if this gNB received it at the end of N2, or if the served UE had temporarily been in idle state and then the gNB refreshed its own K_{gNB} using that NH.

[4]A 5G UE could have different PDU sessions served by different UPFs: e.g., the Netflix traffic goes via one UPF and other traffic via another.

	UE	the (new) serving gNB	Core (SMF)	UPF
PDU-SL'	✓	✓	✓	✓

Table 4: Updated PDU Session List Across Entities at the End of *5GAKE_stack* Procedures

UPFs' Smart Filtering on PDUs. Depending on the type of a UE's access-stratum data connection, the serving gNB will send the UE's data traffic to the corresponding UPF for routing out to the internet. The serving gNB should do this forwarding according to what is recorded in the current PDU session list PDUSL', i.e., send it to the associated UPF using the QoS and other service parameters as stated in the UE's current PDU session list. A UPF can be configured to check that any received request is in agreement with its PDUSL' sent by the *core* for the UE in question; see [3, §§5.7,5.8]. We call this *smart filtering* but also note that this is **not** a mandatory configuration in 5G. As such, the request may be served by the UPF without any checks and thus not match the UE's service parameters.

3 OUR TRUST & THREAT MODELS

We will now consider the different trust and threat models applied to the *5GAKE_stack*:

The Honest 5GAKE_stack. As shown in Figure 2, the Registration procedure (incl. AKA), the XN handover procedure, and the N2 handover procedure all refresh the same subset of 5G keys used directly to secure the channel between the UE and the gNBs (i.e., K^*_{gNB}, NH), as well as those keys (i.e., K_{AMF}) for securing the $N1$ interface between the UE and the AMF. In fact, these procedures are usually run sequentially, e.g., REG, XN, XN, XN, N2, XN, REG, and the lack of security in one can propagate and affect the security of the execution of another. This suggests that registration and handovers need to be analysed individually (especially those that have not been, e.g., the Registration procedure), as well as part of the full *5GAKE_stack*.

Honest-but-Curious gNBs. In the XN handover procedure, we need to trust that the s-gNB will improve the lack of "one-step" backwards security by choosing a Vertical Key Derivation (VKD) whenever it has an available NH for the UE to be handed over. If the s-gNB was set not to do so, this would not require any real software modification on the radio, i.e., the gNB could be forced to do Horizontal Key Derivation (HKD) only via a configuration option. This rogue gNB would be *"honest-but-curious"*, in that it would run the correct XN handover procedure but in a way that facilitates knowledge of its generated K^*_{gNB} for previous gNBs.

Corrupt/Rogue gNBs. Given a PDU session update, which happens during the Registration procedure, XN handover procedure and N2 handover procedure, we trust the serving gNBs to use the correct PDU session list for a given UE and route its data traffic accordingly. Contrary to this, a rogue gNB may choose the PDU session types itself, differently to what the core prescribed thus controlling a UE's level of service (irrespective of the *core*-updated PDU session list and the UE's contract).

In this paper, we formally check the *5GAKE_stack* procedures, separately and/or composed, given the honest, "honest-but-curious", and corrupt threat models for gNB.

CONTRIBUTIONS

The contributions of this paper are as follows: In our **first** contribution, detailed in Section 4, using the `Tamarin-prover` [30], we model and formally verify the following aspects:

(1) **W.r.t Honest Registration procedure.** We model the Registration procedure and formally prove its security in the case of honest participants.
(2) **W.r.t. Honest *5GAKE_stack*.** In a single model, we formalise the Registration procedure (incl. AKA) and XN handover procedure and formally prove the security of this version of the *5GAKE_stack*, if no party is dishonest. This formally shows that even if the two procedures have different and varying parties involved, from an AKE perspective, their composition runs correctly/securely from beginning to end, i.e., at the end of the Registration procedure, the parties finish in agreement over keys as per Table 1, and at the end of the XN handover procedure, the parties finish in agreement over keys as per Tables 2 and 3.
(3) **W.r.t. Corrupt gNBs.** We formalise the Registration procedure, adding several relevant sub-parts of the network such as the UPFs, as well as an AS message for the UE to make data requests. In this augmented REG model, we include malicious gNBs and formally show that:
– parties in the Registration procedure end up in agreement over the UE's updated PDU session list PDUSL', as expected and stated in Table 4.
– after this PDU agreement, a corrupt gNB can fraudulently manipulate the PDU sessions of any UE it serves, thus impacting their quality of service in any manner they wish. In practice, this means that the radio nodes can choose subscribers' QoS irrespective of what their contracts states.
– this malicious behaviour can be stopped if the UPF does "smart filtering" and thus checks the correctness of the PDU sessions it is routing onto for a given UE. We therefore recommend that smart filtering be mandatory for UPFs.
To our knowledge, this is the first time the 5G Registration procedure has been modelled formally; this is true for both a) considering it in the normal Dolev-Yao model, and b) in the case of corrupt gNBs.
(4) **W.r.t. Honest-but-Curious gNBs.** We model the XN handovers including honest-but-curious gNBs which can force horizontal key-derivations. We formally show that one such honest-but-curious gNB can always hand over to another honest-but-curious node, thus making the lack of backwards security last longer while honest gNBs present in the model jsut follow the procedure correctly. Together, this shows that such honest-but-curious gNBs can exist in the network and operate in the *5GAKE_stack*, unhindered.
(5) **Creating, Augmenting, Integrating & Proving.** We created a new and arguably highly non-trivial Tamarin model for REG. We used Tamarin existing models for AKA [12] and for XN handovers [31]. We aligned these models and combine then with our REG model. We augmented all models with new parties (e.g., UPF), PDU session handling, threat models (e.g., corrupt and honest-but-curious gNBs). In various combinations of these models, we prove that all properties that

were inherited still hold, as well as proving the new results above.

Our **second** contribution, detailed in Section 5, is an open-source emulator for XN and N2 handovers. Note that is also supports REG/AKA, but this is not its main focus. This emulator can be used by researchers as an API or building block for 5G experimentation.

Note: All our code and our most up-to-date Tamarin models can be found at [17].

4 FORMAL SECURITY VERIFICATION OF THE *5GAKE_STACK*

In this section, we show the first formal verification of the full *5GAKE_stack*, the formal verification of the Registration procedure, as well as the first formal analyses of the sub-procedures of the *5GAKE_stack* when varying the levels of security and trust in the 5G network. For this, we use the `Tamarin-prover`(Tamarin) [30], a well-know security protocol analysis tool.

Note: Our most up-to-date Tamarin models can be found at [17].

All our Tamarin models follow the relevant 3GPP specifications [1–5, 7–11] closely, which makes them faithful representations of these 3GPP specifications. This is desirable, but also renders some of the verifications labourious, in need of long manual proofs and/or automation by bespoke oracles. Even with these optimised oracles, the complexity of the models meant that our verification (and even just the loading of some models) is not possible without a very powerful machine to support it. Indeed, our verification experiments were run using Tamarin v1.6.1, on a server with two Intel™ Xeon™ E5-2667 V3 at 3.20GHz CPUs (8 cores, 16 threads each) and 378GB of RAM.

4.1 Tamarin Overview

Tamarin is a popular verification tool for symbolic analysis [15] of protocols; it supports an unbounded number of concurrent protocol sessions/executions in the full Dolev-Yao model [23].

Tamarin models are transition systems over a multi-sorted term algebra, operating on the semantics of multiset rewriting logic [24]. Security properties to be analysed are expressed in a guarded first-order logic quantifying over variable inside facts declared in the model. In most cases, for non-trivial models, Tamarin is used as an interactive tool, where the user guides the proof search.

Tamarin Oracles. Tamarin supports various heuristics to cover the search-space yielded by the constraint-solving problem underlying the analysis. These heuristics determine which rules should be prioritised during the proof search. However, a user can create a file to prioritise different rules yielding a bespoke search heuristic. This is called an "oracle"[5] and can be passed to Tamarin to automate otherwise user-guided proofs.

Weakly-typed Models. The most generic type of symbolic verification is the one in which messages inside models are not typed.

An alternative is *weakly typing* some of the messages. For instance, instead of leaving certain terms in an arbitrary form when deducible by the attacker, one can declare a functional symbol to be used in stating that the attacker should only try to compose/decompose specific terms pattern-matching this symbol. This weakly typing does not render the secrecy problem decidable [32], but makes it more tractable. Such techniques have been used before in Tamarin-verification when large systems (such as TLS1.3) were encoded and analysed [22].

In our work, we use oracles for all specification files and, occasionally, we weakly-type parts of the models, e.g, some identifiers used be a number of 5G messages.

[5]Please refer to the Tamarin manual [33] for more details on oracles.

4.2 Suitable Threats for the *5GAKE_stack*

We operate three separate levels of security and trust, as introduced in Section 3. We now argue, in the context of Tamarin, the suitability of applying one threat model or another to the different variants of the *5GAKE_stack* and/or the procedures therein.

4.2.1 The Threat Model of the Dolev-Yao (DY) Attacker. As stated previously, this is the default threat model in Tamarin, for all protocols. It equates to all parties being honest and a Dolev-Yao (DY) intruder actively mediating all communications.

Applying this threat model. Handovers and AKA have already been checked in the DY model, in [31] and in [12], respectively. So, we apply the DY model to: (1) the Registration procedure (as this procedure has never been formally verified before); (2) a version of the full *5GAKE_stack* , i.e., XN in composition with Registration procedure. We call this instance of the stack the *XN-based* 5GAKE_ stack.

Analysing the XN-based 5GAKE_stack. This version of the *5GAKE _stack* is arguably more meaningful to verify than the N2-based *5GAKE_stack*, due to the following reason: From a *pure* Dolev-Yao perspective with uncorrupted parties, XN and N2 are identical, e.g., they share the same (in-)secure channels, etc. However, there are more active parties in XN than in N2 (recall that in N2 the source gNBs are merely passive, while in XN the source gNBs play an active role). Therefore, the verification of a XN-based stack rather than an N2-based stack is more laborious, in a larger model, and could yield more insecurities.

We also consider two threat models of increasingly malicious gNBs, each cast in a Dolev-Yao environment. These are as follows:

4.2.2 The Threat Model of Honest-but-curious gNBs. As introduced and explained in Section 3, this captures gNBs which are simply configured to do just horizontal key-derivation in XN but otherwise follow the protocol. Their aim is to enable the recovery the next-hop AS keys.

Applying this threat model. It is sufficient to apply this threat model to a standalone XN procedure for the following reason: If backwards security is lowered in XN , it is also lower over the whole *5GAKE_stack*. Moreover, the models for the *5GAKE_stack* are less tractable.

4.2.3 The Threat Model of Corrupt gNBs. As introduced and explained in Section 3, this captures gNBs which do not follow the protocols, i.e., their software is modified with corrupted PDU session handling and they can change the PDU session types at will, e.g., either to lower or to up UEs' QoS.

Applying this threat model. We note that the parties informed of PDU session updates are the same in all three procedures. The management of PDU sessions by these parties is also the same in all three procedures. Hence, it does not matter which of the three

procedure this enhanced threat-model is applied to. As we already verified XN under enhanced threats as per the above, we decided to apply this attacker model to the Registration procedure, REG, especially, since REG has not been verified before and is also more tractable than the full *5GAKE_stack* model.

4.3 Specifications in Dolev-Yao Threat Model

In this subsection, we discuss our REG and *5GAKE_stack* Tamarin models. In these models, we also include several UPFs. While all the models in this subsection allow for an arbitrary number of UEs and gNBs, in some cases, for the sake of a proof's traceability, we restrict the number of other entities (e.g., PDU sessions), as we explicitly detail later.

4.3.1 Specification of the Registration procedure Procedure with Honest Parties.

Modelling. The Registration procedure [7, 11] is formed of three parts: (1) initial registration phase; (2) authentication and key agreement; (3) security key setup. We modelled all three parts in Tamarin.

To give an idea of the scale of each sub-procedure, the full Registration procedure consists of 100 messages [11], of which 30 form part one, the next 50 form AKA, and the last 20 messages form part three of the Registration procedure. In the corresponding Tamarin model, the full Registration procedure model without lemmas (i.e., just the protocol rules) has around 1260 lines, with 300 lines encoding AKA.

For the AKA part of Registration procedure, we adapted the model in [12] in [12], and embedded it into the brand new models for the rest of the Registration procedure. Note, however, that the model for AKA from [12] considered roaming, i.e., the UE is authenticating to its operator's network whilst being served by a different operator's network. Yet, we fitted this in a wider Registration procedure model, without roaming (i.e., the UE registers directly in the network of their operator). "Combining" the models together, ours and those in [12], was an aspect to be handled carefully; we did not remove anything from their models nor their lemmas, we simply[6] did not quantify over the the serving roaming network in certain statements/facts/predicates that pertained to the Registration procedure. We also added the extra logic that, once the Registration procedure finishes, the gNB node executing REG with the UE becomes the UE's effective, serving node.

Finally, unlike in the original AKA model in [12], in our full REG model, we also formalised 5G entities such as UPFs and 5G objects such as PDU sessions[7].

Clearly, with these additions, the new statements we proved and some of the predicates we used in the lemmas in our augmented model also quantify over these newly-added entities and objects; in turn, this alone increased multi-fold the complexity of the proofs in our REG model compared to the proofs in the AKA model in [12].

Verification Results. Firstly, we showed that all the sanity and security checks which held/failed in the AKA model in [12] still hold/fail in our full model for the Registration procedure. These results mean that the AKA part of our model was not affected by its transformation and inclusion in our Registration procedure model.

Secondly, alongside the lemmas inherited from the AKA model, we also showed anew that the full model executes securely and correctly in this Dolev-Yao model. To this end, lemma 1 together with lemma 2 in Table 5 demonstrate, in the DY threat model, the correctness of our full-Registration procedure model, as well as key-agreement across all parties at the end of the Registration procedure, in line with Table 1.

4.3.2 Specification of the XN-based 5GAKE_stack *with Honest Parties.* We formally modelled and verified the XN-based *5GAKE_stack* in the DY model.

Modelling. To create this Tamarin model for the XN-based *5GAKE_stack*, we used our aforementioned model for the Registration procedure, in which we embedded the very recent XN model in [31] augmented with certain aspects as the combination of these specification required careful modelling due to the following reasons:

(1) The AKA model in [12] involved 3 parties: the UEs, the serving network's *core* and the operator's *core*.
(2) Our Registration procedure sub-model mainly operates over 3 parties: the UEs, the operator's *core* and an "attaching" gNB.
(3) The XN model in Peltonen et al. operated over 4 parties, the UEs, a source node, a target node and the operator's *core*.
(4) First, in our combined model, we coherently "combined" these parties and made the "attaching" gNB in the REG model be the first s-gNB for an XN execution.
(5) Second, a meaningful XN model with the *5GAKE_stack* requires the cryptographic keys established during the AKA phase to be present. So, in our *5GAKE_stack*model, we included this type of correct information flow between procedures.

The XN sub-model allows for horizontal and for vertical key-derivations in sequence, as per the 3GPP specifications.

The AS message included at the end of the Registration procedure model presented Section 4.3.1 has been removed from our model *5GAKE_stack*, as it is not needed for proving key/data agreement for AKE-security within the *5GAKE_stack*; this avoided over-complicating an already complex model.

Verification Results We now recall the most important results in Table 5.

Firstly, we proved that the model is functionally correct, in the DY model: i.e., the *5GAKE_stack* can execute with XN handovers run on top of Registration procedure. For instance, Lemma 3 in Table 5 encodes the reaching of the end of an XN execution with horizontal key derivation. Lemmas 4 in Table 5 encodes that we can correctly do a horizontal-key-derivation handover followed by a vertical-key-derivation handover, on top of Registration procedure.

Lemmas 3 and 4 that in the XN model in Peltonen et al.; showing them prove in our model, which lifted XN to the *5GAKE_stack* as explained, ascertains the validity of our model but also the modular/incremental build that we applied to create it.

Secondly, we proved that the agreement on keys across parties involved in the AKA procedure as part of REG still holds, even when

[6]Since the serving network does nothing else other than passive proxying over secure channels (for both AKA and Registration procedure), this abstraction is arguably innocuous in AKE statements.

[7]We restrict the model to just one PDU session per UE; this encompasses all logical functionality of PDUs but makes the proofs/model more tractable. We also use, in the proofs for REG, UPFs without "smart filtering", which is sufficient since the gNBs are honest, in this model.

the sub-model for Registration procedure has been modified and included in the full *5GAKE_stack* model. See Lemma 5 in Table 5.

4.4 Specifications with Dishonest gNBs

4.4.1 Specification of the XN Procedure with Honest-but-Curious gNBs. We augmented the XN model in [31] to contain, alongside honest gNBs, also honest-but-curious gNBs. As per Section 3, recall that honest-but-curious gNBs can adaptively choose to do just HKD and so lower the backward-security of XN (recall Figure 3).

Modelling. We implemented the honest-but-curious gNBs in an as modular as possible way. We made gNBs have threat-types: honest or honest-but-curious. Then, parameterised some Tamarin rules to trigger based on these threat-types. To this end, the Tamarin rule "`sran_snd_ho_req_hkd`" for executing an XN-handover with HKD triggers only once the `GoodOrBadHandovers(state1,state2)` restriction holds. In this restriction, `state1` is the threat-type of the source gNB and `state2` is the threat-type of the target gNB. Then, the restriction ensures that an honest-but-curious gNB only ever hands over to another honest-but-curious gNB while an honest gNB can hands over to either an honest or honest-but-curious gNB. While this is somewhat unrealistic, it does show that an honest-but-curious gNB could maliciously try to weaken backward-security for as long as there are other honest-but-curious gNBs in its vicinity to serve the UE.

Verification Results. We recount the most important results stated in Table 5.

The executability lemmas inherited from the original model continue to prove showing continued functional and correctness in the case of honest gNBs, even in the presence of malicious ones. This is captured in Lemma 6 and Lemma 7 in Table 5, which show that honest gNBs do series of HKD and/or VKD handovers just like before.

The main lemmas of most interest in this model are Lemma 8 and Lemma 9 in Table 5. The first one shows that honest-but-curious gNB always handover to another honest-but-curious gNB, while the second shows that as soon as an honest gNB hands over to a honest-but-curious gNB, the subsequent handover will be to another honest-but-curious gNB. In total, this formally shows in 5G, one get be handed over into a subset of honest-but-curious gNBs, and then persist in that subnet, where the AS security is lacking.

Finally, via Lemma 9 in Table 5, we show that in this augmented model for XN, injective agreement between the UE and the target gNB w.r.t. the established security key is maintained, i.e., the statements in Tables 2 and 3 hold even in this augmented model for XN.

4.4.2 Specification of the Registration Procedure with Corrupted gNBs. As we explained in Section 3, at the end of the Registration procedure, the serving gNB will have an AS channel with the UE which will be used to route the UE's traffic to the internet via some UPFs. In Section 3, we also alluded that if these gNBs became rogue, then they could manipulate the PDU session lists of the UEs to route them at will, lowering or upping their QoS. In one new model for the Registration procedure, we formally prove this.

Modelling We carry forward our model for the Registration procedure described in Section 4.3.1 and enhance it.

PDU Session Modelling. In our model, we abstracted a PDU session to simply have an ID and a type. To further reduce the complexity, we only model a PDU session list of length 1, i.e., one PDU session per UE. Note that if we prove insecurities in these idealised and much simplified settings, then these insecurities would thus also apply to a full PDUSL.

Dishonest gNBs' Modelling. In this model, we have two types of gNB nodes: (1) the honest gNB that forwards the AS messages to UPFs with the correct PDUSL, as declared by the UE. (2) the dishonest/corrupt gNB that can manipulate PDUSL and replace it with PDUSL*, where both the type and ID of the session is changed.

This encapsulates that not only can a dishonest gNB manipulate the type of a session for a given UE, but such a dishonest gNB can even replace the session with a PDU session of another UE that it currently serves.

UPFs' Modelling. Finally, we model two behaviours for the UPFs (1) without "smart filtering", i.e., not checking the PDUs received from the gNB; (2) with "smart filtering" enabled, checking the PDUs received from the gNBs against what it holds upon previous receipt from the *core*

Verification Results. In this model, we first formally (dis)prove again all the lemmas we checked for the model of the Registration procedure in the case of honest gNBs; the verification results stay unchanged, as expected. An example of this is lemma 11 in Table 5.

In Table 5, lemmas 14-17 are the most important for this model. They show that honest gNBs route traffic as per PDU sessions, and all is correct even if the UPF does not do "smart filtering" (lemma 14), or if the UPF does do "smart filtering" (lemma 15). In turn, lemma 16 and lemma 17 show that a corrupt gNB manipulating PDUSLs can be detected and stopped if smart filtering is enabled on the UPF, but not otherwise.

4.5 Learnings & Recommendation

4.5.1 Learnings w.r.t. 5GAKE_stack *with Honest gNBs.* W.r.t. our models in the "pure" Dolev-Yao setting, one obvious lesson is that it is non-trivial to "combine" formal models for (sub-)procedures of the *5GAKE_stack*, where each sub-procedure operates over different sets of parties. However, this allowed for a partial re-certification of existing models, coming from different sources (i.e., from [31] and [12]), by aligning them with our models and yielded the first full and verified model of the *5GAKE_stack*.

A second lesson is that the increased complexity of verifying our new models is significant, e.g., our "Registration-procedure part1 (Reg-p1) + AKA + Registration-procedure part2 (Reg-p2)" model vs. the "AKA" model from [12], or our "Reg-p1 + AKA + Reg-p2 + XN" model vs. the "XN" model from [31]. Not only did we need to create several bespoke oracles to manage the proofs, but the increase in time is 3 to 4-fold for AKA and about 10-fold for XN.

Finally, since some aspects of the model/proofs are currently intractable for more than one handover (see Table 5), as future work, we intend to simplify the model in a systematic way and fine-tune the oracles and then (dis)prove more lemmas w.r.t. various key-agreements over several handovers done in series.

4.5.2 Learnings & Recommendations w.r.t. Dishonest gNBs. In Section 4.4.2, we formally showed that –in the REG procedure– rogue gNBs can affect the QoS of UEs by manipulating their PDUs. We

	Lemma	Meaning	Proving Status & Method	Threat Model	Protocol	Time
1	Correctness-UE-Reg-AKA-Honest	Full executability of our REG model, incl. AKA, using a gNB send traffic to an UPF which has no smart filtering enabled	Proved, automated with an oracle	full DY model	REG incl. AKA	1m40s
2	anonymous-injectiveagreement-ue-seaf-kseaf-noKeyRev-noChanRev	The UE and the *core* agree on K_{SEAF} (see Figure 2 or Figure 6), if no key reveal/leakage occurs	Proved, automated with an oracle	DY model (no key reveal)	REG incl. AKA	5m15s
3	executability-HO-hkd	XN^{hkd} can correctly execute after *REG*	Proved, automated with an oracle	full DY model	*5GAKE_stack*	234m49s
4	executability-HO-hkd-vkd	XN^{vkd} can correctly execute after XN^{hkd}	Not proved	full DY model	*5GAKE_stack*	OOM
5	anonymous-injectiveagreement-ue-seaf-kseaf-noKeyRev-noChanRev	The UE and the *core* agree on K_{SEAF} (see Figure 2 or Figure 6), if no key reveal/leakage occurs	Proved, automated with an oracle	DY model (no key reveal)	*5GAKE_stack*	7m55s
6	executability-hkd-hkd	XN^{hkd} can correctly executed twice even if HBC nodes are present	Proved, automated with an oracle	DY model + HBC gNBs	XN	87m12s
7	executability-hkd-vkd	XN^{hkd} and then XN^{hkd} can be correctly executed if honest-but-curious nodes are present	Proved, automated with an oracle	DY model + honest-but-curious gNBs	XN	86m40s
8	executability-bad-to-bad	A honest-but-curious gNB always hands over to a honest-but-curious gNB	Proved, automated with an oracle	DY model + honest-but-curious gNBs	XN	5m02s
9	executability-good-to-bad-general	Once an honest gNB hands over to a honest-but-curious gNB, subsequent gNBs will be honest-but-curious	Proved, automated with an oracle	DY model + honest-but-curious gNBs	XN	39m02
10	injectiveagreement-ue-tran-k-gnb	There exists exactly one honest gNB that derived the same key.	Proved, automated with an oracle	DY model + honest-but-curious gNBs	XN	35m38s
11	secret-supi	The attacker only knows the SUPI if it was revealed.	Proved, automated with an oracle	DY model + dishonest gNBs	REG	7m12s
14	Correctness-UE-Reg-AKA-Internet-Honest-noUPFCheck	If the gNBs are honest and if UPFs do not smart filter, the PDUSL and UE's Internet-access are correct	Proved, automated with an oracle	DY model + dishonest gNBs	REG	17m58s
15	Correctness-UE-Reg-AKA-Internet-Honest-withSmartFilter	If gNBs are honest and UPFs do smart filtering, the PDUSL and UE's Internet-access are correct.	Proved, automated with an oracle.	DY model + dishonest gNBs	REG	16m58s
16	Correctness-UE-Reg-AKA-Internet-Corrupt-withSmartFilter	UPFs with smart filtering detect a changed PDUSL* sent by a corrupt gNB and the UE's traffic is stopped	Proved, automated with an oracle	DY model + dishonest gNBs	REG	18m05s
17	Correctness-UE-Reg-AKA-Internet-Corrupt-noUPFCheck	UPFs with no smart filtering do not detect a changed PDUSL* by a corrupt gNB and the UPF grants the UE incorrect Internet-access	Proved, automated with an oracle	DY model + dishonest gNBs	REG	16m35s

Table 5: Main Verification Results

showed this in a model for REG (for purposes of tractability), but the way the AS messages are manipulated by the gNBs is the same at the end of any sub-procedure in the *5GAKE_stack*. So, this matter affect all AS level of the 5G network. We show that this can be stopped if the UPF does "smart filtering" and checks the correctness of the PDU sessions it is routing onto for a given UE.

As a result, we recommend that "smart filtering" be no longer optional as per the 3GPP specifications [4, 7, 8, 11], but mandatory.

We further recommend that if a UPF with smart filtering finds such a rogue gNB, then an error message be sent to the *core*, which in turn should inform the UE using a K_{AMF}-encrypted message.

4.5.3 Learnings & Recommendations w.r.t. Honest-but-curious gNBs. In Section 4.4.1, we showed that honest-but-curious gNBs can act (together) to systematically lower the backward security of the K_{gNB} key.

This means that N2, where the source gNBs do not actively compute the next-hop K_{gNB} key, should be preferred to XN. But, as the next section will show, N2 is more costly communication-wise than XN.

So, a possible alternative, which we do not detail in here, is a protocol whereby the target gNB would contribute actively to the computation of the next-hop K_{gNB} during XN^{hvd}, with a value that the source gNB does not know, and thus would be unable to compute the next-hop K_{gNB}. This protocol would require a new input by the target gNB be sent to the *core* and from the *core* to the UE (using encryption with K_{AMF}) and thus provide an acceptable compromise in terms of security and communication complexity.

Another solution relies on the UE, which can theoretically check if a Horizontal Key Derivation or a Vertical Key Derivation is due in an XN handover (by seeing if it has an unused NH), and thus catch when a rogue gNB is maliciously forcing a hkd in XN. This solution is somewhat impractical, as the UE is informed by the

gNBs as to what type of key-derivation to do, in order to cover for losses/desynchronisation of data by the UE, e.g., after a total signal loss.

Finally, we have contacted working groups at ETSI about our findings; the conversations are incipient and on-going.

5 *5GAKE_STACK* EMULATOR

We implemented an emulator of the *5GAKE_stack*. As we are mainly focused on AKE aspects, our emulator also focuses on the handling and update of the keys across the different 5G entities, and we thus abstracted away some message data. We call this open-source, proof-of-concept implementation: 5GAKE_C. The 5GAKE_C implements the initial registration and AKA, as well as the different handover mechanisms. Here, we focus on describing the implementation of the handovers, as – in general– these are less well understood than AKA. 5GAKE_C is written in C++.

The latest version of the code is available at [17]

5.1 Overview of the *5GAKE_stack* Emulator

5GAKE_C focuses on AKA, as well as both handover procedures: XN (with both vertical and horizontal key derivation) and N2 (with and without re-keying). That is, we are interested primarily on AKE aspects, 5GAKE_C leaves out most details of the REG procedure that do not pertain to AKA.

More concretely, within 5GAKE_C we include most exchanges of messages sent back and forth, but we leave out some details related to the message contents, which are orthogonal to the key-updates. Also, in the emulation of 5G communication, we do not attempt to reproduce the network layers. Instead we focus on representing the messages which are passed in the protocols.

Modular Implementation. We built this emulator to be easily extendible to other procedures in 5G. We now explain this modular aspect. We first created a message-passing logic that can support other 5G protocols. Each entity in the model runs in its own thread and manages a multi-threaded queue for its input messages. Output messages are placed in the queue of a 'router' entity that then puts the message into the appropriate entity's input queue. To enable this, each of the different entities register their input queue with the router. Each of the (5G) entities has an ID, which the router uses as their "address" for this message passing. Further, messages in 5GAKE_C also have a rather generic format, shown below:

Source	Dest.	Ref. Pt.	RC	SUCI	Payload

Figure 4: Generic Message Format

- *Source* – the ID of the source entity;
- *Dest* – the ID of the destination entity;
- *Ref. Pt.* – the 3GPP "reference points" for the interfaces (see Figure 1) N1 interface (N1), N2, Xn, N12 and N13. In addition we defined several extra "reference points": RRC for UE to gNB messages and CMD for messages used by the controlling program to trigger the different actions (command messages);
- *RC* – the "reason code" for the message, each of the messages has a reason code which is used to identify the particular message and its role in the protocol;
- *Subscribers Universal Concealment Identity (SUCI)* – the standard Subscribers Universal Concealment Identity, which in the 5GAKE_C is abusively used in all messages to identify the relevant UEs;
- *Payload* – the actual data contained in a particular message.

5G Entities Implemented in 5GAKE_C. As we focussed on AKA and AKE we only implemented the entities involved in these procedures. In addition, we focussed on the NAS and took no account of user plane functions and the management of PDUs on handover. This will be considered for future work on 5GAKE_C.

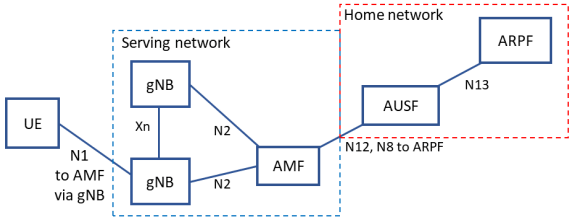

Figure 5: Entities in 5GAKE_C.

Figure 5 shows the implemented entities:

- *UE.* The user equipment which contains a Subscriber Identity Module (SIM), securely holds the mobile operator's data and carries out the calculations needed for authentication.
- *gNB.* The radio node in the 5G network. It provides the link between the UE and the core network. Apart from the initial messages, communication between a gNB and the UE is integrity and privacy protected. The keys used to do this are generated as part of the AKA procedure and updated as part of any handover process.
- *AMF.* The Access and Mobility Management Function is part of the serving network (SN). It interacts with the AUSF and the UE, and stores the intermediate keys that are established as a result of the UE authentication process.
- *AUSF.* The Authentication Server Function is part of the home network and manages the authentication of the UE.
- *ARPF.* The Authentication Credential Repository & Processing Function is part of the *core.* It stores the sensitive data and carries out the calculations needed for the authentication process. In the model, it also includes the functionality of the Unified Data Management (UDM) which provides support for the generation of the AKA authentication credentials and for storage and management of the SUPI for each subscriber in the home network.

5GAKE_C supports an arbitrary number of UEs and of gNBs, but only one of each of the other entities (AMF, AUSF and ARPF). This implies that in our implementation, there is, e.g., no change in the AMFs during handovers.

In 5GAKE_C, each entity is represented by a C++ class and each of the actual entities in the model runs in its own thread. As outlined above, to facilitate the message passing a Msg_router class is used and this also runs in its own thread. Each entity has an input queue that it registers with the Msg_router. The Msg_router registers its input queue with each entity and this is used by them for sending messages. All messages from a UE are sent via their current gNB.

Combining AKA and Handovers inside 5GAKE_C. The C++ classes defined for 5GAKE_C are used in two programs: one designed to test the XN handovers and the other the N2 handovers. The programs can be compiled with a number of options:

- PRINT. This prints the protocol message flow and details about the different keys and entities involved. The output for part of an N2 handover is shown in Figure 10.
- MSG_LOG. This prints the messages being sent. Figure 9 show some of the output in this case.
- MSG_TIMING. This includes the protocol timings. Results from this option are shown in Table 6. For this option all of the other options should be disabled to avoid timing differences arising due to the other processes running.
- DEBUG. As the name suggests, this is used for debugging and prints much more information.

For both programs, any executed emulation of the *5GAKE_stack* starts with each UE in the model being sent a command to "attach" to a gNB. The UE is "attached" to the gNB and this triggers the start of the AKA protocol, The AKA protocol runs without further intervention using a Finite State Machine (FSM) on each of the entities, with each entity involved in authenticating the given UE (a gNB, the AMF, the Authentication Server Function (AUSF), and the Authentication Credential Repository & Processing Function (ARPF)), maintaining an FSM for that UE. The messages in the FSM models use the UE's SUCI as an ID. It is easy to add, in 5GAKE_C, the handling of 5G-GUTIs once the UE is authenticated, but –focusing on the AKE aspects– we left this for future work.

At the end of a successful authentication, the relevant keys are distributed to each entity, as per the full 3GPP specifications of AKA [8], or as recalled in Section 2.2, in Table 1. At this point, each FSM is in an active, but idle state ready for further processing.

In either program, the execution-flow is now at the stage where commands can be sent to trigger the relevant handovers. A source gNB (s-gNB) is sent a command to handover to another specified target gNB (t-gNB). In a real system, this decision is triggered by measurements by the s-gNB, which it then sends in a measurement report. We obviously do not have these, so we prompt a handover by sending a handover command, instead.

Handovers can be done directly between gNBs using the XN interface/reference point to handle the process, or, where there is no direct communication between the gNBs, via the core (using the N2 interface). There are command codes in 5GAKE_C to trigger these various options.

Before describing these two options, note that at the end of a handover, the source gNB removes its FSM for the UE, while the target gNB now has an FSM for the UE.

5.2 The XN Implementation in 5GAKE_C

We implemented the XN protocol [11] shown in Figure 7 and one sample XN execution in 5GAKE_C can be seen in Figure 9. The reason codes can readily be matched up with the messages shown in the figure. To save space the 8 byte SUCI is simply replaced by "SUCI".

In 5GAKE_C, we have defined a command to a gNB to trigger a handover. Once triggered the XN protocol shown in Figure 7 runs without intervention. If the source gNB has a (Next Chain Counter (NCC), NH) pair available then vertical key derivation is used, otherwise horizontal key derivation is used. Directly after the AKA protocol has completed the gNB has no (NCC, NH) pair available and so the first handover is always a horizontal one. At the end of this handover, the gNB receives an (NCC, NH) pair and so, unless the pair is used for other procedures the following handover will be a vertical one. To "use" the (NCC, NH) pair a second command to a gNB has been defined, this just marks the (NCC, NH) pair as being used so that the next handover that is triggered will be a horizontal one. The program used to test the XN protocol has one UE, three gNBs and one of all of the other entities. The program starts with the UE 'attaching" to one of the gNBs and executing the AKA protocol. A first handover is then triggered which will be, as explained above, a horizontal one. This is then followed by a second handover and this can be either a horizontal or vertical one depending on a parameter given as the program is run.

5.3 Experimentation with 5GAKE_C

For each handover carried out, we can measure the total number of bytes transferred in the messages and the time taken. The time taken is measured from the first message from the source gNB that triggers the protocol until the final message. For XN this is the Handover Success message from the target gNB to the source gNB, while for the N2 this is the UE Release message from the source gNB to the AMF.

Note: For the tests the code was compiled using x86_64-linux-gnu-g++ version 10.3.0 with optimisation set at -O3. The code was run on an Intel x86_64 processor with 8 i7-8550U CPU cores running at 1.80GHz. How the threads in the model were distributed over these cores was not specified, but left to the Linux OS. The results that we obtained are shown in Table 6.

Handover	Total bytes	Time taken/µs (mean / std.dev.)
XN – horizontal	126	943 / 177
XN – vertical	126	1023 / 136
N2 – no re-keying	182	1389 / 309
N2 – with re-keying	182	1454 / 312

Table 6: Measurements from 5GAKE_C.

For both the XN handover, the messages exchanged are the same. It is the calculations carried out by the different entities that vary. The number of bytes transferred in each case is the same. The same situation applies for the N2 handovers, the number of bytes transferred is the same for either of the N2 handovers. Table 6 also shows that more data is transferred when executing the N2 protocol, even in our stripped down model.

The timing results also show (albeit in a crude way) that the N2 handovers are more resource intensive.

5.4 Future of 5GAKE_C

Currently 5GAKE_C abstracts away many of the lower-level details of the protocols. There is scope to make it more realistic by "correctly' including more details of the messages and, for example, handling the 5G Globally Unique Temporary Identity (5G-GUTI) and the

transfer of PDU sessions. This will involve implementing more of the 5G entities, but adding them into the model is straightforward.

At the code level, we plan to make running it more straightforward by using a configuration file to drive it rather than the current command-line options. While the programs described here either just execute XN or just N2 handovers, these can be mixed and this will be easier to control with a configuration file. The message passing and processing with FSMs is not restricted to 5G and the code can be readily adapted to the testing and evaluation of other protocols.

6 RELATED WORK

Previous work on the *5GAKE_stack* as one single, composed protocol is sparse. Therefore, we focus on exploring prior work on the sub-procedures of the *5GAKE_stack*. Also, this work is on 5G environments, so our related-work section will focus of the 5th generation too, and not so much on prior ones. Similarly, the scope of covered prior work is mainly formal analyses. We are aware of numerous studies which may be related otherwise, but we are space constraint.

Formal Analyses of authentication and key agreement The 5G AKA procedure is the most widely researched protocol within the *5GAKE_stack*. Basin et al., in [12], formally verify it in the Dolev-Yao model, w.r.t. secrecy, synchronisation, and authentication properties; some synchronisation attacks between the UE and the core were found. Their models are in Tamarin and we re-used parts of them herein. We showed new attacks by combining these models with our Registration models, and enhancing the threat model.

A study of the privacy of 5G AKA is presented by Koutsos in [28]; this is done in by a cryptographic model which was then not mechanised within a tool. Orthogonally, Borgaonkar et al., in [16], carry out a privacy-centred study of the AKA procedure (in 3G, 4G and 5G) in the Dolev-Yao Model. Here, we are not concerned with privacy.

Another piece of research into the 5G AKA is by Edris et al. in [25]. Edris et al. used the ProVerif tool [14] to formally analyse 5G AKA, and found some linkablity and replay attacks, under arguably very strong adversarial capabilities.

Formal Analyses of Handovers. There have been multiple different studies for 4G handovers:[13, 20, 21]. These works verified 4G handover procedures (in much simpler settings and models than herein), using ProVerif [14]. Also, a core part of the verification in [13] was focused just on secrecy properties. Meanwhile, [20, 21] did consider payload security and forward/backward secrecy, but all in 4G.

Very recently, Peltonen et al. in [31] proposed the formal verification of 5G handover procedures. This was done in the Dolev-Yao model, against secrecy and authentication properties. No attacks were found by this work, which assumes that all parties are honest. We extended this work in two ways: (1) to consider dishonest parties, (2) to integrate it with the Registration and AKA procedures into one model. We find new insecurity results.

Formal Analyses of Registration procedure. The UE Registration procedure has had the least amount of research, from all the procedures discussed herein. Chen et al., in [18], looks at the possibility of Denial of Service (DOS) within the Registration procedure, but only from a practical experimental approach.

This Work vs. Closely-related Prior Work. In this work, we use the aforementioned AKA models by Basin et al. in [12] and the handover models by Peltonen et al. in [31]. However we do not only extend their models to larger procedures, but we also add more details to them and combine them. Moreover, we add a stronger threat-model to each of the individual models, as well as to the "composed" model for the *5GAKE_stack*. We do revisit the exact findings in either of the inherited models and check the corresponding ones in the extended/composed models too. Finally, we actually focus on verifying the properties of the *5GAKE_stack* in our "composed" model, as well as checking security in our enhanced threat-models. In this setting, we show new attacks and make recommendations to improve the standards.

7 CONCLUSIONS

We formally analysed the REG, AKA procedure (AKA) and the N2/XN handover procedure 5G procedures, from the AKE perspective, as individual procedures and in sequential composition. We varied the threat model, showing formally that even when the radio-nodes are just honest-but-curious and not fully malicious, the overall AKE-security of the procedures can be vastly lowered. Whilst some of these findings can be performed by pen-and-paper analysis after careful understanding of the procedures, we are the first to prove this formally (using a protocol-prover) for any of the procedures, as well as formally doing so for the composition of the procedures. What is more, our models are a faithful representations of the 3GPP specifications [6].

We also built an emulator, `5GAKE_C`, of the procedural stack and used it to empirically assess the communication complexity of the procedures.

In future work, we wish to improve the proof heuristics (i.e., Tamarin oracles) linked to our Tamarin files, as –due to the faithfulness of our modelling and the complexity of the procedures – many proofs take an exceedingly long time even on multi-core machines. We also intend to further develop `5GAKE_C` into a more mature 5G emulator and include more 5G data-fields in the messages.

ACKNOWLEDGMENTS

Rhys Miller acknowledges National Cyber Security Centre (NCSC) funds under the "5GTech-Sec" project, and Ioana Boureanu acknowledges EPSRC grant reference EP/S024565/1 "AutoPaSS".

REFERENCES

[1] 3GPP. 2020. *5G Security Assurance Specification (SCAS); Access and Mobility management Function (AMF).* Technical Specification (TS) 33.512. 3GPP. Version 16.3.0.

[2] 3GPP. 2020. *5G Security Assurance Specification (SCAS) for the Session Management Function (SMF) network product class.* Technical Specification (TS) 33.515. 3GPP. https://www.etsi.org/deliver/etsi_ts/133500_133599/133515/16.02.00_60/ts_133515v160200p.pdf Version 16.2.0.

[3] 3GPP. 2020. *Interface between the Control Plane and the User Plane nodes.* Technical Specification (TS) 29.244. 3GPP. https://www.etsi.org/deliver/etsi_ts/129200_129299/129244/16.05.00_60/ts_129244v160500p.pdf Version 16.5.0.

[4] 3GPP. 2020. *Non-Access-Stratum (NAS) protocol for 5G System (5GS).* Technical Specification (TS) 24.501. 3GPP. https://www.etsi.org/deliver/etsi_ts/124500_124599/124501/16.05.01_60/ts_124501v160501p.pdf Version 16.5.1.

[5] 3GPP. 2020. *PDU session user plane protocol.* Technical Specification (TS) 38.415. 3GPP. https://www.etsi.org/deliver/etsi_ts/138400_138499/138415/16.01.00_60/

ts_138415v160100p.pdf Version 16.1.0.
[6] 3GPP. 2020. Release 16. https://www.3gpp.org/release-16
[7] 3GPP. 2020. *System architecture for the 5G System (5GS).* Technical Specification (TS) 23.501. 3GPP. https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/16.06.00_60/ts_123501v160600p.pdf Version 16.0.0.
[8] 3GPP. 2020. *System architecture for the 5G System (5GS).* Technical Specification (TS) 23.501. 3GPP. https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/16.06.00_60/ts_123501v160600p.pdf Version 16.0.0.
[9] 3GPP. 2020. *Wireless and wireline convergence access support for the 5G System (5GS).* Technical Specification (TS) 23.316. 3GPP. https://www.etsi.org/deliver/etsi_ts/123300_123399/123316/16.04.00_60/ts_123316v160400p.pdf Version 16.0.0.
[10] 3GPP. 2020. *Xn Application Protocol (XnAP).* Technical Specification (TS) 38.423. 3GPP. https://www.etsi.org/deliver/etsi_ts/138400_138499/138423/16.02.00_60/ts_138423v160200p.pdf Version 16.2.0.
[11] 3GPP. 2021. *Procedures for the 5G System.* Technical Specification (TS) 23.502. 3GPP. https://www.etsi.org/deliver/etsi_ts/123500_123599/123502/15.02.00_60/ts_123502v150200p.pdf Version 16.7.0.
[12] David Basin, Jannik Dreier, Lucca Hirschi, Saša Radomirovic, Ralf Sasse, and Vincent Stettler. 2018. A Formal Analysis of 5G Authentication. In *CCS '18 (CCS '18)*. ACM, New York, NY, USA, 1383–1396.
[13] Noomene Ben Henda and Karl Norrman. [n. d.]. Formal Analysis of Security Procedures in LTE - A Feasibility Study. In *RAID '14* (2014) *(LNCS)*. Springer, 341–361.
[14] Bruno Blanchet. 2001. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *CSFW '14*. IEEE Computer Society, 82–96.
[15] Bruno Blanchet. 2012. Security protocol verification: Symbolic and computational models. In *POST '12*. Springer, 3–29.
[16] Ravishankar Borgaonkar, Lucca Hirschi, Shinjo Park, and Altaf Shaik. 2019. New Privacy Threat on 3G, 4G, and Upcoming 5G AKA Protocols. *PoPETs '19* 2019, 3 (2019), 108–127.
[17] Ioana Boureanu, Stephan Wesemeyer, Chris Newton, and Rhys Miller. 2022. 5G Tamarin models and 5G Emulator. GitHub, https://fmsec.github.io/5gtechsec.github.io/.
[18] Chien-Ming Chen, Tsu-Yang Wu, Raylin Tso, and Mu-En Wu. 2014. Security Analysis and Improvement of Femtocell Access Control. In *Network and System Security*. Springer International Publishing, Cham, 223–232.
[19] Katriel Cohn-Gordon, Cas Cremers, and Luke Garratt. 2016. On Post-compromise Security. In *CSF '16*. 164–178. https://doi.org/10.1109/CSF.2016.19
[20] Piergiuseppe Bettassa Copet, Guido Marchetto, Riccardo Sisto, and Luciana Costa. [n. d.]. Formal verification of LTE-UMTS and LTE-LTE handover procedures. 50 ([n. d.]), 92–106. https://doi.org/10.1016/j.csi.2016.08.009
[21] Piergiuseppe Bettassa Copet, Guido Marchetto, Riccardo Sisto, and Luciana Costa. [n. d.]. Formal verification of LTE-UMTS handover procedures. In *ISCC '15* (2015). IEEE, 738–744.
[22] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. 2017. A comprehensive symbolic analysis of TLS 1.3. In *CCS '17*. 1773–1788.
[23] D. Dolev and A. Yao. 1983. On the Security of Public-Key Protocols. *IEEE Trans. Inf. Theory 29* 29, 2 (1983).
[24] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. 1999. Undecidability of Bounded Security Protocols. In *FMSP'99*. ACM.
[25] Ed Kamya Kiyemba Edris, Mahdi Aiash, and Jonathan Kok-Keng Loo. 2020. Formal verification and analysis of primary authentication based on 5G-AKA protocol. In *SDS '20*. IEEE, 256–261.
[26] Erricson. 2020. 5 key facts about 5G radio access networks. https://www.ericsson.com/495922/assets/local/policy-makers-and-regulators/5-key-facts-about-5g-radio-access-networks.pdf.
[27] Infrastructure and Projects Authority. 2018. Analysis of the National Infrastructure and Construction Pipeline. https://www.gov.uk/government/publications/national-infrastructure-and-construction-pipeline-2018
[28] Adrien Koutsos. 2019. The 5G-AKA authentication protocol privacy. In *EuroS&P '19*. IEEE, 464–479.
[29] Philippe Z Lin, Charles Perine, and Rainer Vosseler. 2017. Attacks From 4G/5G Core Networks: Risks of the Industrial IoT in Compromised Campus Networks. https://documents.trendmicro.com/assets/white_papers/wp-attacks-from-4G-5G-core-networks.pdf.
[30] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. [n. d.]. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In *CAV '13* (2013) *(LNCS)*. Springer, 696–701.
[31] Aleksi Peltonen, Ralf Sasse, and David Basin. 2021. A Comprehensive Formal Analysis of 5G Handover. In *WiSec '21 (WiSec '21)*. ACM, New York, NY, USA, 1–12.
[32] Y. S. Ramakrishma and S. A. Smolka. 1997. Partial-Order Reduction in the Weak Modal Mu-Calculus. In *CONCUR '97 (LNCS, Vol. 1243)*. Springer, 5–24.
[33] The Tamarin Team. 2016. Tamarin prover manual. https://tamarin-prover.github.io/manual/tex/tamarin-manual.pdf [Online: accessed 09-April-2019].

A ASPECTS OF THE REGISTRATION PROCEDURE

In this appendix, we show via Figure 6 which parties run the REG procedure, and represent which main keys are refreshed in this procedure and which parties hold these keys. As shown on Figure 6, the REG procedure [7, 11] is split into three parts: (1) initial registration; (2) authentication and key agreement [8]; (3) security key setup .

The objects crossed in red denote the fact that they are no longer to be used and are to be deleted from memory. The dots denote that other keys are present, but they are not of any interest herein. NCC stands for a counter of how many NHs have been used in security keys K_{gNB} since the last REG.

B ASPECTS OF THE XN PROCEDURE

In this appendix, we show via Figure 7 which parties run the XN procedure, and represent which main keys are refreshed in this procedure and which parties hold these keys.

The objects crossed in red denote the fact that they are no longer to be used and are to be deleted from memory. The dots denote that other keys are present, but they are not of any interest herein. NCC stands for a counter of how many NHs have been used in security keys K_{gNB} since the last REG.

C ASPECTS OF THE N2 PROCEDURE

In this appendix, we show via Figure 8 which parties run the N2 procedure, and represent which main keys are refreshed in this procedure and which parties hold these keys.

The objects crossed in red denote the fact that they are no longer to be used and are to be deleted from memory. The dots denote that other keys are present, but they are not of any interest herein. We did not detail on elements such as NAS Count (NASC) but they tell the UE which flavour of the procedure was run, so that the UE can replicate the correct calculations of the security key.

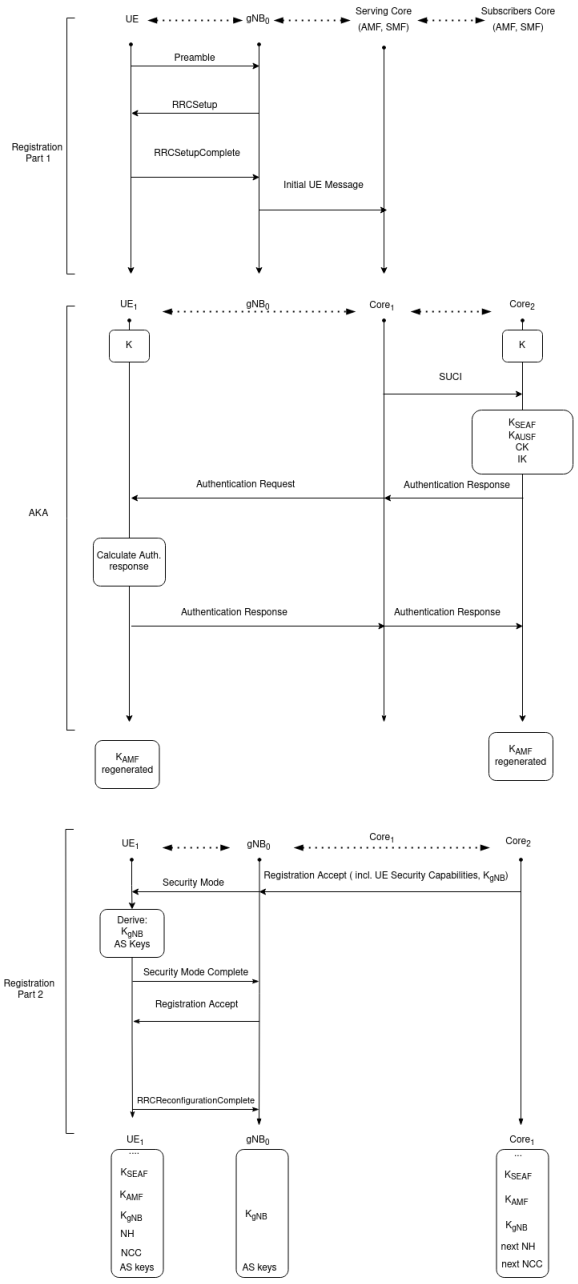

Figure 6: The *5GAKE_stack* Short-term Keys' Update and Distribution at the End of REG

D EXAMPLE EXECUTIONS OF 5GAKE_C

In this appendix, we give more details about the 5GAKE_C code, as well as its outputs. The output for part of an N2 handover execution is shown in Figure 10 and Figure 9 shows the messages sent between different entities.

The messages for the XN follow the protocol shown in Figure 7. Details of the messages (given their reason codes) are:

Xn ho req. The s-gNB sends a handover request to the t-gNB. The payload contains the UE ID, the current Cell Radio Network Temporary Identifier (C-RNTI), the new KgNB and the NCC value.

Xn ho ack. The t-gNB responds with an acknowledgement. The payload contains the old C-RNTI, the new C-RNTI and NCC.

Xn ho UE. The s-gNB sends the handover information to the UE. The payload contains the t-gNB ID, the new C-RNTI and NCC.

UE ho acc. The UE accepts the request.

Xn PDU id. The t-gNB sends information about the transfer to the AMF. This is usually where the PDU sessions are handled, but the AMF also sends a (NCC,NH) to the t-gNB. Although not currently handling PDU sessions, we do want the t-gNB to receive the (NCC,NH) pair.

Xn ho comp. The t-gNB sends a handover completion message to the s-gNB.

AMF ack. The AMF acknowledges the handover. The payload contains NH and NCC.

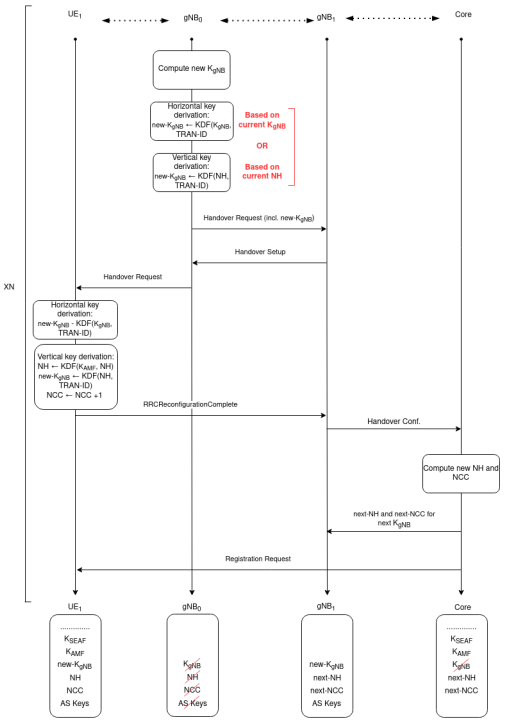

Figure 7: The *5GAKE_stack* Short-term Keys' Update and Distribution at the End of XN

Similarly, the messages for N2 follow the protocol shown in Figure 8. These are the message details:

N2 ho req. The s-gNB sends a handover request to the AMF. The payload contains the t-gNB ID, the UE ID and a flag to indicate whether to re-key, or not.

AMF ho req. The AMF send a handover request to t-gNB. The payload contains the UE ID, NH, NCC and the NASC data (in our case this is the key change flag, the key-set identity and the downlink NAS count).

N2 ho ack. The t-gNB acknowledges the request. The payload contains the NASC again.

N2 ho cmd. The AMF sends the handover data to s-gNB. The payload contains t-gNB ID, NCC and NASC.

N2 ho cmd. The s-gNB forwards the handover data to the UE.

UE ho cfrm. The UE confirms its acceptance of the handover to t-gNB.

N2 ho comp. The t-gNB indicates to the AMF that the handover is complete.

gNB release. The AMF informs the s-gNB that the handover is complete.

gNB confirm. the s-gNB confirms the release.

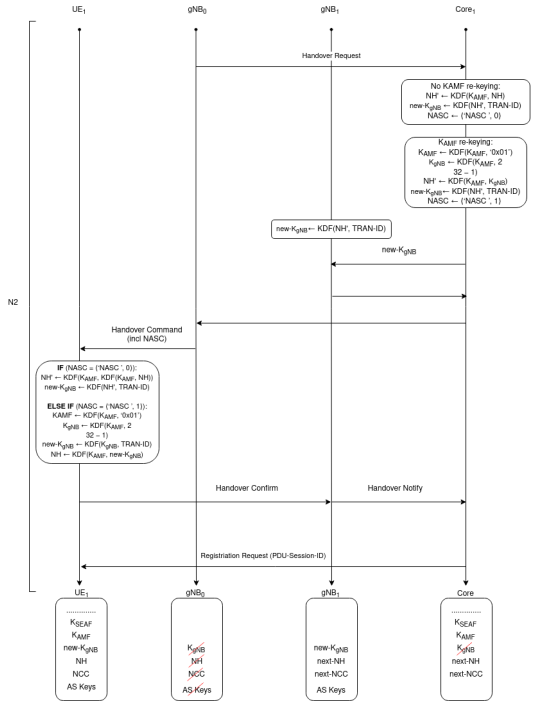

Figure 8: The *5GAKE_stack* Short-term Keys' Update and Distribution after N2

```
gNB11 | gNB12 |  Xn  |  Xn ho req  | SUCI | Payload: 010001aa61a31fd4e0c86ed9fe
8cdc522edbc37e2fee0b0c61237f0da8eaedd51f9f7a00000000
gNB12 | gNB11 |  Xn  |  Xn ho ack  | SUCI | Payload: 0001000300000000
gNB11 |  UE01 | RRC  |  Xn ho UE   | SUCI | Payload: 0c000300000000
UE01  | gNB12 | RRC  |  UE ho acc  | SUCI | No payload
gNB12 | AMF21 |  N2  |  Xn PDU id  | SUCI | No payload
gNB12 | gNB11 |  Xn  | Xn ho comp  | SUCI | No payload
AMF21 | gNB12 |  N2  |   AMF ack   | SUCI | Payload: 7878de6860a5998753bffed327
4b8b6b6c72e472084c6495e2a5c465fd9786e800000001
```

(a) XN Emulator Code Output

```
gNB11 | AMF21 |  N2  |  N2 ho req  | SUCI | Payload: 0c0101
AMF21 | gNB12 |  N2  | AMF ho req  | SUCI | Payload: 01d66b08d
2877f2aadeac8579e42ca26483ec9995586848f4dcdb38d552d5fe7c10000000107010000
0000
gNB12 | AMF21 |  N2  |  N2 ho ack  | SUCI | Payload: 070100000000
AMF21 | gNB11 |  N2  |  N2 ho cmd  | SUCI | Payload: 0c00000001070100000000
gNB11 |  UE01 | RRC  |  N2 ho cmd  | SUCI | Payload: 0c00000001070100000000
UE01  | gNB12 | RRC  | UE ho cfrm  | SUCI | No payload
gNB12 | AMF21 |  N2  | N2 ho comp  | SUCI | No payload
AMF21 | gNB11 |  N2  |gNB release  | SUCI | No payload
gNB11 | AMF21 |  N2  |gNB confirm  | SUCI | No payload
```

(b) N2 Emulator Code Output

Figure 9: XN and N2 Emulator Code Output

```
          gNB -----------------> AMF

     gNB_id: 12
      ME_id: 1
     AMF_id: 21
  New gNB_id: 13

AMF: received N2 handover request

          gNB <----------------- AMF

AMF: doing an N2 handover to gNB_id: 13 for ME_id: 1, AMF key is not re-keyed

New gNB: Received a handover request from AMF 21
AMF key is not re-keyed
    gNB: Received an NH, NCC pair from AMF
     NH: 278c83b5ac2fd923d8239819d7881374b6cc0a4ecfbc27fb2cc1f0509756e3ba
    NCC: 2
  K_GNB: 929dd25a322cb41d502ea4ac1622bafb1b2cb73ea95d02534c52196de72272cc


          gNB -----------------> AMF

     gNB_id: 13
     ME_id: 1

AMF: received handover acknowledgement from target gNB

          gNB <----------------- AMF

AMF: N2 handover command to gNB_id: 12 for ME_id: 1

Old gNB: Received a handover command from AMF 21

UE <----------------- gNB

     gNB_id: 12
  New gNB_id: 13


ME1 received handover data for gNB 13
AMF key is not re-keyed
```

Figure 10: A partial N2 handover (no re-keying) in Emulator Run