

Identity and Security in 5G Authentication

by

Julian Parkin

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2024

© Julian Parkin 2024

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

Chapters 2 and 3 of this thesis are based on works co-authored with my supervisor, Professor Mahesh Tripunitara, who helped draft the manuscripts for publication. I am the primary author of both works; the modelling and implementation described were done on my own.

The work upon which Chapter 2 is based is currently under review at the International Conference on Information and Systems Security (ICISS).

The work upon which Chapter 3 is based is currently under review at the Network and Distributed System Security Symposium (NDSS).

Abstract

In this thesis, we study the provision and protection of user identity in the 5G Authentication and Key Agreement (5G-AKA) protocol. We present two variations of the protocol: the first mitigates a family of de-anonymization attacks that aim to defeat the privacy-protection features of 5G-AKA. It does so by replacing a fixed user identity with a sequence of ephemeral identifiers. This variant is designed to be fully backwards compatible with the existing 5G-AKA authentication message formats, which allows it to be used in roaming scenarios without changes to the visited network. The second protocol is a realization of “Bring Your Own Identity” (BYOI) for 5G-AKA, allowing subscribers to authenticate with an identity provisioned by an external provider. This is accomplished by composing 5G-AKA with OAuth 2.0, a de-facto standard for third-party authorization online. We built and verified a formal model of each protocol using Tamarin, a theorem-prover tool for security protocols. From this, we note some limitations of existing formalizations of secrecy and authentication properties, and propose improvements. Finally, we present an implementation of our BYOI protocol over a simulated 5G system, and show it works against Google’s OAuth 2.0 API. We discuss some practical considerations arising from the implementation.

Acknowledgements

I would like to thank my supervisor, Professor Mahesh Tripunitara for his guidance and mentorship during my master's research.

I also thank my readers, Professors Florian Kerschbaum and Catherine Rosenberg, for their feedback and insights.

Furthermore, I thank the authors of the Tamarin verification tool, which was used to great extent in this thesis.

Thanks also to my employer, Altera (formerly Intel Programmable Solutions Group), for giving me time to pursue my master's degree.

Finally, thanks to my parents for their love and support during my educational journey.

Dedication

This thesis is dedicated to my parents, who worked to inspire my interest in science and engineering from a young age.

Table of Contents

Author's Declaration	ii
Statement of Contributions	iii
Abstract	iv
Acknowledgements	v
Dedication	vi
List of Figures	xi
List of Tables	xii
List of Abbreviations	xiii
List of Symbols	xv
1 Introduction	1
1.1 Outline	1
1.2 5G Authentication and Key Agreement	2
1.2.1 The 5G network	2
1.2.2 Preliminaries	3

1.2.3	Protocol Description	3
1.2.4	On Registration	6
1.3	Tamarin	7
1.3.1	Protocol Rules	7
1.3.2	Property Specification	8
1.3.3	Adversary Model	9
1.3.4	Proof Strategy	9
1.4	Formal Security Properties	10
1.4.1	Lowe’s Hierarchy	10
1.4.2	Agreement Properties in Tamarin	11
1.4.3	Secrecy Properties in Tamarin	12
2	Countering the “SUCI Catcher”	14
2.1	The “SUCI Catcher” Attack on 5G Privacy	14
2.2	Prior Approaches	15
2.2.1	Preventing an Attacker from Computing the SUCI	15
2.2.2	Including a Timestamp in the SUCI	17
2.2.3	Detecting Replays	17
2.2.4	Including a MAC in the SUCI	18
2.2.5	Linking Authentication to the SUCI	18
2.2.6	Authenticating Base Stations	18
2.2.7	Other Modifications	19
2.2.8	Conclusion	19
2.3	Our Protocol	19
2.3.1	Design Goals	19
2.3.2	Protocol Description	20
2.3.3	Effectiveness	22
2.4	Practical Considerations	23

2.4.1	Calculation and Storage of hSUPI	24
2.4.2	On Our Use of Repeated Hashing	24
2.4.3	SUCI Format	24
2.4.4	Hardware Implementation	26
2.5	Analysis	26
2.5.1	Modelling	27
2.5.2	Correctness	27
2.5.3	Effectiveness	30
3	“Bring Your Own Identity” for 5G	33
3.1	Related Work	34
3.2	OAuth 2.0 and OpenID Connect	35
3.2.1	OAuth 2.0	35
3.2.2	Security Enhancements	37
3.2.3	OpenID Connect	38
3.3	Composing 5G-AKA and OAuth 2.0	38
3.3.1	Our Protocol	38
3.3.2	Compatibility with 5G architecture	41
3.3.3	Desired Properties	42
3.4	Analysis	43
3.4.1	Model	44
3.4.2	Properties	44
3.4.3	Results	45
3.5	Nuances of Formal Verification	49
3.5.1	A Weakness of Weak Agreement	49
3.5.2	Non-Algorithmic Parties	50
3.5.3	Many-to-One Relationships	51
3.6	Prototype	52
3.6.1	Implementation	52
3.6.2	Takeaways	54

4	Conclusions	56
4.1	Concluding Remarks	56
4.2	Future Work	57
	References	58
	APPENDICES	65
A	Complete 5G-AKA key derivation	66
A.1	Security Anchor Function Key	66
A.2	Authentication Response	66
B	Source Code Releases	67
B.1	Tamarin Model of the SUCI-Catcher Mitigation	67
B.2	Implementation of “Bring Your Own Identity”	67
	Glossary	69

List of Figures

1.1	5G Network Model	2
1.2	5G-AKA Protocol	4
1.3	Example Tamarin rule	8
1.4	Example Tamarin lemma	8
1.5	Example non-injective agreement property	11
1.6	Example secrecy property	12
2.1	SUCI-Catcher Attack	16
2.2	5G SUCI format	25
2.3	Update Lemmas	28
2.4	Unchanged Lemmas	29
2.5	Agreement Lemma, and Associated Helper Lemmas	31
3.1	5G Network Model with Authorization Server	33
3.2	OAuth 2.0 Authorization Code Grant	36
3.3	5G-AKA composed with OAuth 2.0	39
3.4	Prototype implementation of our protocol	53

List of Tables

2.1	Modified 5G-AKA protocol (ref. Section 1.2.3 for original steps)	21
3.1	Secrecy Properties	46
3.2	Authentication properties between existing 5G-AKA parties	47
3.3	Authentication properties involving new parties	48
3.4	Choice of cryptographic primitives	55

List of Abbreviations

Notation	Description
3GPP	Third Generation Partnership Project 1 , 15 , 26
5G-AKA	5G Authentication and Key Agreement 1–3 , 7 , 10 , 17 , 18 , 20 , 22 , 24 , 26 , 29 , 32 , 34 , 40 , 41 , 43–46 , 52 , 56 , 57 , 66
AS	Authorization Server 33 , 35 , 37 , 38 , 40–43 , 46 , 49 , 50 , 52
AV	Authorization Vector 5 , 40
CSRF	Cross-Site Request Forgery 37
GUTI	Global Unique Temporary Identifier 7 , 14
HN	Home Network 2 , 3 , 5–7 , 15 , 17–20 , 22 , 24–27 , 29 , 32 , 33 , 38 , 40–46 , 49 , 51–54 , 56 , 57
HTTP	Hypertext Transfer Protocol 20 , 38 , 56
HTTPS	Hypertext Transfer Protocol Secure 44 , 45
IMSI	International Mobile Subscriber Identity 3 , 14 , 24 , 25
JSON	JavaScript Object Notation 20
KDF	Key Derivation Function 5 , 54 , 66
MAC	Message Authentication Code 5 , 18 , 22 , 54
MCC	Mobile Country Code 25 , 26

Notation	Description
MNC	Mobile Network Code 25 , 26
MSIN	Mobile Subscription Identification Number 25
NAI	Network Access Identifier 40
NAS	Non-Access Stratum 50 , 53
NITZ	Network Identity and Time Zone 17
NTP	Network Time Protocol 17
PKCE	Proof Key for Code Exchange 37 , 45
SN	Serving Network 2 , 3 , 5–7 , 15 , 19 , 20 , 22 , 23 , 26 , 32 , 38 , 40–46 , 52–54 , 56 , 66
SUCI	Subscription Concealed Identifier 3 , 5 , 6 , 14 , 15 , 17 , 18 , 20–26 , 30 , 40 , 68
SUPI	Subscription Permanent Identifier 3 , 6 , 14 , 20 , 21 , 40 , 42–45 , 52
TLS	Transport Layer Security 37 , 50
TS	Technical Specification 3
UE	User Equipment 2 , 3 , 5–7 , 14 , 15 , 18–20 , 22 , 25 , 29 , 34 , 35 , 38 , 40–46 , 50–54 , 57 , 66 , 67
URI	Uniform Resource Identifier 35 , 37 , 38 , 40 , 41
USIM	Universal Subscriber Identity Module 2 , 3 , 15 , 17 , 18 , 26 , 40 , 66

List of Symbols

Notation	Description
AUTN	5G-AKA authentication token 5 , 6 , 18 , 20 , 54
EK_{HN}	HN's extended key, used for encrypting auth_uri in 5G-AKA BYOI. 38 , 40 , 54
EK_{UE}	UE's extended key, used for encrypting redirect_uri in 5G-AKA BYOI. 38 , 41 , 54
hSUPI	Ephemeral identifier for mitigation of the SUCI-Catcher attack 20 , 22–27 , 29 , 30
HXRES*	5G-AKA hashed expected response 5 , 6 , 20
K_{SEAF}	5G Security Anchor Function key 5 , 6 , 38 , 43 , 46 , 66
K	5G-AKA long-term symmetric key 3 , 5 , 6 , 21 , 40 , 45 , 52 , 66
PK_{HN}	HN public key, for SUCI encryption 3 , 21
RAND	5G-AKA random challenge 5 , 6 , 20 , 21 , 66
RES*	5G-AKA reponse 6 , 20–22
SK_{HN}	HN private key, for SUCI decryption 5 , 45
SQN_{HN}	5G-AKA sequence number (HN perspective) 5 , 6 , 21
SQN_{UE}	5G-AKA sequence number (UE perspective) 6
SQN	5G-AKA sequence number (unspecified perspective) 3 , 5 , 6 , 24
XRES*	5G-AKA expected reponse 5 , 6 , 66

Chapter 1

Introduction

The mobile network is a global over-the-air network that provides communication and internet services to mobile devices. Access to the network generally requires a subscription with a network operator, who operates their own network infrastructure (or, in the case of a mobile virtual network operator, purchases access to the infrastructure of another operator). Mobile technology has seen widespread adoption: at the end of 2023, there were 8.5 billion subscriptions [36]. The subscriber-operator relationship naturally requires a method to identify and authenticate subscribers. This thesis consists of two projects that explore improvements to subscriber authentication in the latest, fifth generation (5G), mobile network standards. The 5G standards are developed by the [Third Generation Partnership Project \(3GPP\)](#), who publishes a set of technical specifications describing all aspects of the network. Among the many changes made in 5G were improvements to authentication and security. This thesis will focus on [5G Authentication and Key Agreement \(5G-AKA\)](#), one of the two standardized primary authentication methods for public networks.

1.1 Outline

This chapter first provides an overview of [5G-AKA](#) in Section 1.2 as a common starting point for the modifications discussed in the following chapters. Section 1.3 describes the [Tamarin](#) theorem prover, used extensively in this thesis for protocol analysis. Finally, Section 1.3 describes how desired security properties are encoded in Tamarin.

Chapter 2 discusses an attack on 5G privacy known as the “SUCI catcher,” and develops a protocol modification to defeat it.

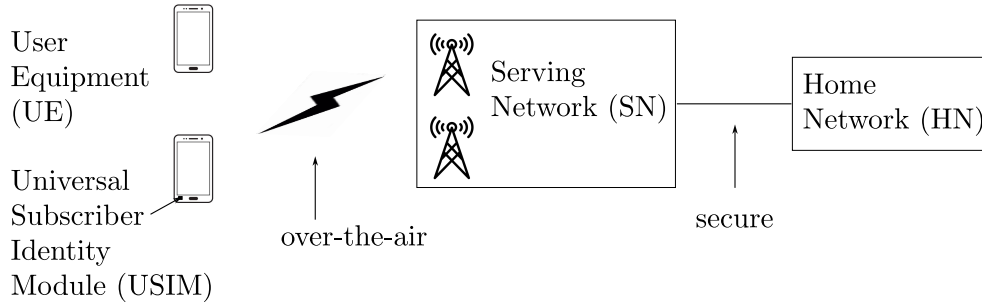


Figure 1.1: 5G Network Model

Chapter 3 develops a method for “bring your own identity” in 5G, where the user’s identity is managed by a third party, by composing 5G-AKA with OpenID Connect, a common authentication protocol used for web applications.

Finally, Chapter 4 provides some concluding remarks and direction for future work.

1.2 5G Authentication and Key Agreement

This section describes the [5G-AKA](#) protocol, which aims to provide authentication between a subscriber and the mobile network. The protocol also allows both parties to derive a session key, used for deriving further keys that protect 5G application traffic.

1.2.1 The 5G network

The 5G network employs a service-based architecture, composed of a large number of entities with narrowly-specified functionality [13]. For the purposes of this thesis, we follow the convention developed by Basin et al. in [21], and consider the abstract network model shown in Figure 1.1. The 5G network functions are grouped into two entities: a [Serving Network \(SN\)](#), who provides network access, and a [Home Network \(HN\)](#) who handles subscriptions. Often, these two entities are owned by the same network operator, but this model provides sufficient fidelity to consider the case of roaming, where the subscriber access the network through an operator different than the one to which it has subscribed. The subscriber’s device is called the [User Equipment \(UE\)](#), and the contains a [Universal Subscriber Identity Module \(USIM\)](#), which stores sensitive authentication data.

The UE and SN communicate over radio, so any authentication protocol must assume that an attacker can intercept messages or generate its own. By contrast, the SN and HN are assumed to have established secure communications through some method outside the context of subscriber authentication. Any roaming services require a pre-existing business relationship between the two network operators.

1.2.2 Preliminaries

5G-AKA is a challenge-response protocol: the UE provides a response to a random challenge. Authentication is based on a symmetric key, K , shared between the UE and HN. Both UE and HN also maintain a sequence number, SQN , used for replay protection: the UE's sequence number may only advance. On the UE side, both K and SQN are stored in the USIM.

In 5G, the UE is identified by a **Subscription Permanent Identifier (SUPI)**. Several types of SUPI are specified [2], but in public networks, it takes the form of the **International Mobile Subscriber Identity (IMSI)**, a permanent identifier provided by the USIM [26]. To protect user privacy, the SUPI is encrypted to form a **Subscription Concealed Identifier (SUCI)** before being sent over radio [11].

1.2.3 Protocol Description

Figure 1.2 shows the protocol sequence of 5G-AKA, described by 3GPP **Technical Specification (TS) 33.501** [11]. Authentication is initiated from the network, so the sequence first shows the UE attempting to register with the network by sending a registration request. The protocol steps in the figure are detailed in the following paragraphs.

Encrypt SUPI The Subscriber encrypts its SUPI using the HN's public key:

$$\boxed{\text{SUCI} \leftarrow \text{ECIES-ENC}(PK_{\text{HN}}, \text{SUPI})}$$

The ECIES-ENC function represents the Elliptic-Curve Integrated Encryption Scheme [25]: the currently standardized method for encrypting SUPIs [11]. Although not shown here, the SUCI also contains plaintext elements identifying the HN so that the SN may identify the HN responsible for authentication [2].

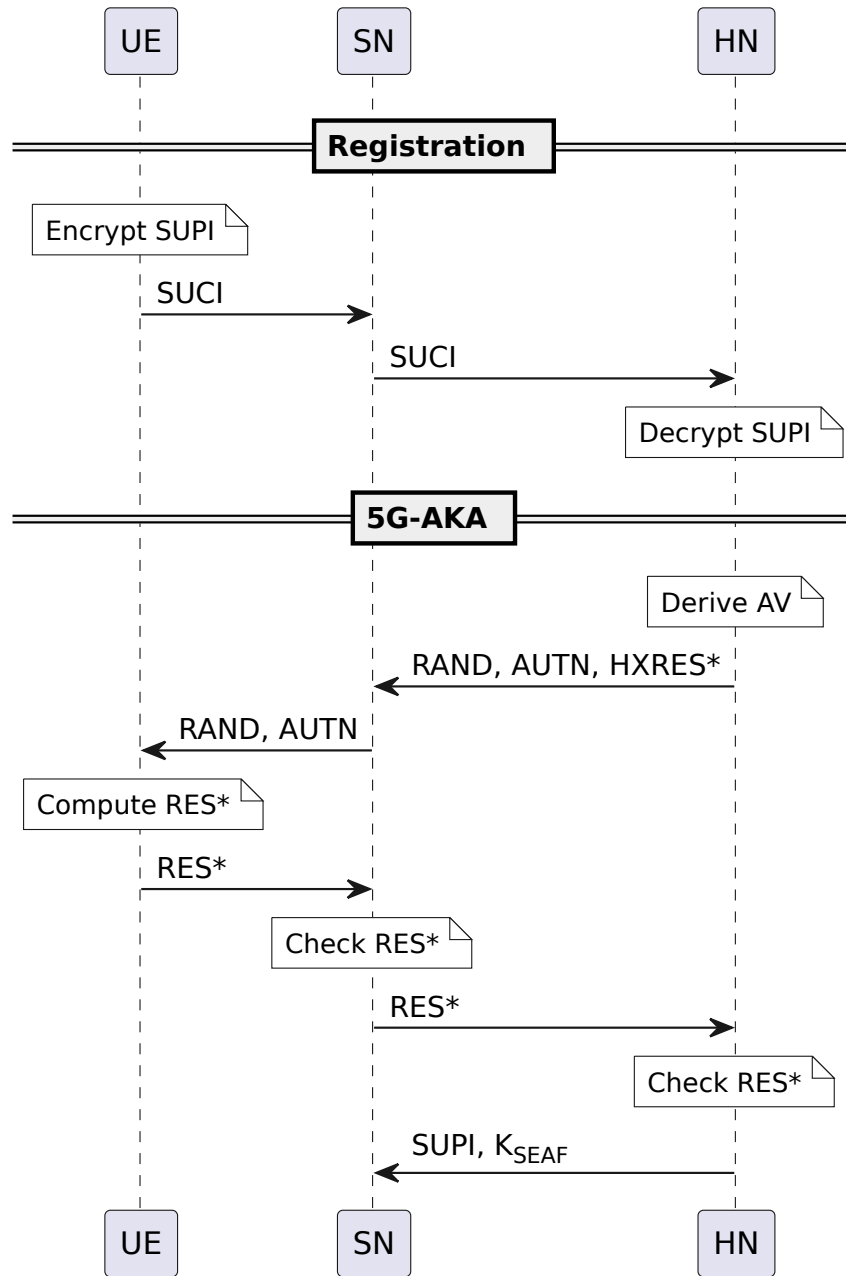


Figure 1.2: 5G-AKA Protocol

Decrypt SUPI To initiate authentication, the SN sends a request to the HN containing the SUCI. The HN is able to decrypt the SUCI using its private key and look up the subscriber in its database to retrieve the corresponding K .

$$\begin{aligned} \text{SUPI} &\leftarrow \text{ECIES-DEC}(\textcolor{blue}{SK}_{\text{HN}}, \text{SUCI}) \\ K &\leftarrow \text{lookup-subscription}(\text{SUPI}) \end{aligned}$$

Derive AV The HN advances its SQN and derives an [Authorization Vector \(AV\)](#) according to the following steps.

$$\begin{aligned} \textcolor{blue}{\text{RAND}} &\leftarrow \text{random}() \\ \text{MAC} &\leftarrow f_1(K, \textcolor{blue}{\text{SQN}}_{\text{HN}} \parallel \text{RAND}) \\ \text{AK} &\leftarrow f_5(K, \text{RAND}) \\ \text{SQN}_{\text{ENC}} &\leftarrow \text{SQN}_{\text{HN}} \oplus \text{AK} \\ \textcolor{blue}{\text{AUTN}} &\leftarrow \text{SQN}_{\text{ENC}} \parallel \text{MAC} \\ \textcolor{blue}{\text{XRES}}^* &\leftarrow f_2(K, \text{RAND}) \\ \textcolor{blue}{\text{HXRES}}^* &\leftarrow \text{SHA256}(\textcolor{blue}{\text{XRES}}^* \parallel \text{RAND}) \\ \textcolor{blue}{K}_{\text{SEAF}} &\leftarrow \text{KDF}(K, \text{RAND} \parallel \text{SQN}_{\text{ENC}} \parallel \text{SNname}) \end{aligned}$$

Functions f_1 and f_2 are [Message Authentication Codes \(MACs\)](#), and f_5 is a [Key Derivation Function \(KDF\)](#)—the generic f symbols are used in the standards [6]. XRES^* is the expected response to the authentication challenge, which is hashed to form HXRES^* . The Security Anchor Function key, K_{SEAF} , is a per-session key derived from the authentication process. The derivation steps for K_{SEAF} and XRES^* have been simplified for the purpose of illustration. A complete version of the derivation steps can be found in [Appendix A](#). RAND , AUTN , and HXRES^* are sent to the SN, who forwards RAND and AUTN to the UE as an authentication request.

Compute RES* The UE computes an authentication response.

$$\begin{aligned}
& \text{AK} \leftarrow f_5(K, \text{RAND}) \\
& \text{SQN}_{\text{ENC}} \parallel \text{MAC} \leftarrow \text{AUTN} \\
& \text{SQN}_{\text{HN}} \leftarrow \text{SQN}_{\text{ENC}} \oplus \text{AK} \\
& \text{XMAC} \leftarrow f_1(K, \text{SQN}_{\text{HN}} \parallel \text{RAND}) \\
& \text{RES}^* \leftarrow f_2(K, \text{RAND}) \\
& K_{\text{SEAF}} \leftarrow \text{KDF}(K, \text{RAND} \parallel \text{SQN}_{\text{ENC}} \parallel \text{SNname})
\end{aligned}$$

Two conditions are required for the UE to accept the authentication:

1. XMAC matches MAC
2. SQN_{HN} is fresh

The first condition ensures the authentication vector has been generated by the HN. The second serves to prevent replays: in practice it is equivalent to SQN_{HN} being larger than the UE's SQN, SQN_{UE} , although there are some additional checks required [6]. After accepting the authentication, the UE advances SQN_{UE} to match SQN_{HN} .

If the UE accepts the authentication, it will provide an authentication response containing RES^* . If not, it will provide an error response instead, which is not shown in the protocol diagram.

Check RES^* (SN) Assuming the UE provides an authentication response, the SN is able to verify its correctness by hashing it in the same way that XRES^* is hashed to form HXRES^* . This allows the SN to independently check the authentication result without exposing XRES^* outside of the HN.

Check RES^* (HN) The HN compares RES^* directly to XRES^* . If the response was correct, the authentication was successful, and it will forward K_{SEAF} to the SN for use in further protocol steps. The HN also provides the SN with the decrypted SUPI.

1.2.4 On Registration

Figure 1.2 showed the straightforward case where the UE uses the SUCI in the registration request, so the SN initiates authentication in order to learn the SUPI. In practice, the

two steps are not always linked. Once the UE has authenticated, it is provisioned with a [Global Unique Temporary Identifier \(GUTI\)](#), which may be used in subsequent registration requests [10]. Registration may complete without authentication if the authentication context associated with the GUTI can be transferred to the SN. Whether authentication is required in this case is up to HN policy [11]. Equally, the SN may initiate authentication when the UE is already registered, since the authentication procedure is network initiated. The lack of a one-to-one relationship between registration and authentication is considered when developing modifications to 5G-AKA in the following chapters.

1.3 Tamarin

[Tamarin](#) [20] is a theorem prover tool designed for the analysis of cryptographic properties. It operates in the symbolic model (originally due to Dolev and Yao [33]), meaning that all cryptographic primitives are modelled as black-box functions with defined properties, and there is an insecure channel modelled by passing messages through an attacker. This section provides a summary of the main features of Tamarin used in this thesis; for further detail about its implementation and operation, we refer the reader to other work [50, 51].

1.3.1 Protocol Rules

Tamarin models a protocol as a group of multiset-rewriting rules [51]: the state of the system is a multiset of facts, and each rule describes a substitution of one pattern of facts with another. The hypothetical example shown in Figure 1.3 models a party who receives a message in the clear and sends its encryption under some symmetric-key scheme: the rule consumes a message, m , in the form of an `In( $m$ )` fact, and outputs the encrypted message `senc( $m$ ,  $\tilde{k}$ )` along with some identifier $\tilde{id}$ inside an `Out` fact. While almost all facts are created and used only within the protocol specification, the `In` and `Out` facts are special and model messages passed through an untrusted network. The `senc` function is available from a built-in theory of symmetric encryption, but custom functions and associated equations may also be defined, as long as the equations satisfy certain restrictions.

The `!User( $\tilde{id}$ ,  $\tilde{k}$ )` fact in the same example provides the relation between the key and the identifier, and is not consumed by the rewriting: the `!` prefix identifies a [persistent fact](#). It would generally be created by another rule that initializes the user by creating the key and identifier.

```

rule encrypt_message:
  [ !User(~id, ~k)
    , In(m)
  ]
  --[ Encrypt(~id, m) ]->
  [ Out(<~id, senc(m, ~k)>) ]

```

Figure 1.3: Example Tamarin rule

1.3.2 Property Specification

Tamarin deals with theorems (referred to as [lemmas](#)) that claim properties of [traces](#), which are legal sequences of applications of rewriting rules [\[66\]](#). In order to reference rules, and any relevant terms to which they are applied, from lemmas, rules must be annotated using [action facts](#), which can be viewed as defining specific events. One example is the `Encrypt(~id, m)` fact shown in [Figure 1.3](#).

Tamarin lemmas are statements involving action facts and time points, expressed in first-order logic. The time points, notated with a `#` prefix, are variables representing the times at which the action facts occur. Ordering between time points can be expressed using inequality symbols: `#j < #i` states that time *j* occurs before time *i*.

```

lemma user_encrypted:
  " All id m #i. Encrypt(id, m)@i
    ==> (Ex #j. CreateUser(id)@j & #j < #i) "

```

Figure 1.4: Example Tamarin lemma

For example, [Figure 1.4](#) shows a lemma that claims whenever the `Encrypt` action occurs, a `CreateUser` action for the same identity has occurred at an earlier time.

The combination of rules and lemmas allows Tamarin to consider the full range of protocol behavior: since any rule with satisfied prerequisites can be applied, either side may start a new session instead of continuing the existing one, and messages may be modified by an adversary or be held back and replayed later.

1.3.3 Adversary Model

Tamarin includes a model of an adversary that can view, modify, and send messages over the untrusted network modelled through the special `In` and `Out` facts [64]. Other potential forms of compromise have to be modelled explicitly; for example, a protocol using public-key cryptography may have a rule allowing the adversary to obtain private keys. It is preferable to model the worst-case adversary capability in the protocol, and then disallow certain actions in each lemma, as this ensures properties are proven under a minimal and explicit set of assumptions (e.g. which, and whose, keys are actually secret).

Most multi-user protocols are expected to be robust against malicious users. Modelling of malicious or compromised users in Tamarin can be done using two possible methods. First, if the model allows the adversary to obtain the keys, and other private data, of any untrusted party, then the full range of malicious behavior is naturally included, as the attacker can simulate any action on behalf of that party. Second, adversary actions can be explicitly allowed using additional rules. This requires more thought to ensure all possible malicious behaviors are included, but may be necessary for elements of the model that simulate abstract components, such as an ideal secure channel.

1.3.4 Proof Strategy

Theorem proving in Tamarin is accomplished by searching for a trace satisfying desired properties [51]. When the prover is given a universally quantified formula, it will attempt to search a trace satisfying its negation, and conclude the theorem is true if no such trace exists. While automated search heuristics are available in the software, repeatedly making a suboptimal choice of which fact to consider next can easily lead to a combinatorial explosion of cases, preventing the solver from reaching a conclusion within a feasible amount of time and effort, as it must pursue every case until contradiction in order to conclude no counterexample exists. To resolve this, Tamarin allows the user to provide an [oracle](#): an external program that, when queried with a list of goals, outputs a ranking describing the order in which they should be pursued. (Recent versions of Tamarin also provide a method for specifying an oracle in the same file as the model.)

When Tamarin first loads a protocol specification, it performs a precomputation to enumerate all possible sources for each fact [65]. If it is not able to resolve all sources of some fact, this is called a [partial deconstruction](#). Since these partial deconstructions can interfere with the ability to prove lemmas, Tamarin allows the user to write [sources lemmas](#) that are assumed during a second round of precomputation that aims to resolve the partial

deconstructions. The sources lemmas must still themselves be proven without the benefit of the second precomputation stage, in order to avoid introducing incorrect assumptions into the proofs.

It is also possible to mark lemmas for “reuse,” which allows them to be assumed when proving other lemmas [66]. To avoid introducing circular arguments, each reuse lemma can only be used to prove lemmas following it in the input file. Allowing the reuse of a lemma provides a useful way to encapsulate repeated arguments, similar to how lemmas are used in hand-written proofs. This is especially important when the lemma requires induction to prove, as such a property is not visible to the search process without its delineation as a lemma with separate proof.

1.4 Formal Security Properties

To analyze security protocols using [Tamarin](#), one must express desired properties in formal logic. This section explains the definitions of authentication and secrecy employed for this purpose in this thesis.

1.4.1 Lowe’s Hierarchy

The Tamarin manual [66] recommends using definitions based on Lowe’s hierarchy of authentication specifications [47]. This strategy is also employed by Basin et al. in their analysis of [5G-AKA](#) using Tamarin [21].

Lowe’s work defines a series of authentication properties of varying strength [47]. Specifically, if a initiator A has completed a run of the protocol, which it believe to be with a responder B, then

- [Aliveness](#) requires that B had been running the protocol.
- [Weak agreement](#) adds the requirement that B believed it was running with A (but does not require that this run be unique).
- [Non-injective agreement](#) adds the requirement that B ran as responder, and that A and B agree on the values of a set of data items.
- [Injective agreement](#) adds the requirement that each such run by A corresponds to a unique run by B.

The Tamarin formalization treats each data item separately, so all three of the agreement properties are useful: weak agreement allows considering authentication separately from any data items, non-injective agreement is required when a data item is not unique per session, and injective agreement is the strongest property where it can be proved.

1.4.2 Agreement Properties in Tamarin

The formalization employed by [21] is a generic style of property specification, meaning that the same action facts are used in each lemma. Figure 1.5 shows an example non-injective agreement property. The property has several components. First, the **Commit** action fact in the antecedent of the implication represents the initiator completing an execution of the protocol. The fact records the identities of the initiator and assumed responder and the value of the agreed data item. It would be placed in the rule representing the final protocol step from the initiator perspective. Tuples (defined in angle brackets) are used to tag the fact with the names of the roles and data item (terms in single quotes are constants), as the same fact is reused for other agreement properties.

```
lemma non_injective_agreement:
  " All A B data #i.
    Commit(A, B, <'RoleA', 'RoleB', <'data', data>>)@i
    ==> (Ex #j.
      Running(B, A, <'RoleA', 'RoleB',
        <'data', data>>)@j)
    | (Ex X k #j.
      Rev(X, <'key', k>)@j & Honest(X)@i) "
```

Figure 1.5: Example non-injective agreement property

Next, the first alternative in the consequent of the implication, the existence of a **Running** fact, represents that the responder has been running the protocol, with the same initiator and data. This fact would be placed in the rule representing the final protocol step run by the responder that is observable by the initiator, consistent with the explanation of Lowe: “We take [B has previously been running the protocol] to mean that B has progressed at least as far as the last message that B sends; clearly A can never be assured that B received any subsequent messages” [47].

The second alternative in the consequent represents a security assumption. It requires two things. First, that any rule allowing the adversary to learn secret data includes a **Rev**

action fact. This fact records the type of data revealed, and the identity of the party to which it belongs. Second, that the rule that contains the **Commit** fact is annotated with **Honest** action facts recording the identities of parties assumed honest (i.e. that are trusted) in that protocol session by the initiator. If these rules are followed, the alternative states generically that some honest party has its key compromised.

If the lemma is proven, then, it guarantees that non-injective agreement is achieved for this combination of parties and data item, or that a party trusted by the initiator had its key compromised. Equivalently, if no trusted party had its key compromised, then non-injective agreement is achieved. Additional security assumptions may be modelled using further alternatives.

This example showed how non-injective agreement is modelled in Tamarin. Weak agreement uses the same form of [lemma](#), but without referencing a specific data item. The injective agreement lemma developed by [\[21\]](#) adds, to the non-injective agreement specification, the requirement that there are no other **Commit** facts with the same data item value. This ensures that completed sessions are unique from the initiator perspective.

1.4.3 Secrecy Properties in Tamarin

The form of secrecy property used in this thesis is shown in [Figure 1.6](#). It states that a data item assumed secret should not be known to the adversary (the **K** action fact represents adversary knowledge), subject to security assumptions expressed in the same way as in the agreement properties.

```
lemma secrecy:
  " All A data #i. Secret(A, <'data', data>)
    ==> not (Ex #j. K(data)@j)
      | (Ex X k #j.
          Rev(X, <'key', k>)@j & Honest(X)@i) "
```

Figure 1.6: Example secrecy property

This definition is intuitive, but unlike with **Commit** and **Running**, it is not obvious where to place the **Secret** action fact. As pointed out by Cremers and Mauw, such claims are still local assertions about the global state of the protocol [\[29\]](#). They provide an example where secrecy even depends on perspective: a party who sends an encrypted message using public-key cryptography can usually claim secrecy because the message is only read by a

trusted party; however, a party who receives an encrypted message cannot claim secrecy because the message may have been sent by anyone, including the attacker, in which case the attacker would know the content of the message.

This suggests a method for determining where to place the secrecy claim is required. We will develop such a method in Section [3.4.2](#).

Chapter 2

Countering the “SUCI Catcher”

This chapter considers an attack on 5G privacy and a corresponding mitigation. The attack is described in Section 2.1. Section 2.2 explains some prior approaches to mitigation, from which we distinguish our approach. Our modified protocol is presented in Section 2.3. After the presentation, we discuss some additional considerations in Section 2.4 and explain how we formally verified the protocol in Section 2.5.

2.1 The “SUCI Catcher” Attack on 5G Privacy

In prior mobile standards, the user identifier was not encrypted, allowing an attacker to track users by recording their identity [26]. A device designed to perform this attack by acting as a fake base station is called an IMSI catcher [56]. When the UE establishes a radio link with the fake base station, it will attempt to register, allowing the IMSI catcher to send an identity request. This request serves two purposes: (i) improving the efficiency of the attack by avoiding the need to wait for a naturally occurring registration request, and (ii) bypassing the use of temporary identifiers as an attack mitigation. (3G had a temporary identifier called the TMSI [17], which operated similarly to the 5G GUTI.)

As previously mentioned, 5G improves subscriber privacy by encrypting the SUPI to form the SUCI, which prevents IMSI-catcher-like attacks from directly obtaining the permanent identifier. Recently, however, Chlosta et al. develop an attack that reveals some information about a SUCI—specifically, whether it represents the same subscriber as another SUCI. To do so, they use the registration and authentication request as an oracle for equivalence of SUCIs. The procedure for this attack is shown in Figure 2.1, under

the section labeled “SUCI probe.” The Attacker is assumed to have man-in-the-middle capabilities over the radio link, by combining the fake base station of an IMSI Catcher with a method to present itself as a UE to a legitimate base station. In the figure, the SN and HN are grouped as one entity since the internal details of 5G-AKA are not relevant to the attack.

Upon receiving a registration request from an unknown UE with some identity $SUCI_A$, the attacker sends a registration request to the real SN using a second identity $SUCI_B$. In response, the SN will initiate authentication with the HN and forward an authentication vector intended for the UE identified by $SUCI_B$. The attacker then sends this authentication vector to the unknown UE. If the UE accepts the authentication, then it must be the same UE identified by $SUCI_B$.

This new SUCI-Catcher attack requires knowing a SUCI that corresponds to a particular subscriber of interest (i.e. the one sent as $SUCI_B$) in order to be useful. [26] presents two ways to achieve this:

1. Generating a SUCI by encrypting a known SUPI using the public key of the HN.
2. Linking a subscriber to a captured SUCI using other surveillance mechanisms (e.g. security cameras).

These can be viewed as two variants of the same attack with differing difficulty of mitigation. Both methods are shown in Figure 2.1, but it should be noted that an arbitrary amount of time may pass before the SUCI probe is performed.

2.2 Prior Approaches

As motivation for our design choices, we first consider existing proposals for mitigating the SUCI-Catcher attack from prior work.

2.2.1 Preventing an Attacker from Computing the SUCI

Work from the 3GPP [7] suggests that variant (1) of the SUCI-catcher attack is not a problem in practice, because the public key of the HN is hidden in the USIM, and the standards allow for an HN to require that the USIM to compute the SUCI.

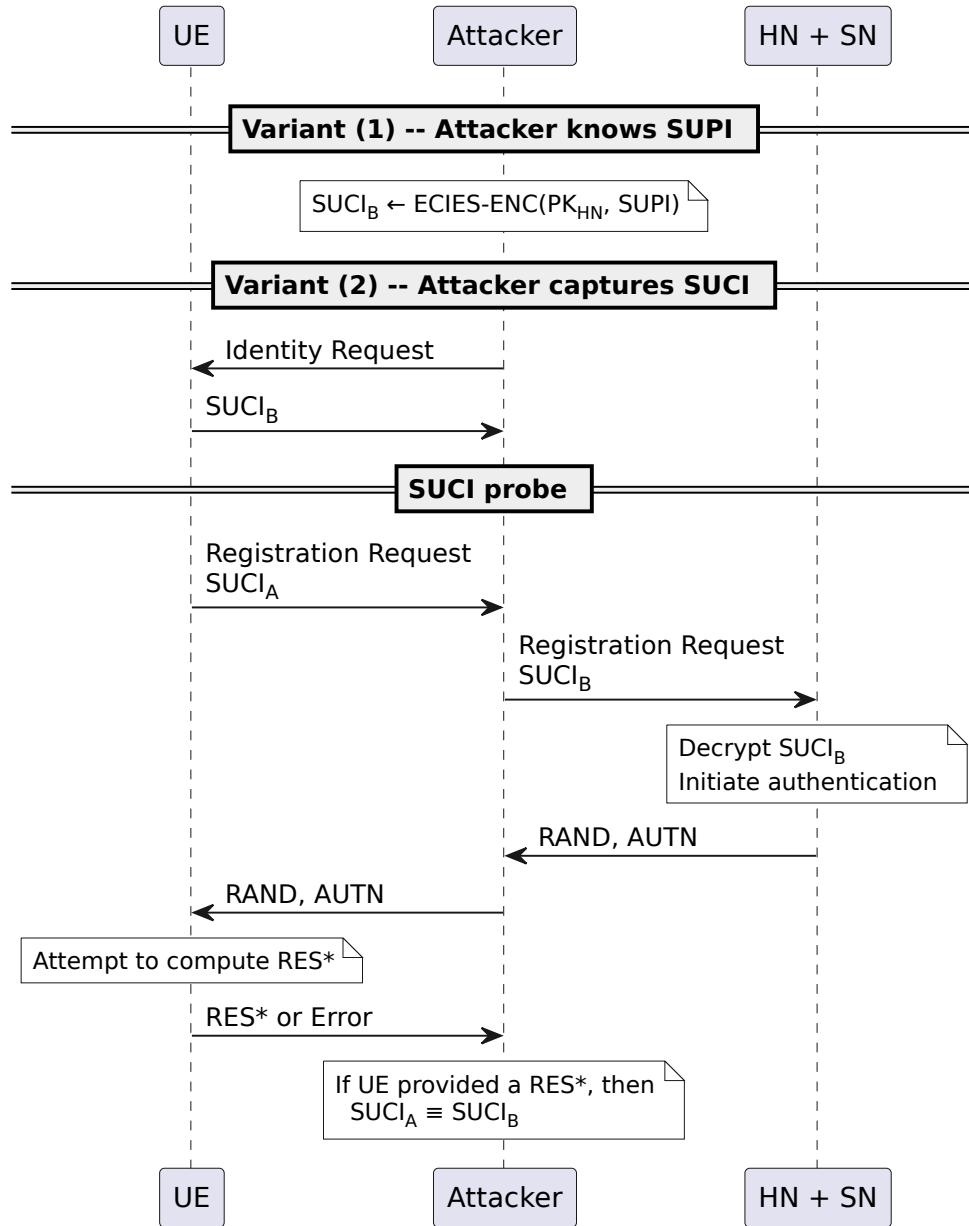


Figure 2.1: SUCI-Catcher Attack

We believe this approach would be difficult to implement as current cryptographic practice typically assumes that public keys are not sensitive, so the HN would have to carefully re-examine all its protocols and procedures that deal with its public key.

Additionally, this places higher weight on the tamper resistance of the USIM compared to only the protection of the 5G-AKA secret key: every USIM produced for the HN would be provisioned with the same public key, so it must be assumed that the attacker would obtain a USIM and attempt to extract the key.

2.2.2 Including a Timestamp in the SUCI

Another suggestion of [7], to include a timestamp in the SUCI, appears to be effective at preventing variant (2) of the attack as the HN simply has to reject registration requests with SUCIs created outside a certain window around the current time. This method has a flaw: current cellular devices typically synchronize to the current time using either [Network Identity and Time Zone \(NITZ\)](#) [9], where time information is sent directly by the network, or [Network Time Protocol \(NTP\)](#) [49], where time information is requested once an connection has been established to the Internet (for example, current Android documentation lists NITZ and NTP as the only sources enabled by default [16]). In either case, the authenticity of the time information received by the device is only assured after authentication [55], which creates a circular dependency if accurate timestamps are needed by the authentication process.

2.2.3 Detecting Replays

Detecting replays through repeated messages is a common strategy for preventing protocol misuse. In the context of 5G-AKA, [7] and [26] suggest storing previously used SUCIs to detect reuse. This does not provide a provable security enhancement compared to the base protocol, since the HN must store a finite number of SUCIs, or store them for a finite time, especially since a promise to store SUCIs is an obvious target for a Denial-of-Service attack. Given enough time or repeated authentications, a captured SUCI will become replayable, so this mitigation requires a heuristic assumption of the time between interactions with the SUCI catcher in practical attack scenarios. Additionally, replay prevention is not equivalent to preventing variant (2) of the SUCI catcher, due to the possibility of using a captured SUCI that had not previously been sent to the network for the SUCI probe.

2.2.4 Including a MAC in the SUCI

To ensure that the SUCI has been computed by the UE without assuming that the HN’s public key is secret, the SUCI computation can be changed so that it can only be performed with knowledge of the 5G-AKA key. The proposed realization of this idea in [7] uses a key derived from the 5G-AKA key to compute a MAC over the SUCI. This successfully prevents variant (1) of the SUCI-catcher attack, with the only disadvantage being that this modified procedure must be implemented in the USIM. This does not address variant (2).

2.2.5 Linking Authentication to the SUCI

An approach that prevents both variants of the SUCI-catcher attack is to link the registration to the authentication, so that they form one protocol. In [7], this is done by including a random nonce in the SUCI, which is included in the data over which the MAC in AUTN is computed. However, as mentioned in Section 1.2.4, registration and authentication are not linked in the current 5G protocols, and there are instances where they need to be done separately. Thus, any modification to 5G-AKA cannot “piggyback” additional authentication material inside the SUCI unless it provides for a separate authentication procedure when there is not a preceding registration request, leading to a protocol for which it is difficult to prove correctness because of the many possible interactions between authentication and other procedures.

2.2.6 Authenticating Base Stations

An alternative approach that focuses on preventing the man-in-the-middle attack on the network rather than modifying the SUCI or authentication is to authenticate the base stations. The use of a public-key system to establish secure communication with network intermediaries has been proposed in some prior work [7, 12, 17]. This approach succeeds in preventing the SUCI-catcher, and related attacks, by preventing an adversary from setting up a base station. However, it requires the adoption of a public-key infrastructure to distribute keys to entities approved by the HN—a substantial endeavor that represents a major new effort in terms of standardization, implementation, deployment, and operation.

2.2.7 Other Modifications

We use this category as a placeholder for various design choices that are possible if the registration and authentication processes are altogether replaced with a new protocol. For example, a replacement may be a protocol based on public-key cryptography rather than a shared symmetric key, in order to achieve perfect forward secrecy. A discussion of potential design choices in a protocol designed “from scratch” is ultimately open ended, and such a protocol is likely to require major changes to existing implementations and other parts of the 5G system.

2.2.8 Conclusion

Most of these proposals only address variant (1) (Sections 2.2.1 and 2.2.4) or variant (2) (Sections 2.2.2 and 2.2.3) separately, and have other disadvantages that were noted. Solutions that address the full attack (Sections 2.2.5, 2.2.6, and 2.2.7) require significant changes to network operation. By contrast, the protocol we present in the next section achieves backwards compatibility with existing SN implementations, avoiding changes to most network entities.

2.3 Our Protocol

We now present our proposed approach to mitigation: first, in Section 2.3.1 we summarize our design goals. Our protocol description follows in Section 2.3.2. After, we discuss, in Section 2.3.3, its effectiveness against the SUCI-catcher attack.

2.3.1 Design Goals

Our primary goal is of course to prevent the SUCI-catcher attack. Specifically, variant (1) should be entirely prevented, while variant (2) should be defeated under a clear set of assumptions on network behavior. Given also our discussion from the previous section, we adopt the following design constraints:

1. The HN’s public key may be public.
2. The UE does not need a reliable time source.

3. There is no one-to-one relationship between registration and authentication.
4. The SN may participate in the protocol without any change to its software.

The first three of these constraints avoid specific disadvantages from the prior approach we discuss in the previous section. We call (4) above backwards compatibility. This definition of backwards compatibility is useful because it ensures the possibility of gradual deployment: an HN may deploy the modified protocol without requiring that all their roaming partners, who may act as SN, support it.

To achieve backwards compatibility, the message formats should be such that an existing SN would process them without error. While the packet format used for messages exchanged between the UE and SN includes a length field for RAND and RES* [1], it is not clear whether the SN would forward fields with abnormal length: the communications between the SN and HN use a defined HTTP interface returning data in JSON format [3], so the SN is likely to validate individual fields when translating between JSON and message format used with the UE. The SN will also continue to hash RES* and compare it to HXRES*. For these reasons, we assume the lengths of AUTN, RAND, and RES* are fixed, and that the HXRES* check must operate unchanged.

2.3.2 Protocol Description

We propose the use of an ephemeral identifier in place of the SUPI in 5G-AKA. As a motivating example, consider a case where there is only one valid temporary identifier at a time, and both the UE and HN move to a new identifier after each authentication. This serves to defeat variant (1): assuming the ephemeral identifier is secret, an attacker would not be able to produce their own SUCIs. It also mitigates variant (2) since a captured SUCIs would become invalid after any authentication, as it would contain an old identifier.

However, this hypothetical system is impractical as it is not possible for both the UE and HN to agree on a coordinated time to move to the next identifier (this is the famous two generals’ problem [39]). Consequently, we relax the requirement and have both entities maintain a window of two valid identifiers at a time: the “current” identifier and the “next” identifier. For concreteness, we consider the case where the each identifier, called an hSUPI for “hash SUPI,” is the hash of the previous one—corresponding to the update rule $\text{hSUPI}_{n+1} = H(\text{hSUPI}_n)$. (See, however, Section 2.4.2 for discussion on the specificity of this approach.)

Table 2.1 shows the changes required in our modified protocol. The UE will use hSUPI instead of SUPI if it needs to produce a SUCI. This means that the HN must be able to

Step	Change
Encrypt SUPI	Use hSUPI as the SUPI, $\text{SUCI} \leftarrow \text{ECIES-ENC}(PK_{\text{HN}}, \text{hSUPI}_{\text{UE},n})$
Derive AV	Include hSUPI in the MAC, $\text{MAC} \leftarrow f_1(K, SQN_{\text{HN}}, \text{RAND}, \text{hSUPI}_{\text{HN},n+1})$
Compute RES*	1. Accept either of two possible MAC tags, $\text{XMAC}_1 \leftarrow f_1(K, SQN_{\text{HN}}, \text{RAND}, \text{hSUPI}_{\text{UE},n})$ $\text{XMAC}_2 \leftarrow f_1(K, SQN_{\text{HN}}, \text{RAND}, \text{hSUPI}_{\text{UE},n+1})$ 2. If the Authentication Vector is valid and MAC matched XMAC ₂ , advance hSUPI, $\text{hSUPI}_{\text{UE},n} \leftarrow \text{hSUPI}_{\text{UE},n+1}$
Check RES*	If RES* is accepted, advance hSUPI, $\text{hSUPI}_{\text{HN},n} \leftarrow \text{hSUPI}_{\text{HN},n+1}$

Table 2.1: Modified 5G-AKA protocol (ref. Section 1.2.3 for original steps)

look up subscribers by their two possible valid hSUPI values. To coordinate advancing the hSUPI, the HN includes its view of the next hSUPI in the [MAC](#) computed as part of the authentication vector. The UE will advance its hSUPI to ensure it is one step ahead of the HN before sending RES*. The HN can then update its own hSUPI upon receiving RES*.

2.3.3 Effectiveness

Do our changes provide a useful security guarantee in practice? We argue that they do: first, variant (1) is prevented since the attacker does not know the ephemeral identifier. Second, the update rule appears to mitigate variant (2) as any captured SUCI will become invalid after at most two authentications. In this section, we consider the remaining attack surface for the second variant and show that it is minimized and can be further mitigated.

The most obvious class of attacks involve causing the authentication to fail at some stage, so that the hSUPIs do not update. This represents a weaker form of SUCI catcher since the catcher may only prevent authentication if the subscriber continues to be within the range of its fake base station, whereas the greatest threat to privacy from the attack on the existing 5G-AKA is the ability to identify a subscriber who returns to the area covered by the SUCI catcher after an arbitrary amount of time has passed. Additionally, such attacks manifest as a denial of service, which is noticeable to the subscriber, who is unable to use their cellular service, and is more easily detected and mitigated. Chlosta et al. report that the device they tested cancels a registration attempt after two successive authentication failures [26]. This type of behavior defeats attacks requiring authentication to continuously fail.

Another possible attack would be for the attacker to force the UE one step ahead, so that the next SUCI it uses is replayable until the following authentication—we first explicate what such an attack would look like:

1. The UE attempts to register with the attacker (SUCI catcher).
2. The attacker forwards messages between UE and SN as normal but modifies RES* to trigger a failure. This leaves the UE's hSUPI advanced by one step relative to that of the HN.
3. The attacker uses an identity request to capture a SUCI computed from the UE's current hSUPI and drops the connection.
4. The UE eventually connects to the legitimate network and authenticates, but its hSUPI does not advance because it is already ahead.

5. If no further authentications with the legitimate network occur, then the attacker may use the captured SUCI to confirm the presence of the subscriber using the SUCI probe method from Figure 2.1, once the subscriber connects to the SUCI catcher at a later time.

This is already a narrower attack surface compared to the existing protocol: repeating the SUCI probe to compare the subscriber to a list of targets manifests as repeated failed authentication, which is prevented by devices that cancel registration after repeated failures. This limits the scalability of the attack—in terms of the number of targets that can be monitored—to the number of failed authentications allowed. [26] avoids this limitation in their SUCI catcher implementation by allowing a successful authentication between probes, but in our protocol this would advance the subscriber and invalidate any previous captured SUCI.

Additionally, there is a way to further mitigate this remaining attack: the 5G standards also provide for a periodic registration update, where the subscriber periodically re-registers with the network [10]. Initiating authentication as part of the periodic registration provides a way to ensure that any previously captured SUCI is invalidated while the subscriber is away from the physical SUCI catcher. Authentication as part of the periodic registration appears to already be supported by Cisco’s 5G implementation, so it would be compatible with the current standards [27].

Another solution would be to always perform a second authentication once the security context from the first is installed, since in this state the messages are encrypted [1], preventing an adversary from capturing a SUCI using an identity request. While more effective than performing authentication during periodic registration, since the hSUPI update occurs immediately, this requires a change to SN behavior unlikely to be supported in current implementations. However, this change would be a simple repeat of an existing protocol step, which would be easy for implementations to add.

Based on these points, we claim that our protocol defeats the SUCI catcher attack under concrete and achievable assumptions of network behavior, and that we provide a strict improvement to subscriber privacy.

2.4 Practical Considerations

In this section we detail some additional considerations that would apply to a practical implementation of our modified protocol.

2.4.1 Calculation and Storage of hSUPI

Our protocol requires both parties to store their [hSUPI](#) persistently, and ensure that the update process is atomic, so that correct protocol state is maintained. This should be very similar to the requirements for the storage of the sequence number by both parties. However, the [5G-AKA](#) protocol provides an explicit mechanism to resynchronize the [HN's](#) sequence number to that of the subscriber [11]. Although it provides an “escape hatch” in the case of any unexpected logic error that desynchronizes the counter state, the resynchronization mechanism is a point of vulnerability for subscriber anonymity. In fact, some existing attacks exploit the resynchronization mechanism to defeat subscriber privacy [17, 21, 22]. Our protocol would mitigate these attacks, as they rely on replaying an authentication request to trigger resynchronization, but the the replay would not have a valid MAC once the subscriber’s hSUPI is advanced.

To avoid similar attacks, we do not consider a resynchronization mechanism for the hSUPI sequence. As we discuss in Section 2.5.2, the [Tamarin](#) proof provides assurance that the subscriber and HN remain synchronized. However, there is always the possibility of unexpected software or hardware errors. Developing a resynchronization mechanism for the hSUPI under the same backwards-compatibility restrictions as our protocol, while avoiding vulnerabilities similar to those of the [SQN](#) resynchronization mechanism, is a potential area for future work.

2.4.2 On Our Use of Repeated Hashing

Repeated hashing as a method for generating the sequence of hSUPI values may not be a particularly attractive approach to pseudorandom number generation: some seeds may produce sequences with short period, and the output is the state, so all future outputs can be computed if one output is known. We used this method for modelling within Tamarin as it is easy to describe and shows that unpredictability from previous outputs is not required for the proof, since the hSUPI is encrypted before being sent. In practice, it is straightforward to replace the repeated hashing with another method of generating a pseudorandom sequence.

2.4.3 SUCI Format

Although in the Tamarin specification the [SUCI](#) is the straightforward asymmetric encryption of the SUPI, since this is sufficient to model its main features, an actual [IMSI](#)-based

SUCI has a specific format requirement, reproduced in Figure 2.2 from [1]. As we show in the figure, the [Mobile Country Code \(MCC\)](#) and [Mobile Network Code \(MNC\)](#) from the IMSI are included as plaintext, since they are needed to determine the country and HN respectively, to which the subscriber should be routed [2]. This can be retained in our modified protocol by continuing to copy these digits from the IMSI. A similar observation applies to the routing indicator digits used for intra-HN routing, although they are not part of the IMSI.

Bit								Byte
8	7	6	5	4	3	2	1	
Element = 5GS mobile identity								1
Length								2
								3
1	1	1	1	0	Type = SUCI			4
MCC digit 2				MCC digit 1				5
MNC digit 3				MCC digit 3				6
MNC digit 2				MNC digit 1				7
Routing digit 2				Routing digit 1				8
Routing digit 4				Routing digit 3				9
0	0	0	0	Protection scheme ID				10
HN public key ID								11
Scheme output								12–n

Figure 2.2: 5G SUCI format

The part of the IMSI that is encrypted is the [Mobile Subscription Identification Number \(MSIN\)](#), which is used by the HN to look up the subscriber after decrypting the SUCI [11]. Since the MSIN consists of 9–10 decimal digits, one approach to recreate this would be to define a reduction function from the space of hSUPI values (e.g. 256 bits if using SHA256) to the space of MSIN values. This reduction function would be used by the [UE](#) when creating an MSIN to encode in the SUCI and by the HN when looking up subscribers by hSUPI. However, this has the problem that even for 10-digit MSIN values, only a few hundred thousand subscribers can be supported before a collision is expected due to the birthday paradox (in fact it is even fewer since each subscriber has two valid hSUPIs as explained in the protocol description). This number can easily be exceeded by large network providers.

It would instead appear to be far easier to use the full hSUPI as an identifier since the HN will be the only one processing it. The SUCI format includes a length so the final scheme output field containing the encrypted data can be of variable length, which would allow for the full hSUPI to be encrypted and included in the SUCI. One concern with this is whether the argument against changing the authentication request—that there is

no evidence the [SN](#) would process fields of non-standard length—applies here too. It turns out that it does not: the standard provides two encryption schemes, with different output size, and further reserves four protection scheme identifiers for proprietary schemes specific to each HN [\[11\]](#). The proprietary schemes are allowed to expand the input size by up to 3000 bytes (apparently to allow for post-quantum algorithms). For this reason, the SUCI is not expected to be a specific size, so it is reasonable to assume the SN will forward the slightly longer SUCI containing the hSUPI to the HN as normal.

This analysis does point to a limitation of the subscriber privacy mechanism that is not addressed by our improvements: the country and network provider of the subscriber are identifiable through the MCC and MNC digits, which cannot be encrypted since they are needed to route the registration request to the correct HN. In most cases knowing the network provider is insufficient to identify a particular subscriber, but it is possible to imagine cases, such as when visiting a different country, where a particular network provider is rare enough to identify a subscriber with high probability.

2.4.4 Hardware Implementation

We expect our modified protocol to be easier to implement on the [USIM](#) than comparable modifications involving random nonces or public key cryptography. The original [3GPP](#) USIM (for 4G and earlier) did not support random number generation, which is one reason that 5G-AKA, based on the original authentication primitives, only uses symmetric-key cryptography [\[21\]](#). While our modification would require new logic to be implemented in the USIM, the pseudorandom sequence of ephemeral SUPIs, implemented through repeated hashing or otherwise, would only involve symmetric cryptographic primitives, which we expect are easier to implement on resource-constrained systems than asymmetric primitives. Our protocol also does not add any requirement for random number generation in the USIM, which would require specialized hardware that may not be present.

2.5 Analysis

To ensure the correctness of our protocol, we verified our protocol using [Tamarin](#). This section explains our approach. Our Tamarin model has been released publicly at [\[57\]](#): the details can be found in [Appendix B.1](#).

2.5.1 Modelling

As a starting point for our proof, we used the Tamarin model of 5G-AKA from Basin et al. [21]. To encode our modified protocol, the main operation that needed to be added was the computation of successive hSUPI values. The update rule chosen for our model is well-suited to implementation in Tamarin, as a hash function can be implemented as an arbitrary function symbol with no associated equations.

We also modelled the storage of the hSUPI by each party. A fact providing the current hSUPI for a given user is somewhere between a regular “linear” fact, which is consumed when used, and a persistent fact since while it can be modified, only one instance per user should exist. A natural way to implement this is using a linear fact, while ensuring that every rule that consumes it outputs a (potentially unchanged) replacement. Tamarin calls this an injective fact, and has special-case handling for some instances [65].

We implemented the protocol steps that had to accept two possible hSUPI values as two separate rules, using pattern matching (where legality is enforced by writing a rule’s prerequisites to be of the desired form) to ensure the right rule is selected.

2.5.2 Correctness

The first goal of our analysis was to show that it is always possible to successfully authenticate—that the hSUPI update rules ensure the two sides remain “in sync.” To prove this aspect of correctness, we proved the two lemmas: `ue_hsupi_update` and `hn_hsupi_update`. They respectively assert two facts

1. When the subscriber updates its hSUPI to some value, the HN has previously updated to the subscriber’s current hSUPI value.
2. When the HN updates its hSUPI to some value, the subscriber has previously updated to that new value.

Together, these two lemmas guarantee that the only two valid states of the hSUPI values are for both entities to have the same value, or for the subscriber to be one update ahead. This can be seen by considering an inductive argument on the sequence of updates, noting that both sides are initialized to the same value during subscriber registration as the base case.

The lemmas are also shown in Figure 2.3. As might be expected, the `HSUPI_HN_Init` and `HSUPI_HN_Update` action facts represent the initialization and update of the hSUPI by

```

lemma ue_hsupi_update:
  " All supi idHN hsupi_next hsupi #i.
    HSUPI_UE_Update(supi, idHN, hsupi_next, hsupi)@i
    ==> (Ex #j. HSUPI_HN_Init(supi, idHN, hsupi)@j
          & #j < #i)
        | (Ex #j. HSUPI_HN_Update(supi, idHN, hsupi)@j
          & #j < #i)
        | (Ex X key #j. Rev(X, <'k',key>)@j
          & Honest(X)@i
          & (X = supi | X = idHN)) "

lemma hn_hsupi_update:
  " All supi idHN hsupi #i.
    HSUPI_HN_Update(supi, idHN, hsupi)@i
    ==> (Ex hsupi_prev #j.
          HSUPI_UE_Update(supi, idHN,
                          hsupi, hsupi_prev)@j
          & #j < #i)
        | (Ex X key #j. Rev(X, <'k',key>)@j
          & Honest(X)@i
          & (X = supi | X = idHN)) "

```

Figure 2.3: Update Lemmas

the HN. The [UE](#) version of these facts operate similarly. These two lemmas include the security assumption that the [5G-AKA](#) key is secret.

Consistent with the experience of [\[21\]](#), we found that an [oracle](#) was required to prove our lemmas; the automated search was not effective for our protocol. In order to create an oracle, we considered what the “main argument” would be if the correctness of a lemma were to be presented manually. Using the graphical interface to Tamarin, it was possible to manually follow the argument for a few cases, and then use that experience to implement the same procedure in the oracle, so that it can be applied in the automatic mode. In this respect, the theorem prover does not eliminate the need for intuition in developing an argument, but is effective at automating the analysis of a large number of cases.

```

lemma ue_hsupi_unchanged:
  " All supi idHN hsupi #i.
    HSUPI_UE_Unchanged(supi, idHN, hsupi)@i
  ==> (Ex #j. HSUPI_UE_Init (supi, idHN, hsupi)@j
      & #j < #i)
    | (Ex hsupi_prev #j.
        HSUPI_UE_Update(supi, idHN, hsupi,
                        hsupi_prev)@j
      & #j < #i) "

lemma hn_hsupi_unchanged:
  " All supi idHN hsupi #i.
    HSUPI_HN_Unchanged(supi, idHN, hsupi)@i
  ==> (Ex #j. HSUPI_HN_Init(supi, idHN, hsupi)@j
      & #j < #i)
    | (Ex #j. HSUPI_HN_Update(supi, idHN, hsupi)@j
      & #j < #i) "

```

Figure 2.4: Unchanged Lemmas

Due to the stateful nature of the hSUPI updates, we used the inductive reasoning mode of Tamarin for the proofs of the update properties. Since the protocol does not guarantee a finite number of messages between updates, it was possible to search backwards infinitely without reaching the inductive hypothesis, so we developed two further lemmas to be reused in the proofs: `ue_hsupi_unchanged` and `hn_hsupi_unchanged`, shown in [Figure 2.4](#).

These lemmas provide that every hSUPI value that is used originated from an update or initialization, which is a simple property provable on its own using induction. Their reuse allows other proofs to conclude the existence of a previous update in situations that would otherwise lead to the infinite search problem—where the backwards search does not reach the inductive hypothesis.

Our modifications created a new [partial deconstruction](#) from using the hSUPI to form the [SUCI](#) sent from the registration request. A lemma linking the hSUPI back to a previous update or initialization could not be proven on its own because of the infinite search problem, and a more literal version linking the hSUPI to any previous rule that outputs one leads to an explosion in precomputation time. We found that, instead, promoting the more general `ue_hupi_unchanged` lemma to a [sources lemma](#) was effective at resolving the partial deconstruction.

2.5.3 Effectiveness

After the invariant preserved by the update lemmas, the second aspect of our protocol we sought to prove is the effectiveness of the mitigation against the SUCI-catcher attack. We expect that the ephemeral nature of the hSUPI provides [non-injective agreement](#) on the SUCI after the registration request. This property is expressed by the `hn_suci_from_ue` lemma. Proving this lemma required two helper lemmas:

1. `hn_ue_hsupi_unique`, asserting that two subscribers do not have the same hSUPI.
2. `hn_hsupi_secret`, asserting that when the hSUPI is known to the adversary, either the initial hSUPI or the HN’s private key was compromised.

All three lemmas are provided in Figure [2.5](#).

When first implementing the `hn_hsupi_secret` lemma we encountered a limitation in Tamarin’s ability to analyze the hSUPI sequence. Tamarin provides a build-in fact $K(x)$ for users asserting adversary knowledge, which normally should be used for lemmas about secrecy [\[66\]](#). This fact is implemented as an action fact on the rule that enables the adversary to send public messages—if the adversary can send something, then it must know it [\[50\]](#). The adversary knowledge is represented internally with two distinct types of fact: $K^\downarrow(x)$ —knowing a term x that can be deconstructed (e.g. a tuple), and $K^\uparrow(x)$ —knowing a term x that cannot be further deconstructed. The problem with this is that $K(x)$ derives from $K^\uparrow(x)$, so a search for $K(\text{hSUPI}_n)$, will consider $K^\uparrow(\text{hSUPI}_n)$, leading

```

lemma hn_ue_suci_unique:
  " All supi_a supi_b idHN_a idHN_b hsupi #i #j.
    HSUPI_HN_Unchanged(supi_a, idHN_a, hsupi)@i
    & HSUPI_UE_Use(supi_b, idHN_b, hsupi)@j
    ==> (supi_a = supi_b) & (idHN_a = idHN_b) "

lemma hn_hsupi_secret:
  " All supi idHN hsupi #i #j.
    HSUPI_HN_Update(supi, idHN, hsupi)@i
    & KU(hsupi)@j
    ==> (Ex X k #k. Rev(X, <'skHN', k>@k
    & Honest(X)@i
    & X = idHN)
    | (Ex X hsupi_init #k.
    Rev(X, <'hsupi', hsupi_init>@k
    & Honest(X)@i
    & (X = supi | X = idHN)) "

lemma hn_suci_from_ue:
  " All a b t #i. Commit(a, b, <'HSS', 'UE',
    <'SUCI', t>>@i
    ==> (Ex #j. Running(b, a, <'HSS', 'UE',
    <'SUCI', t>>@j
    & #j < #i)
    | (Ex X k #k. Rev(X, <'skHN', k>@k
    & Honest(X)@i)
    | (Ex X hsupi_init #k.
    Rev(X, <'hsupi', hsupi_init>@k
    & Honest(X)@i) "

```

Figure 2.5: Agreement Lemma, and Associated Helper Lemmas

to an infinite search sequence as $K^\uparrow(\text{hSUPI}_n)$ may be derived from $K^\uparrow(\text{hSUPI}_{n-1})$ (using the equation $\text{hSUPI}_n = \text{H}(\text{hSUPI}_{n-1})$), which in turn may be derived from $K^\uparrow(\text{hSUPI}_{n-2})$, and so on. Using induction fails to resolve this because the inductive hypothesis would be stated in terms of $K(x)$ rather than $K^\uparrow(x)$, and Tamarin does not currently model the implication $K(x) \Rightarrow K^\uparrow(x)$ [28]. To work around this limitation, we restated our lemma in terms of $K^\uparrow(x)$, using an undocumented feature where $K^\uparrow$ is available as KU . Doing so in a helper lemma is safe because it does not independently represent a required property and the conclusions of any lemmas proven using the helper lemma are unchanged.

One final change required to complete our proof was specifying exactly which parties may have particular secrets compromised in certain lemmas. This arose from an unexpected weakness in the generic style of property specification recommended by Tamarin, where each rule is tagged with assumptions on honest (i.e. uncompromised) parties [66]. When reusing a lemma with a compromise of a secret as a possible conclusion, the intention is for this compromise to be of a secret relevant to the security claim, a property expected to be provided by the explicit annotation of honest parties in the generic style of property specification. However, Tamarin is largely untyped (other than general distinctions such as fresh terms or constants). Avoiding strict typing has the advantage of preventing the accidental exclusion of attacks that may exist in a real implementation, but also allows conclusions that are not realistic. In the 5G-AKA model, the SN identifies itself to the HN, technically allowing an adversary to provide the identity of an unrelated subscriber or HN as the SN’s identity. Then, the annotation of the SN as honest in most protocol steps creates the assumption that this unrelated party is honest, preventing the conclusion that a compromised secret belongs to the HN or subscriber actually performing the protocol. This could be resolved by adding the assumption that the channel between the SN and HN is uncompromised, so the SN’s identity could not be manipulated by the adversary, but this weakens the conclusion by relying on a security assumption not expected to be required in practice. Instead, the parties to whom a secret may actually belong were explicitly specified in each of our lemmas designed for reuse.

It is not expected that this weakness in the generic style of property specification affects existing work, since any counterexample accidentally excluded would have to be a result of the identity confusion, and the parties in 5G-AKA are not at all treated interchangeably. However, any divergence between a lemma and the formal property it is meant to represent is deserving of scrutiny, so future implementations may wish to avoid the issue. As well as the method used in our lemmas, which was easy to “retrofit” to an already annotated model, a possible approach could be to add explicit tagging to the identity in the annotations, as is already done for different secret items.

Chapter 3

“Bring Your Own Identity” for 5G

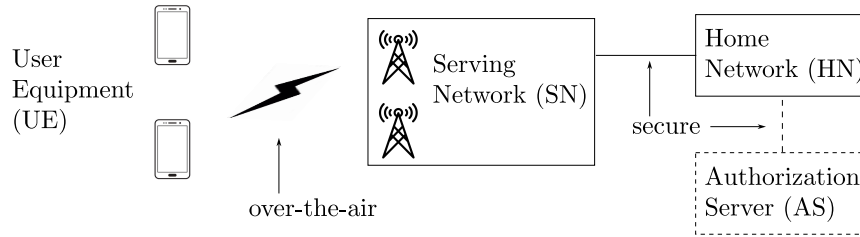


Figure 3.1: 5G Network Model with Authorization Server

In this chapter, we explore a possible implementation for “bring your own identity” in 5G. Our definition of “bring your own identity” is that the user identity is managed by an external identity provider, instead of being implicitly tied to the HN. Figure 3.1 shows that extension of the 5G network model, which we consider in this chapter, that adds an [Authorization Server \(AS\)](#) operated by the identity provider.

This concept has several advantages. One is that a goal of future cellular standards is to provide seamless enterprise access [59]. A user who is a member of, say, a university, should be able to access the cellular network in that role. Backing this type of use-case up is an agreement between the university and a cellular telecommunications company so that, for example, the university, and not individual members of the university, gets billed for usage. And a member of the university has established an identity with that university only, and not the telecommunications company, i.e., the university is the identity provider, while the telecommunications company is the billing agent, i.e., the traditional HN. Another

advantage with bring your own identity is that an identity bound to a user in a manner that is agnostic to the services of which the user avails is then a cross-domain identity [45, 48]. Finally, there is also a security benefit: we can legitimately perceive identity provision as a sophisticated technology. This is a reason that it has emerged as a business opportunity for identity providers [38, 52]. By allowing a separate entity to focus on the technology that underpins the security and privacy of users in a system, the overall security posture of cellular communications improves.

Our protocol design is based on composing 5G-AKA with OpenID Connect, a common protocol used for authentication in web applications. Before presenting the protocol design, we discuss some related work in Section 3.1. We also explain OAuth2.0 and OpenID Connect in Section 3.2 to provide a background for our use of these protocols. Section 3.3 is the specification of our protocol, after which we present the results of our formal verification in Section 3.4. We noted some limitations of the formal security properties that we employed, which we discuss separately in Section 3.5. Finally, Section 3.6 describes a prototype implementation of our protocol that we built.

3.1 Related Work

3GPP has studied a similar concept to bring your own identity in TR22.904 [5] that they term “User-Centric Identifiers.” The study considers several possible use cases, including where multiple users share a UE, and recognizes that identity provision may be handled by an external party. The scope of the report is limited to collecting requirements for such a system, and does not propose a design.

Modifications and improvements to 5G-AKA are a popular topic of study. For example, previous works have considered forward secrecy [18, 68], and symmetric-key-only designs [23, 53, 67]. We believe our work is unique in being the first practical variant of 5G-AKA that allows a separate identity provider for primary authentication.

Several authors [41, 69] propose the use of blockchain technologies for subscriber authentication in 5G. Depending on implementation, this may allow for user identity to be managed outside the HN. However, blockchains also have distinct disadvantages, acknowledged by these pieces of prior work, such as slower update latency. Instantiating a blockchain-based protocol requires a decision as to which blockchain implementation is to be used. By comparison, our protocol is based on OAuth2.0 and OpenID Connect in order to take advantage of existing authentication implementations that have been proven through practice.

The 3GPP standard for the 5G security architecture [11] describes how EAP-TLS [31] may be used for primary authentication within private networks. In this way, UE identity is based on a client TLS certificate. This is convenient for networks of IoT devices, but TLS certificates are not a common form of identification for human users, which poses a challenge to extending this authentication method to public networks without other modifications.

Other papers [42, 63] have already considered OAuth2.0 or OpenID in the context of 5G networks, but only as a way to authenticate with additional services after primary authentication. 3GPP has also provided a similar standard [8] where the HN acts as the identity provider for OpenID authentication, allowing the user to authenticate to other services. This is the opposite of bring your own identity: instead of bringing an external identity to 5G-AKA, the identity for 5G-AKA (the SIM card) is used externally.

3.2 OAuth 2.0 and OpenID Connect

3.2.1 OAuth 2.0

OAuth 2.0, specified by RFC 6749 [40], is an authorization framework for online services to allow users to share access with third parties. Third party access is controlled through [access tokens](#), which allow such access to be revocable and limited in scope.

A third party that wishes to obtain access to a service on behalf of a user is known as a [client](#) in OAuth 2.0 terminology. The client obtains an access token from an [AS](#) operated by the service by following the procedure in Figure 3.2. The user interacts with the client and AS using a browser. The figure shows the “Authorization Code Grant” authorization mechanism. While there are other methods in the RFC, this method is commonly used in practice, and is the most relevant to our work. The main steps in the procedure, labeled (A)–(B) in the figure, are:

- (A) The user makes an authorization request to the AS composed of the client’s ID (client.id) and a [URI](#) (redirect_uri) that points back to the client. The request is initiated by the client: for example, in the case of a website, the client may use a redirect to cause the user to make the request, as shown in our figure.
- (B) The user is authenticated by the AS. There is no procedure specified by OAuth 2.0, but this is commonly done by having them log in to their account. The AS also confirms that the user consents to granting the client access.

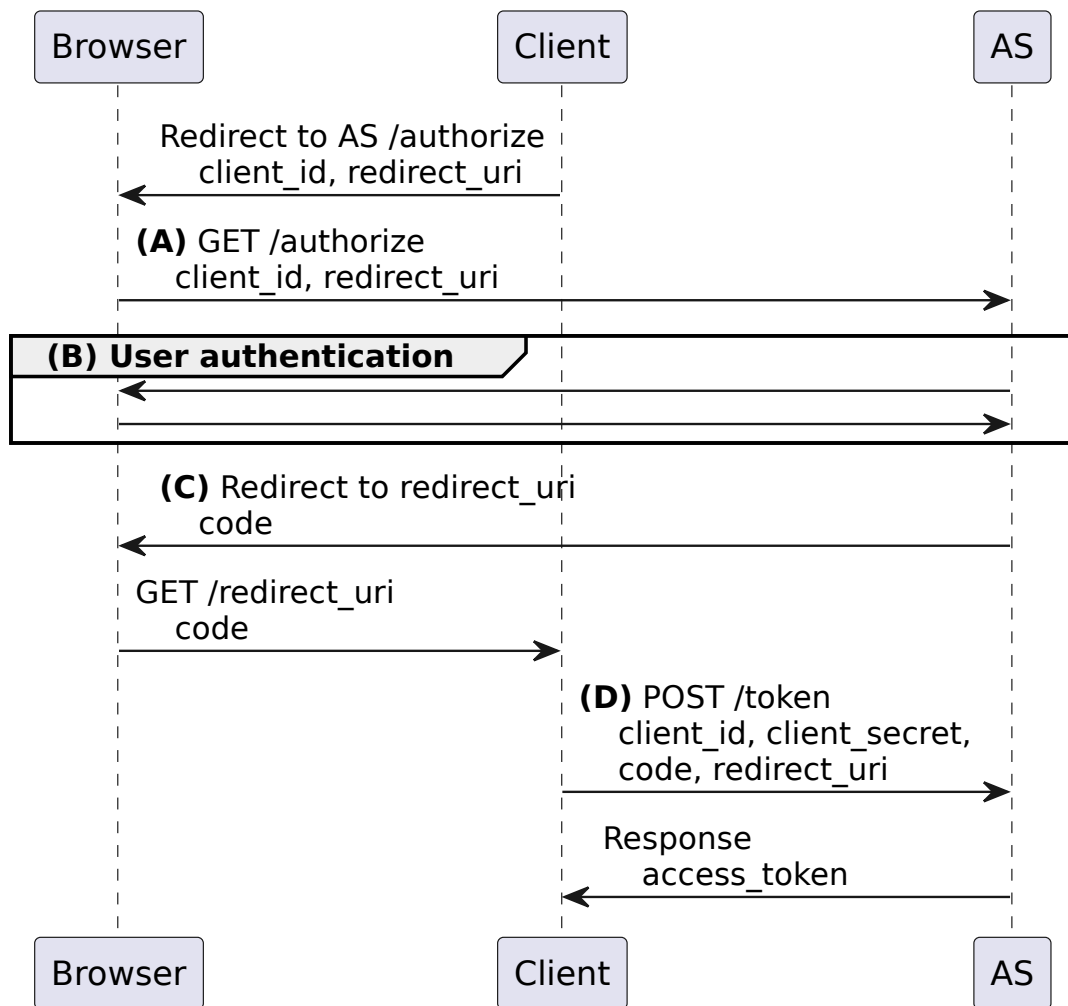


Figure 3.2: OAuth 2.0 Authorization Code Grant

- (C) Assuming user authentication is successful, the AS provides an authentication response by redirecting the user back to `redirect_uri`, including an [authorization code](#) (code) in the URI parameters, so that it can be read by the client. `redirect_uri` is registered in advance with the AS to ensure it is owned by the client.
- (D) The client exchanges the authorization code for an access token by making a request to the AS. The client authenticates by providing a secret (`client_secret`) that was given to it by the AS during client registration.

The two-stage process provides improved security, since it is easier to intercept the authentication response than the token response. For example, when the client is an application rather than a website, then common methods of providing a `redirect_uri` are to use a private URI scheme registered by the application, or to simply use the loopback interface (`localhost`), and both methods may allow the response to be intercepted by a malicious application running on the same device [32]. By contrast, the token request is a direct request and response to the AS, protected by [TLS](#).

3.2.2 Security Enhancements

Practical OAuth 2.0 implementations usually support additional features to enhance security. Two such features are [Cross-Site Request Forgery \(CSRF\)](#) protection and [Proof Key for Code Exchange \(PKCE\)](#).

One potential CSRF attack on OAuth 2.0, described in section 10.12 of the RFC [40], occurs when an attacker is able to get the user to access the client's `redirect_uri`. In this case, the attacker can provide the authorization code, potentially causing the user to associate an attacker-controlled account with the client. To prevent this, OAuth 2.0 allows a state parameter to be included in the authentication request. The AS will include the same parameter in the authentication response; if the client checks that the two parameters match, it is assured that the response corresponds to the request.

PKCE [62] is designed to mitigate the impact of an attacker intercepting an authorization code. When using PKCE, the client generates a random value (the code verifier) and sends its SHA256 hash (the code challenge) in the authorization request. The AS saves the code verifier and requires the client to present the code challenge in the token request. This ensures that an attacker who intercepts the authorization code is unable to redeem it for an access token. PKCE is especially important for clients that are unable to protect a client secret (such as native applications) as the token request from these clients is otherwise unauthenticated.

3.2.3 OpenID Connect

While OAuth 2.0 includes an authentication step between the user and AS, it is primarily designed to provide the client with authorization to access resources on behalf of the user. The RFC does not include a standard method for the client to query the identity of the user. OpenID Connect [61] fills this gap by building an authentication protocol on top of OAuth 2.0. The feature most relevant to our work is that an ID token attesting to the user’s identity is included with the access token response, so the client does not need a separate query to identify the user.

3.3 Composing 5G-AKA and OAuth 2.0

This section describes our protocol design (Section 3.3.1), and some interactions with the existing 5G architecture (Section 3.3.2). We then develop a set of security properties that we expect from the composed protocol in Section 3.3.3.

3.3.1 Our Protocol

Our composition is shown in Figure 3.3. Our exposition follows that of the prior work of Basin et al. [21] with a few exceptions—we retain the SN and HN roles, split the Subscriber role into a UE and the User (the human subscriber), and add the AS role. We also include K_{SEAF} in the final message sent to the HN by the SN, instead of the authentication request. This reflects a change in the official specification between version 15.1.0 of TS33.501 [4], used by [21], and more recent versions of the specification [11].

We assume that, as with standard OAuth 2.0, the user interacts with the AS over a browser. This is trivial in cases where the user has Wi-Fi access during initial authentication; otherwise, this requires the SN to provide a restricted network connection before the user has authenticated, such as by obtaining a list of possible addresses from each authorization server, and restricting connections to only those endpoints. Defining a variant of OAuth 2.0 that doesn’t operate over HTTP may be a useful topic for future work. (We use the term variant loosely since such a design would be almost a new protocol, and would have to standardize the method of user authentication.)

The client role, from OAuth 2.0, is distributed between the UE and HN, so we define two new keys used to exchange URIs:

$$EK_{HN} \parallel EK_{UE} = \text{KDF}(CK \parallel IK, \text{“5GBYOIEK”}),$$

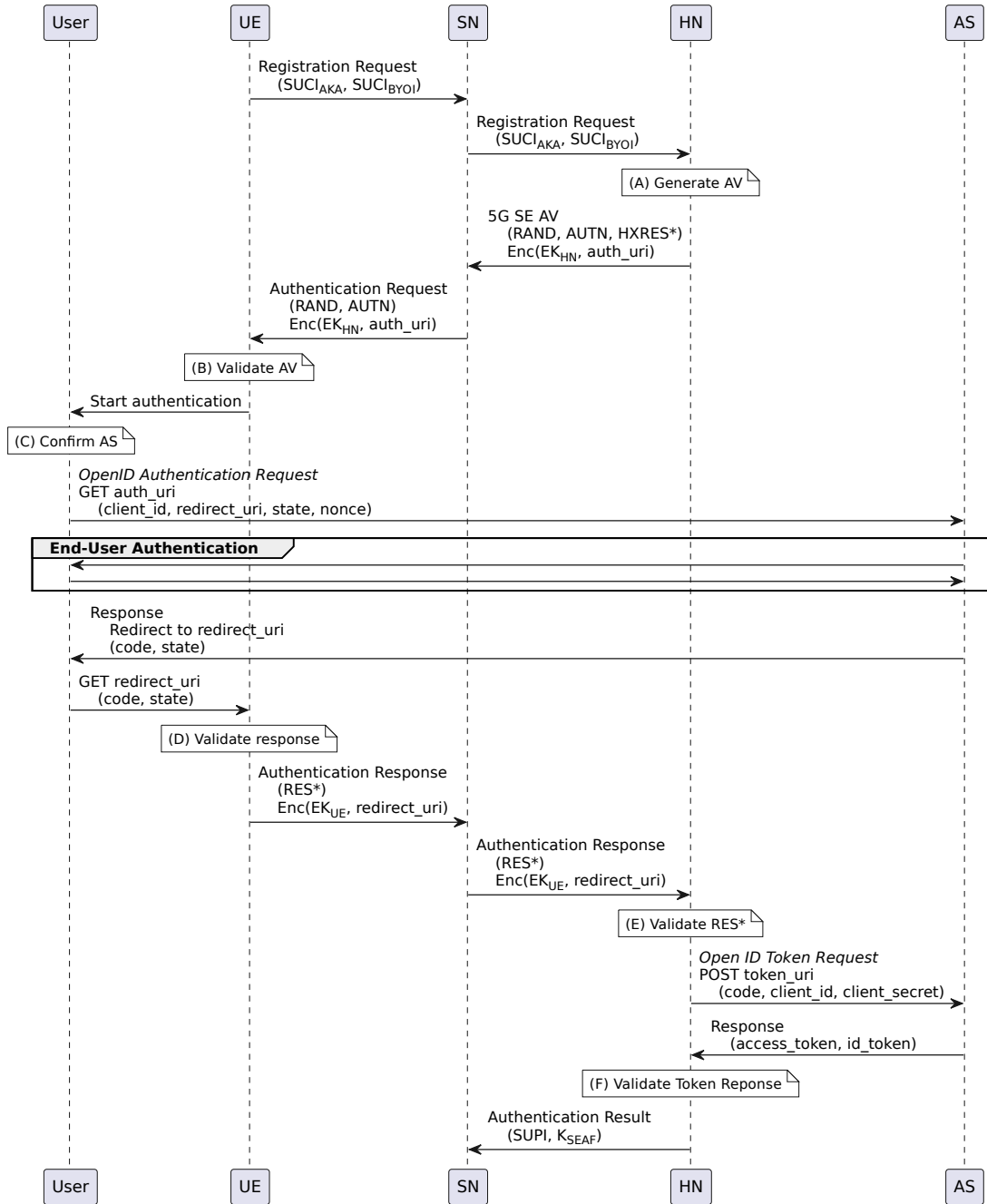


Figure 3.3: 5G-AKA composed with OAuth 2.0

where KDF is a key-derivation function and

$$CK = f_3(K, \text{RAND}) \quad IK = f_4(K, \text{RAND}),$$

are keys already derived during 5G-AKA, as described in Appendix A.

Since the SN initiates authentication, we require the UE to indicate the user’s choice of AS in the registration request. The 3GPP specifications [2, 13] already define a SUPI format for private networks that is simply a Network Access Identifier (NAI) as specified by RFC7542 [30]—effectively username@realm with some restriction on allowed characters. We adopt this format for our protocol, with the URI of the AS used as the realm, and have the UE send a second SUCI, $\text{SUCI}_{\text{BYOI}}$, containing this form of SUPI.

This introduces a potential attack on the protocol: as the registration request is not bound to the subsequent authentication, it is easy for an attacker to change the requested AS. To prevent this, the UE confirms the choice of AS with the user before starting the OAuth flow. This prevents the attack entirely, but we still point out some approaches to reduce the impact of human error on this step:

- The HN only allows an AS from a pre-defined list, so the number of ASes that may be compromised or malicious is limited. This is normal in OAuth scenarios, as the HN must decide in advance which providers it trusts to attest to the user’s identity.
- The UE can automatically reject authentication requests involving an AS that was not sent in a recent authentication request.
- The registration request can be authenticated with a key derived from the 5G-AKA key K . This feature would have to be added to the USIM as K is not exposed directly to the UE.

We now elaborate on the steps marked (A)–(F) in Figure 3.3.

- (A) The HN generates an AV as normal. Additionally, it forms the URI for an OAuth 2.0 authorization request, `auth_uri`, according to the OpenID Connect specification. The redirect URI in the request must point back to the UE, so we assume that, as described in [32], either localhost or a custom URI scheme is provided. `auth_uri` is encrypted using EK_{HN} and included with the AV.
- (B) The UE validates the AV as normal and then decrypts `auth_uri`. The UE also extracts the state parameter from the URI and saves it.

- (C) As previously mentioned, the UE confirms the choice of AS with the user before directing the browser to `auth_uri`. OAuth 2.0 proceeds normally between the user and AS. The part of the figure marked “End-User Authentication” corresponds to the AS’s authentication of the user.
- (D) The UE receives a redirect containing the OAuth 2.0 authentication response. It independently verifies the state parameter matches the value saved previously before encrypting the entire response URI, `redirect_uri`, using EK_{UE} . The encrypted response URI is included with the 5G-AKA authentication response.
- (E) The HN validates the 5G-AKA authentication response and decrypts the response URI. It independently checks the state parameter before sending an OAuth 2.0 [access token](#) request. We assume the access token request is served by the AS. In practice it may use a separate URI, but this isn’t an essential point.
- (F) The HN receives an access token and an OpenID Connect ID token before providing the SN with a successful authentication result.

The HN may read the user’s identity from the ID token. If further information is required, it may make requests to the provider of the AS using the access token; we do not show this step.

3.3.2 Compatibility with 5G architecture

It is important that our protocol is compatible with the ways in which 5G-AKA, and other standard authentication methods, are used within the 5G system. Two important aspects that involve identity and session management are roaming and reauthentication.

Roaming in 5G is supported through two configurations: local breakout and home routed [13]. These configurations differ primarily in the provision of the data network, which handles features such as internet access: in local breakout roaming, the visited network provides access through their own services; in home routed roaming, the visited network routes user traffic back to the home network. However, the authentication service always rests within the home network. By keeping the distinction between SN and HN in our protocol design, we retain compatibility with this model.

Explicit implementation of our protocol is still required on the SN side since it must include the additional data added to the authentication request and response. This requires a software change to SNs that will act as roaming provider, but it is difficult to avoid this

requirement as the authentication messages formats are fixed-length: a point that we discuss in Section 3.6.

As discussed in Section 1.2.4, authentication in 5G-AKA is initiated by the network [11], so it is not tied to the registration request, as might be implied by the previous figure. The SN may initiate authentication with the HN using the SUPI instead of the SUCI in cases where the user identity is known but reauthentication is desired. To support reauthentication using our protocol, the HN can store the most recent choice of AS, or pass the user identity along with the SUPI in authentication result, so that the SN may include it in any reauthentication request.

This discussion raises the important point of session management with respect to the user identity. Reauthentication, as described, does not allow for a different choice of AS. Even more fundamentally, once the UE has authenticated, the same authentication context may be reused for subsequent registrations, and even transferred to another network, without authentication [10, 11]. For some uses, it may be useful to support an explicit “log out” mechanism on the UE that will deregister from the network and delete the authentication context, so that the device can be easily re-registered to a different user.

3.3.3 Desired Properties

To meet our expectation for bring your own identity, we expect our protocol to provide the same guarantees as OpenID Connect in its normal setting. This ensures the user is authenticated to the HN via the AS. For the purpose of this analysis, we adopt the authentication properties defined by Lowe [47], as summarized in Section 1.4.

The security properties of OAuth 2.0 and Open ID are not explicitly stated in their specifications [40, 61], so we infer them from the behavior and design of the protocol:

- The AS is required to authenticate the user, so we expect *weak agreement between the user and AS*.
- The AS also asks the user to explicitly confirm they are granting access to the client [46], so we expect *non-injective agreement on the HN’s identity between the user and AS*, as the HN is playing the role of the OAuth 2.0 client.
- Through Open ID Connect, we also use the OAuth 2.0 process as an authentication protocol, so we expect *weak agreement between the user and HN*.
- The client presents its client secret to the AS to prove its identity. This is a form of authentication, so we expect *weak agreement between the HN and AS*.

- Through the access token, the AS grants the client authorization to access resources belonging to a user, so we expect *injective agreement on the access token* and *non-injective agreement on the user's identity between the HN and AS*.
- For completeness, we also consider *non-injective agreement on the AS's identity between the user and HN*. This is less explicit, but is still a reasonable expectation of the protocol.

We also desire that our protocol retains the existing authentication properties of 5G-AKA, so we consider pairwise weak agreement and pairwise agreement on the SUPI, the SN's name, and K_{SEAF} , between the UE, SN, and HN, to allow a comparison to the analysis of 5G-AKA by Basin et al [21].

Finally, we also expect that certain sensitive data should remain secret in our protocol. The items we identify are:

- *The SUPI*, since it can be used to track users if revealed.
- K_{SEAF} , since it is used to derive further keys for 5G operation.
- *The user's identity*, for the same reason as for the SUPI.
- *The user's password, client's secret, and OAuth 2.0 authorization code*, all items reasonably expected to remain secret. (We assume that the user authenticates with the AS using a password, as this is the most common scenario.)
- *The access token*, as it can be used to access the user's information.

We do not explicitly consider the secrecy of long-term keys that are used but never sent in any message, such as the 5G-AKA key or the HN's private key, as these are always secure in any symbolic model (except for the trivial case where they are revealed directly).

3.4 Analysis

We now discuss the manner in which we have analyzed our composed protocol from the previous section for the properties we desire. We first explain how we model our protocol and the properties using Tamarin in Sections 3.4.1 and 3.4.2, and then detail our results in Section 3.4.3. Our model will be made publicly available, as explained in Appendix B.2.

3.4.1 Model

Although our model is inspired by the work of Basin et al. in [21], we implemented the model from scratch as it would anyway require extensive additions to include our changes.

We keep the separation of user and UE as different parties, since this is fundamental to the analysis of OAuth 2.0. In many other authentication protocols, authentication is based on long-term keys, allowing the protocol to be executed automatically. The association between the keys and different users is implicit. Here, the user participates in the authentication by interacting with other parties through a browser.

Modelling the user naturally requires a model of the browser and HTTPS requests. As a starting point, we implemented a solution based on [35] (specifically the “HTTPS V3” model), but found that our model became too complex to load within a reasonable amount of time and memory. We suspect that this was due to the large space of possible behaviors introduced by the model, especially involving redirects. To address this, we created a parallel copy of the model for each HTTPS request made in the protocol, and modelled redirects as advancing to the next request. This represents a limitation of our model: although attacks involving interception, modification, or initiation of requests are properly considered, attacks that require out-of-sequence redirects may be incorrectly excluded.

Our model includes a key-confirmation stage as [21] identifies this as a requirement for achieving certain security properties with 5G-AKA. In practice, this would be provided by the UE and SN attempting to activate the security context after authentication, as specified in [1].

3.4.2 Properties

For our analysis of authentication, we adopt the formalization of Lowe’s hierarchy discussed in Section 1.4.2. As for secrecy, that section also describes the lack of an intuitive formulation of secrecy claims.

Basin et al. [21] describe inserting a secrecy claim “for each rule of agent a ... who accesses or stores the identifier **SUPI**,” but do not do so consistently in their model: the UE does not claim secrecy of the SUPI when it sends it as part of the registration request, but instead after receiving an authentication request. While this does not make a difference for 5G-AKA, it is easy to imagine related scenarios where it would. If 5G-AKA were extended so that a UE could subscribe with multiple HNs, the secrecy claim would fail on traces where the UE sends a registration request to a malicious HN before authenticating with a legitimate one, but only because the set of trusted parties is misidentified. With the

secrecy claim instead made when sending the registration request, any HN to whom the UE sends a registration request would be correctly identified as trusted.

We submit that the reason placing secrecy claims is difficult is that there are really three subtly different types of claims commonly made. Addressing them separately provides a systematic method for correctly inserting claim events into a model. The three forms of claim we identify are:

- *Input secrecy*: A data item with external significance is an input to the protocol and is required to remain secret. The SUPI in 5G-AKA is one example where this is applicable. The secrecy assertion should be made by the party to whom the data belongs, whenever they use it for the protocol.
- *Output secrecy*: The protocol generates a data item that is expected to be secret. In the case of a key agreement protocol, this applies to the agreed-upon session key. The secrecy assertion should be made by all parties relying on the secrecy of the data before they consider the protocol complete, but does not have to be made at the first opportunity.
- *Internal secrecy*: A data item is only significant within the protocol, but must intuitively remain secret for correct functionality. This usually applies to long-term keys. The placement of the secrecy assertion is more heuristic since this case is an internal assertion—if internal secrets are not protected, then successful proofs of other properties may be erroneous—but it is reasonable to make the assertion when the data is used.

These definitions were used when writing the secrecy properties for our model. The specific mapping between these definitions and the data items we consider is included with our results.

3.4.3 Results

We summarize the results of our analysis in Tables 3.1, 3.2, and 3.3, using a similar notation to [21], where $\neg X$ means data item X is not revealed. The data items we consider are K —the 5G-AKA key, SK_{HN} —the HN’s private key, pw —the user’s password, and cs —the client secret. Additionally, $\neg ch$ means the channel between the SN and HN is not compromised, $\neg \text{HTTPS}$ means HTTPS is not compromised, and PKCE means PKCE is performed. STATE indicates that the UE checks the state parameter of the OAuth 2.0

Item	Type	Assumptions
SUPI	Input	$\neg sk_{hn} \wedge \neg ch$
User identity	Input	$(\neg cs \wedge \neg \text{HTTPS}) \vee (\text{PKCE} \wedge \neg K \wedge \neg \text{HTTPS})$
User password	Internal	$\neg \text{HTTPS}$
Client secret	Internal	$\neg \text{HTTPS}$
Authorization Code	Internal	$\neg K \wedge \neg pw \wedge \neg \text{HTTPS}$
Access Token	Internal	$(\neg cs \wedge \neg \text{HTTPS}) \vee (\text{PKCE} \wedge \neg K \wedge \neg pw \wedge \neg \text{HTTPS})$
K_{SEAF}	Output	$\neg K \wedge \neg ch$

Table 3.1: Secrecy Properties

authentication response. Finally, *wa* means the property is equivalent to [weak agreement](#), and $\times$ means it could not be satisfied.

The secrecy properties in Table 3.1 are all achieved with reasonable assumptions. As might be expected, PKCE provides an alternative way to the client secret of preventing an attacker from redeeming an authorization code for an access token. For each result, we include our assignment to one of the three forms of secrecy claims in the Type column.

Table 3.2 shows the authentication properties between the existing 5G-AKA parties. Compared to the assessment of unmodified 5G-AKA in [21], there are two differences. One is that injective agreement on K_{SEAF} between the HN and SN is not achieved from the HN’s point of view. This is a consequence of sending K_{SEAF} in the final message between HN and SN: the HN does not have confirmation the HN received the message (at least within the scope of the protocol). This difference is explained in Section 3.3.1.

Another difference is that non-injective agreement on the SN’s name between the HN and UE requires $\neg K \wedge \neg ch$ instead of just $\neg K$. This is due to a small modeling difference. Without $\neg ch$, the attacker may inject messages on behalf of any SN. The model of [21] saves the name of the SN that sends the registration request and requires that the authentication response be sent (apparently) by the same SN. Our model does not enforce this link, so the HN may not agree with the UE if a different SN provides the authentication response. In practice, this property is expected to ensure that both UE and HN are communicating with the same SN, so it is not unreasonable to require $\neg ch$ so that further messages sent to the HN originate with that SN.

The properties involving new parties in Table 3.3 are all met, except for where the [AS](#) is unable to confirm the HN’s receipt of certain data items—the user’s identity and the [access token](#)—since there is no further message after the access token request. This behavior is present in standard OAuth 2.0. Although it may be possible to model a further request

Point of View Partner	UE				SN		HN	
	SN	HN	UE	HN	UE	HN	UE	SN
Weak Agreement	$\neg K \wedge \neg ch$	$\neg K$	$\neg K$	$\neg K \wedge \neg ch$	$\neg K$	$\neg ch$	$\neg K$	$\neg ch$
Non-injective Agreement on SUPI	wa	wa	wa	wa	wa	$\neg ch$	wa	$\times$
Non-injective Agreement on SN's ID	wa	$\neg K \wedge \neg ch$	wa	wa	wa	wa	$\neg K \wedge \neg ch$	wa
Injective Agreement on K_{SEAF}	$\neg K \wedge \neg ch$	$\neg K$	$\neg K$	$\neg K \wedge \neg ch$	$\neg K \wedge \neg ch$	$\neg K \wedge \neg ch$	$\neg K$	$\times$

Table 3.2: Authentication properties between existing 5G-AKA parties

Point of View Partner	User			HN		AS	
	HN	AS		User	AS	User	HN
Weak Agreement	$\neg K \wedge \neg pw$ $\wedge \neg \text{HTTPS}$ $\wedge \text{STATE}$	$\neg K \wedge \neg pw$ $\wedge \neg \text{HTTPS}$ $\wedge \text{STATE}$		$\neg pw \wedge \neg \text{HTTPS}$	$\neg \text{HTTPS}$	$\neg pw$	$\neg cs$
Non-injective Agreement on User's ID	wa	wa		wa	$\neg \text{HTTPS}$	wa	$\times$
Non-injective Agreement on HN's ID	wa	$\neg K \wedge \neg pw$ $\wedge \neg \text{HTTPS}$ $\wedge \text{STATE}$		wa	wa	$\neg pw \wedge \neg \text{HTTPS}$	wa
Non-injective Agreement on AS's ID	$\neg K \wedge \neg pw$ $\wedge \neg \text{HTTPS}$ $\wedge \text{STATE}$	wa		$\neg pw \wedge \neg \text{HTTPS}$	wa	wa	wa
Injective Agreement on Access Token	-	-		-	$\neg \text{HTTPS}$	-	$\times$

Table 3.3: Authentication properties involving new parties

using the access token, much like the implicit key confirmation of 5G-AKA, such a request is not required when using OpenID Connect, as the user’s identity is already provided in the access token response.

The assumptions required for weak agreement between the AS and user or HN, from the AS’s perspective, appear weak— $\neg$ HTTPS should also be required to protect the secrets used to authenticate in each instance. This is addressed in the next section.

Overall, we consider our security goals to be achieved: the secrecy properties were all met; the 5G-AKA authentication properties were preserved, as neither of the two identified differences were caused by our changes; and the new authentication properties were achieved, except where not possible under standard OAuth 2.0 + OpenID Connect.

3.5 Nuances of Formal Verification

While constructing our model and the associated proofs, we found several instances where the methodology suggested in the [Tamarin](#) manual [66] did not behave as expected. We detail them here as they represent potential areas for further development of formal authentication properties.

3.5.1 A Weakness of Weak Agreement

We note that the results for [weak agreement](#) between the [AS](#) and user or [HN](#) are intuitively wrong: $\neg$ HTTPS should be required, as otherwise the user’s password or HN’s client secret are exposed to any attacker, trivially compromising their use as authenticators. This is an artifact of how weak agreement is defined. Recall that weak agreement only requires that the responder has been running the protocol, apparently with the initiator [47]. Consider a degenerate authentication protocol where a client sends their password in plaintext to a server. Even though the password may be captured and reused many times, it must have been originally publicized by the client during a protocol run, so weak agreement is achieved. From this, we conclude that weak agreement does not ensure that a protocol is usable for more than one authentication.

To address this limitation, we propose a modified definition of weak agreement. A protocol achieves *extended weak agreement* if, after any time T_0 , when an initiator A starts and completes a run of the protocol, apparently with responder B, and no protocol runs that began before T_0 are advanced, then B has been running the protocol, after T_0 , and apparently with A.

This definition captures the intuitive requirement that it should not be possible to link all protocol runs by A to a single run by B arbitrarily far in the past. It also has unique advantages. Compared to recent weak agreement, another definition of Lowe that requires the responding session to have occurred within a certain time duration, extended weak agreement is easier to formally verify, as it only requires an order on events, and not a full model of timestamps and elapsed time. Unlike [injective agreement](#), it can be analyzed without reference to any particular data item.

As an experiment, we implemented a Tamarin model of the Andrew protocol as described and attacked by [24]. The core flaw in the protocol is that the server sends the client a session key in a final message not bound to any previous protocol message. This allows an attacker to substitute any session key message from a previous protocol run. Despite this, the protocol achieves weak agreement and [non-injective agreement](#) on the session key from the client’s perspective. (Interestingly, Lowe provides this as an example of a protocol that does *not* achieve non-injective agreement [47], but we find that it does—a replayed session key still originates from a legitimate responding session.) Yet, the replay attack prevents the protocol from achieving modified weak agreement: the attacker can cause a client who begins a protocol run after T_0 to consider the protocol complete before the server has sent the final message. This example shows how our definition can be useful for the analysis of authentication.

3.5.2 Non-Algorithmic Parties

As mentioned previously, we modelled the user as a separate entity because they interact with the AS directly through the browser without mediation by the UE. This required special attention to how parties that do not interface directly with the underlying cryptographic algorithms still achieve agreement. As an example, a user who signs in to a website does not directly inspect the [TLS](#) messages exchanged between that website and their browser to determine that subsequent interactions are part of the same session; instead, they make an assumption that the software involved (primarily their browser) is stateful and only maintains one session at a time. This form of assumption is also reasonable in the context of 5G: the UE is stateful and cannot maintain simultaneous authentication runs. (See [1] for a detailed specification of the UE’s behavior in relation to the [Non-Access Stratum \(NAS\)](#), over which registration and authentication are performed.)

By default, however, a Tamarin model will allow for unlimited concurrent sessions. The multiset-rewriting rules that define protocols can be applied in any sequence, which is key to modelling concurrent execution by different parties, but this also allows multiple sessions

to run using the same identity. Usually this poses no issue as the cryptography involved prevents any mix up of messages intended for different sessions. In some cases, multiple sessions are an expected part of the model: for example, the HN interacts with many UEs simultaneously. Yet, when modelling a party that can only link steps by observing they run consecutively, this behavior is insufficient for the required proofs.

We first attempted to model the stateful nature of the UE directly by maintaining a state for each UE and only allowing certain transitions. We then required that, for agreement, the user start and finish the protocol without there being any intervening user sessions on the same UE. (If the user passes their device to someone else, then obviously that person may log in instead.) Unfortunately, this model has poor performance, especially when reasoning about the state, so we were not able to complete the required proofs. It appears that a model with more constrained behavior, while conceptually simpler for human understanding, can produce more challenging constraints for the Tamarin prover.

As a second attempt, we created a session identifier that is provided to the user by the UE during each observable step, and required as a pre-condition of agreement that the user observe the same session identifier in each protocol step. This may seem to be an ad-hoc invention that doesn't correspond to physical reality; however, we draw a correspondence in the following manner. First, the behavior of users other than the subject of agreement has fewer restrictions, so the session-identifier model represents a strictly weaker assumption than the stateful model in this respect. Second, the pre-condition in the stateful model that the user start and finish the protocol while maintaining possession of the UE would result in observing a consistent session identifier, so the two models are equivalent for a single session. The only way in which the session-identifier model represents a stronger assumption is that it applies to previous sessions by the user who is asserting agreement. Because our model of the user does not maintain any state between sessions, we believe it is unlikely for attacks to be missed due to this difference. We used the session-identifier strategy for the user-to-X agreement properties in Table 3.3.

3.5.3 Many-to-One Relationships

As explained in Section 1.4.2, in the formalization we use, rules are annotated with assumptions on honest (i.e. trusted) parties (using the **Honest action fact**), so that it is easy to state generically in theorems that an honest party does not have a particular data item compromised. This is important since it allows implicit modelling of compromised parties: if parties not explicitly trusted can have their keys compromised, then the attacker can simulate arbitrary behavior on their behalf through its ability to send and receive messages.

The implicit nature of this modelling makes it possible to inadvertently exclude the possibility of compromised parties in cases where there is a many-to-one relationship between parties. For example, since the 5G-AKA key K is symmetric, it would be usual to consider it as belonging to both the UE and the HN, but then any lemma involving the HN that assumes no honest party has K compromised excludes the K of all UEs subscribing with the HN from compromise. This creates the stronger assumption that there are no compromised UEs on the HN’s network.

For lemmas involving the UE and HN directly, it is easy to adjust the claims to only require that the UE have no compromised K . However, this becomes more difficult when the UE is trusted transitively, such as with the secrecy of the authorization code: the AS does not know the identity of the UE, but does rely on its behavior to protect the code. In this case, to avoid excluding all UEs from compromise, we specified that specifically UEs who were used to authenticate with the AS should not have K compromised.

Another example is the secrecy of the SUPI, where [21] notes that they had to assume the HN–SN channel was secure for all SNs: the SUPI is unchanging, so once it has been compromised, then it is no longer secret in other sessions. In our version of the model, we were able to refine this to the weaker assumption that only the SNs who had been sent the SUPI had a secure channel.

Using more specific pre-conditions to identify honest parties provides assurance that the protocol is secure against compromised parties, but we do not claim that the conditions for these two examples were the minimal required assumptions. Developing a method to systematize these types of claims is left as a topic for future work.

3.6 Prototype

As a test of practicality, we implemented our protocol in Java over a simulated 5G system, shown in Figure 3.4. Our implementation interfaces with existing OpenID Connect identity providers: in our experiments, we tested using that of Google [38]. We plan to provide an open-source release of our implementation, as described in Appendix B.2.

3.6.1 Implementation

The components that were developed as part of our simulated network are shown within the dashed boarder in the figure. The UE hosts a web server since this provides a simple

method to construct a user interface—by allowing access through a browser—and allows integration with OpenID Connect. In 5G networks, operations such as registration and authentication are coordinated over the [NAS](#), which uses a set of pre-defined message formats [\[1\]](#). To emulate this messaging, and the associated radio link, we run the UE and [SN](#) in separate threads and allow the exchange of pre-defined messages through two queues, one for each direction of interface. For simplicity, we implemented the [HN](#) operations as a library that is called directly by the HN, as this approximates the request–response system of SN–HN communications. In practice, as 5G has a service-based architecture, both SN and HN would be composed of multiple servers that handle distinct operations [\[13\]](#).

Our implementation

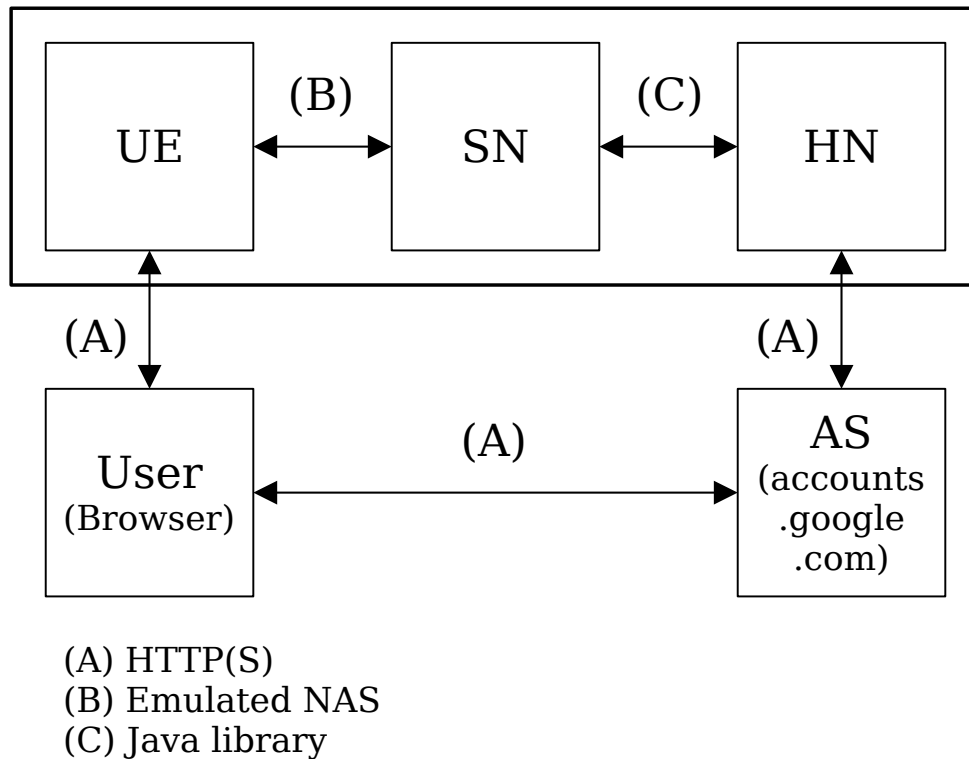


Figure 3.4: Prototype implementation of our protocol

3.6.2 Takeaways

Building our prototype required determining what concrete cryptographic primitives to instantiate in place of the generic symbols in our model. We employed the BouncyCastle [44] library for implementation of cryptographic operations. As explained in Section 3.3.1, we used the existing SUCI format to include the user’s identity in the registration request. For SUCI encryption, we implemented Profile A (ECIES using X25519 as the underlying elliptic-curve primitive) as specified by 3GPP TS33.501 [11]. To derive EK_{UE} and EK_{HN} , we used the Concatenation KDF (the one-step key derivation specified in NIST SP 800-56A [19]), since this was available in BouncyCastle.

The encryption operation modelled by Tamarin is properly instantiated as an authenticated encryption primitive since the symbolic model does not allow for ciphertext malleability. (See, for example, [15], for cases where protocols can be attacked when symbolic encryption is replaced with an unauthenticated block cipher mode). Other components of the registration request are protected by the MAC included with AUTN, but we authenticate `auth_uri` separately because this MAC is commonly implemented using the MILENAGE algorithm, which accepts only the fixed-length inputs used in AKA [14]. The generation of AUTN is performed inside the USIM [11], so it would be difficult to modify in practice.

We used AES-GCM [34] for the encryption of `auth_uri` and `redirect_uri` since it is a commonly employed authenticated encryption mode. In our protocol, each key is used for exactly one message, and the 5G-AKA sequence number system prevents replays, so using a fixed nonce, which would otherwise compromise security [37], is acceptable.

Our choices of primitives are summarized by Table 3.4. We expect that the implementation cost of the additional primitives is insignificant on modern devices, but we note one possibility for optimization: the standard ECIES profiles [11] use the ANSI-X9.93 KDF (described in [25]) for key derivation and AES-CTR followed by HMAC-SHA-256 for encryption and authentication. These primitives can be assumed to be available on any UE and reused for our protocol. This option corresponds to the “lightweight” option column of the table.

We also notice that our protocol naturally requires changes to the NAS messages and SN-to-HN requests, both of which imply that the SN must be modified to support our system. This was our expectation: our protocol involves significant changes to 5G-AKA, but it was especially apparent when implementing the protocol.

Table 3.4: Choice of cryptographic primitives

Purpose	Our Implementation	“Lightweight” option
SUCI encryption	TS33.501 Profile A [11]	TS33.501 Profile A [11]
EK derivation	Concatenation KDF [19]	ANSI-X9.63-KDF [25]
URI encryption	AES-GCM [34]	AES-CTR with HMAC-SHA-256

Chapter 4

Conclusions

4.1 Concluding Remarks

In this thesis, we presented two modified versions of the [5G-AKA](#) protocol. Chapter 2 described a method of mitigating an attack on 5G privacy by using a sequence of ephemeral identifiers in place of a fixed permanent identifier. Chapter 3 developed a composition of 5G-AKA and OpenID Connect, with the goal of providing “bring your own identity” functionality in 5G.

Both protocols were analyzed using the [Tamarin](#) theorem prover to show they met desired security properties. A common theme from both works was that a generic style of property specification could not always be directly applied: some properties required modifications in order to distinguish specific parties in their security assumptions. This suggests that there is still room for future work on methodologies for systematizing security claims. In the second work, it also was apparent that simplifications to the protocol model were required to achieve reasonable performance, so there is a frontier of protocol complexity and fidelity past which verification becomes infeasible. This requires manual judgment as to what portions of the model can be simplified without significant risk of eliminating attacks. For example, even in the base 5G-AKA model of Basin et al. [21], the [HN](#) and [SN](#) communicate over an abstract secure channel, when in practice they use an [HTTP](#) protocol sent through a secure proxy [11]. Manual effort was also required to guide the search process of the prover using [oracles](#). These factors point to the inherent

hardness of automated theorem proving—ultimately an undecidable problem—even using state-of-the-art software such as Tamarin.

We also implemented our “bring your own identity” proposal to demonstrate its feasibility using a real OpenID Connect service. This prototype was based on a simplified 5G network model, but still provided some useful insights into the considerations that would apply to a complete implementation.

4.2 Future Work

As mentioned in the previous section, one area for future work would be the development of new methods to formalize security claims that address the various limitations identified in this thesis. We make some contributions to this by proposing a typology of secrecy claims in Section 3.4.2 and an extended definition of [weak agreement](#) in Section 3.5.1; however, there is still significant scope for work that tests these definitions with other protocols and develops solutions to the other limitations noted in Sections 2.5.3 and 3.5.

Another direction for further research is the further development of “bring your own identity” in 5G. Our prototype could be extended to operate over a real 5G network, using the many open-source 5G network implementations [\[43, 54\]](#). Additionally, our proposal still retains the traditional [5G-AKA](#) key exchange between the [UE](#) and [HN](#). There is scope for more radical proposals that eliminate the HN entirely to decouple the roles of account management and network operation. This would significantly increase competition in the wireless industry, as the consumer would be able to switch between networks arbitrarily, but would also require significant changes to the assumptions of current mobile standards.

References

- [1] 3rd Generation Partnership Project; Technical Specification Group Core Network and Terminals. Non-access-stratum (NAS) protocol for 5g system (5GS). 3GPP TS 24.501 v17.5.0, 2021.
- [2] 3rd Generation Partnership Project; Technical Specification Group Core Network and Terminals. Numbering, addressing and identification. 3GPP TS 23.003 v18.1.0, 2023.
- [3] 3rd Generation Partnership Project; Technical Specification Group Core Network and Terminals. Unified data management services. 3GPP TS 24.501 v18.1.0, 2023.
- [4] 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects. Security architecture and procedures for 5G system. 3GPP TS 33.501 v15.1.0, 2018.
- [5] 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects. Study on user centric identifiers and authentication. 3GPP TR 22.904 v16.1.0, 2018.
- [6] 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects. Security architecture. 3GPP TS 33.102 v16.0.0, 2020.
- [7] 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects. Study on authentication enhancements in the 5G system (5GS). 3GPP TR 33.846 v17.0.0, 2021.
- [8] 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects. Identity management and generic authentication architecture (GAA) interworking. 3GPP TR 33.924 v17.0.0, 2022.

- [9] 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects. Network identity and time zone (NITZ). 3GPP TS 22.042 v17.0.0, 2022.
- [10] 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects. Procedures for the 5G system (5GS). 3GPP TS 23.502 v18.0.0, 2022.
- [11] 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects. Security architecture and procedures for 5G system. 3GPP TS 33.501 v17.4.1, 2022.
- [12] 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects. Study on 5G security enhancement against false base stations (FBS). 3GPP TR 33.809 v0.20.0, 2022.
- [13] 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects. System architecture for the 5G system (5GS). 3GPP TS 23.501 v18.1.0, 2023.
- [14] 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security;. Specification of the MILENAGE algorithm set: An example algorithm set for the 3GPP authentication and key generation functions f1, f1*, f2, f3, f4, f5 and f5*; document 2: Algorithm specification. 3GPP TS 35.206 v18.0.0, 2024.
- [15] Naveed Ahmed, Christian D. Jensen, and Erik Zenner. Towards symbolic encryption schemes. In Sara Foresti, Moti Yung, and Fabio Martinelli, editors, *Computer Security – ESORICS 2012*, pages 557–572, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
- [16] Android Open Source Project. Time overview. <https://source.android.com/docs/core/connect/time>, 2022. Accessed: 2023-03-14.
- [17] Myrto Arapinis, Loretta Mancini, Eike Ritter, Mark Ryan, Nico Golde, Kevin Reardon, and Ravishankar Borgaonkar. New privacy issues in mobile telephony: Fix and verification. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS '12*, page 205–216, New York, NY, USA, 2012. Association for Computing Machinery.

- [18] Jari Arkko, Karl Norrman, Mats Näslund, and Bengt Sahlin. A USIM compatible 5G AKA protocol with perfect forward secrecy. In *2015 IEEE Trustcom/Big-DataSE/ISPA*, volume 1, pages 1205–1209, 2015.
- [19] Elaine Barker, Lily Chen, Allen Roginsky, Apostol Vassilev, and Richard Davis. Recommendation for pair-wise key-establishment schemes using discrete logarithm cryptography. NIST SP 800-56A Rev 3, 2018.
- [20] David Basin, Cas Cremers, Jannik Dreier, Simon Meier, Ralf Sasse, and Benedikt Schmidt. Tamarin prover. <https://tamarin-prover.github.io/>. Accessed: 2024-07-18.
- [21] David Basin, Jannik Dreier, Lucca Hirschi, Saša Radomirovic, Ralf Sasse, and Vincent Stettler. A formal analysis of 5G authentication. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS '18*, page 1383–1396, New York, NY, USA, 2018. Association for Computing Machinery.
- [22] Ravishankar Borgaonkar, Lucca Hirschi, Shinjo Park, and Altaf Shaik. New privacy threat on 3G, 4G, and upcoming 5G AKA protocols. *Proceedings on Privacy Enhancing Technologies*, 2019(3):108–127, 2019.
- [23] An Braeken. Symmetric key based 5G AKA authentication protocol satisfying anonymity and unlinkability. *Computer Networks*, 181:107424, 2020.
- [24] Michael Burrows, Martin Abadi, and Roger Needham. A logic of authentication. *ACM Trans. Comput. Syst.*, 8(1):18–36, feb 1990.
- [25] Certicom Research. Sec 1: Elliptic curve cryptography; version 2.0. <http://www.secg.org/sec1-v2.pdf>, 2009.
- [26] Merlin Chlosta, David Rupprecht, Christina Pöpper, and Thorsten Holz. 5g suci-catchers: Still catching them all? In *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks, WiSec '21*, page 359–364, New York, NY, USA, 2021. Association for Computing Machinery.
- [27] Cisco Systems. Ultra cloud core 5G access and mobility management function, release 2021.04 – configuration and administration guide. https://www.cisco.com/c/en/us/td/docs/wireless/ucc/amf/2021-04/config-and-admin/b_ucc-5g-amf-config-and-admin-guide_2021-04/5G-AMF_Config_Guide_chapter_0111.html, 2021. Accessed: 2023-05-15.

- [28] Cas Cremers and Philip Lukert. K vs ku. Comments on GitHub issue. <https://github.com/tamarin-prover/tamarin-prover/issues/447>, 2021. Accessed: 2023-05-25.
- [29] Cas Cremers and Sjouke Mauw. *Operational Semantics and Verification of Security Protocols*. Springer Berlin, Heidelberg, 2012.
- [30] Alan DeKok. The network access identifier. RFC 7542, 2015.
- [31] William Denniss and John Bradley. EAP-TLS 1.3: Using the extensible authentication protocol with TLS 1.3. RFC 9190, 2017.
- [32] William Denniss and John Bradley. OAuth 2.0 for native apps. RFC 8252, 2017.
- [33] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
- [34] Morris Dworkin. Recommendation for block cipher modes of operation: Galois/-counter mode (gcm) and gmac. NIST SP 800-38D, 2007.
- [35] Sandra Dünki. Modelling and analysis of web applications in Tamarin. Master thesis, ETH Zurich, Zurich, 2019.
- [36] Ericsson. Ericsson mobility report: June 2024. <https://www.ericsson.com/49ed78/assets/local/reports-papers/mobility-report/documents/2024/ericsson-mobility-report-june-2024.pdf>, 2024. Accessed: 2024-07-18.
- [37] Niels Ferguson. Authentication weaknesses in GCM. 2005.
- [38] Google. OpenID Connect. <https://developers.google.com/identity/openid-connect/openid-connect>. Accessed: 2024-07-18.
- [39] J. N. Gray. Notes on data base operating systems. In R. Bayer, R. M. Graham, and G. Seegmüller, editors, *Operating Systems: An Advanced Course*, pages 465–466. Springer Berlin Heidelberg, Berlin, Heidelberg, 1978.
- [40] Dick Hardt. The OAuth 2.0 authorization framework. RFC 6749, 2012.
- [41] Maede Hojjati, Alireza Shafieinejad, and Halim Yanikomeroglu. A blockchain-based authentication and key agreement (AKA) protocol for 5G networks. *IEEE Access*, 8:216461–216476, 2020.

- [42] Ed Kanya Kiyemba Edris, Mahdi Aiash, and Jonathan Kok-Keng Loo. Network service federated identity (ns- fid) protocol for service authorization in 5g network. In *2020 Fifth International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 128–135, 2020.
- [43] Sukchan Lee. <https://open5gs.org/>. Accessed: 2024-07-18.
- [44] Legion of the Bouncy Castle Inc. Bouncy Castle – open-source cryptographic APIs. <https://www.bouncycastle.org/>, 2024. Accessed: 2024-07-18.
- [45] Bin Liu, Keping Yu, Chaosheng Feng, and Kim-Kwang Raymond Choo. Cross-domain authentication for 5G-enabled UAVs: a blockchain approach. In *Proceedings of the 4th ACM MobiCom Workshop on Drone Assisted Wireless Communications for 5G and Beyond*, DroneCom ’21, pages 25–30, New York, NY, USA, 2021. Association for Computing Machinery.
- [46] Torsten Lodderstedt, Mark McGloin, and Phil Hunt. OAuth 2.0 threat model and security considerations. RFC 6819, 2013.
- [47] G. Lowe. A hierarchy of authentication specifications. In *Proceedings 10th Computer Security Foundations Workshop*, pages 31–43, 1997.
- [48] Christian Mannweiler, Lars Christoph Schmelz, Simon Lohmüller, and Bernhard Bauer. Cross-domain 5G network management for seamless industrial communications. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pages 868–872, 2016.
- [49] Jim Martin, Jack Burbank, William Kasch, and David L. Mills. Network time protocol version 4: Protocol and algorithms specification. RFC 5905, 2020.
- [50] Simon Meier. *Advancing automated security protocol verification*. PhD thesis, ETH Zurich, 2013.
- [51] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. The TAMARIN prover for the symbolic analysis of security protocols. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification*, pages 696–701, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
- [52] Meta for Developers. Manually Build a Login Flow. <https://developers.facebook.com/docs/facebook-login/guides/advanced/manual-flow/>. Accessed: 2024-07-04.

- [53] Jorge Munilla, Mike Burmester, and Raquel Barco. An enhanced symmetric-key based 5G-AKA protocol. *Computer Networks*, 198:108373, 2021.
- [54] OpenAirInterface.org. <https://openairinterface.org/>. Accessed: 2024-07-18.
- [55] Shinjo Park, Altaf Shaik, Ravishankar Borgaonkar, and Jean-Pierre Seifert. White rabbit in mobile: Effect of unsecured clock source in smartphones. In *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '16, page 13–21, New York, NY, USA, 2016. Association for Computing Machinery.
- [56] Shinjo Park, Altaf Shaik, Ravishankar Borgaonkar, and Jean-Pierre Seifert. Anatomy of commercial IMSI catchers and detectors. In *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*, WPES'19, page 74–86, New York, NY, USA, 2019. Association for Computing Machinery.
- [57] Julian Parkin and Mahesh Tripunitara. Countering SUCI-catchers in cellular communications. <https://github.com/Julian2n7000/countering-suci-catchers-model/>.
- [58] Rahul Punchhi. An implementation of 5G-AKA and a usability analysis of OpenLDAP access control lists (ACLs). Master's thesis, 2021.
- [59] Ahmad Rostami, Deepak Nair, Malgorzata Svensson, and Jan Backman. 5G for enterprise networking. <https://www.ericsson.com/49290c/assets/local/reports-papers/white-papers/2024/5g-for-enterprise-networking.pdf>, 2024. Accessed: 2024-06-15.
- [60] Negar Sabour. Static verification of an implementation of 5G-AKA. Master's thesis, 2023.
- [61] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore. OpenID Connect core 1.0 incorporating errata set 2. https://openid.net/specs/openid-connect-core-1_0.html. Accessed: 2024-06-26.
- [62] Nat Sakimura, John Bradley, and Naveen Agarwal. Proof key for code exchange by OAuth public clients. RFC 7636, 2015.
- [63] Bernardo Santos, Van Thuan Do, Boning Feng, and Thanh van Do. Towards a standardized identity federation for internet of things in 5G networks. In *2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of*

People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), pages 2082–2088, 2018.

- [64] The Tamarin Team. Model specification using rules. https://tamarin-prover.com/manual/master/book/005_protocol-specification-rules.html, 2016. Accessed: 2024-07-18.
- [65] The Tamarin Team. Tamarin prover manual: Advanced features. https://tamarin-prover.com/manual/develop/book/011_advanced-features.html, 2016. Accessed: 2024-07-18.
- [66] The Tamarin Team. Tamarin prover manual: Property specification. https://tamarin-prover.github.io/manual/book/007_property-specification.html, 2016. Accessed: 2024-07-18.
- [67] Awaneesh Kumar Yadav, Manoj Misra, Pradumn Kumar Pandey, An Braeken, and Madhusanka Liyange. An improved and provably secure symmetric-key based 5G-AKA protocol. *Computer Networks*, 218:109400, 2022.
- [68] Ilsun You, Gunwoo Kim, Seonghan Shin, Hoseok Kwon, Jongkil Kim, and Joonsang Baek. 5G-AKA-FS: A 5G authentication and key agreement protocol for forward secrecy. *Sensors*, 24(1), 2024.
- [69] Kaifeng Yue, Yuanyuan Zhang, Yanru Chen, Yang Li, Lian Zhao, Chunming Rong, and Liangyin Chen. A survey of decentralizing applications via blockchain: The 5G and beyond perspective. *IEEE Communications Surveys & Tutorials*, 23(4):2191–2217, 2021.

APPENDICES

Appendix A

Complete 5G-AKA key derivation

A.1 Security Anchor Function Key

As the [USIM](#) procedure is reused from 3G, it produces two more keys [6]:

$$CK \leftarrow f_3(K, \text{RAND}) \quad IK \leftarrow f_4(K, \text{RAND})$$

Originally, these were the ciphering key and integrity key, but in [5G-AKA](#), they are concatenated to form one key used for further key derivation [11]. K_{SEAF} is derived via an intermediate key derivation step.

$$\begin{aligned} K_{\text{AUSF}} &\leftarrow \text{KDF}(CK \| IK, 0\text{x}6\text{A}, \text{SNname}, \text{SQN}_{\text{ENC}}) \\ K_{\text{SEAF}} &\leftarrow \text{KDF}(K_{\text{AUSF}}, 0\text{x}6\text{C}, \text{SNname}) \end{aligned}$$

A.2 Authentication Response

The expected authentication response is actually treated as input to a [KDF](#), in order to create a modified response of longer bit length bound to the [SN's](#) name. The [UE](#) produces its response in the same fashion.

$$\begin{aligned} \text{XRES} &\leftarrow f_2(K, \text{RAND}) \\ \text{XRES}^* &\leftarrow \text{KDF}(CK \| IK, 0\text{x}6\text{B}, \text{SNname}, \text{RAND}, \text{XRES}) \end{aligned}$$

Appendix B

Source Code Releases

B.1 Tamarin Model of the SUCI-Catcher Mitigation

This model is publicly available at [57]. In this repository, the theory file can be found at `CounteringSUCICatchers/src/5G_AKA_with_hsupi.spthy`. Two [oracles](#) are included in the same directory: `5G_AKA_with_hsupi.pl` and `5G_AKA.oracle`; the latter is the original oracle from the model of Basin et al. [21]. The repository also contains some utility scripts for automatically executing [Tamarin](#) to verify the theory. Details can be found in the README file.

B.2 Implementation of “Bring Your Own Identity”

This project will be released in the same repository as other prior work on 5G-AKA [58,60]. It will be included under the directory `workspaceGradle/project5gAka/5g-aka-byoi`. In this directory, the main subdirectories are

- `tamarin`—The [Tamarin](#) model of the protocol. The model can be found under `tamarin/src/byoi_stg1.spthy.m4`. Running the makefile under the parent directory will output the theory file under `tamarin/thy`. This model requires a recent version of Tamarin that includes support for natural numbers and inline oracles.
- `ue-emulator`—The model implementation of the protocol, including the [UE](#) (implemented as a web interface), and network. Instructions for running the implementation can be found in the README file of this directory.

- **shared**—A library for base 5G cryptographic functions such as authentication vector generation or [SUCI](#) encryption. A test suite is included under **shared/src/test**.

Glossary

- access token** In OAuth 2.0, a token that allows the client to access user-owner resources. [35](#), [37](#), [38](#), [41](#), [43](#), [46](#)
- action fact** In Tamarin, a fact that represents an event and forms part of the trace rather than current state. [8](#), [11](#), [12](#), [27](#), [51](#)
- aliveness** An authentication property that only requires that the responder has been running the protocol. [10](#)
- authorization code** In OAuth 2.0, a code that can be exchanged for an access token by the client. [37](#), [43](#), [52](#)
- client** In OAuth 2.0, a third party who requests access to resources controlled by the AS. [35](#), [37](#), [38](#), [42](#)
- ID token** In OpenID Connect, an additional token sent in the access token response that attests to the user's identity. [38](#)
- injective agreement** An authentication property that extends weak agreement to also require agreement on a data item, and an injective correspondence between initiator and responder sessions. [10](#), [11](#), [43](#), [50](#)
- lemma** In Tamarin, a statement in first-order logic about the set of all traces. [8–10](#), [12](#), [27](#), [29](#), [30](#), [32](#)
- non-injective agreement** An authentication property that extends weak agreement to also require agreement on a data item. [10](#), [11](#), [30](#), [42](#), [43](#), [50](#)
- oracle** In Tamarin, a ranking of proof goals. [9](#), [29](#), [56](#), [67](#)

partial deconstruction In Tamarin, when the sources of a fact could not be enumerate during precomputation. [9](#), [30](#)

persistent fact In Tamarin, a fact that is not consumed when used as the input of a rule, in contrast to an ordinary (linear) fact. [7](#)

sources lemma In Tamarin, a lemma assumed during a second precomputation phase that attempts to resolve partial deconstructions. [9](#), [10](#), [30](#)

Tamarin Theorem prover tool for analysis of cryptographic properties. [1](#), [7–12](#), [24](#), [26](#), [29](#), [30](#), [32](#), [43](#), [49–51](#), [54](#), [56](#), [57](#), [67](#)

trace In Tamarin, a sequence of valid applications of the multiset-rewriting rules that specify a protocol. [8](#), [9](#)

weak agreement An authentication property that requires that the responder has been running the protocol and agrees on the identity of the initiator. [10](#), [11](#), [42](#), [43](#), [46](#), [49](#), [50](#), [57](#)