Kenneth Gabrielsen

Private Identification Schemes for 5G IoT Networks

July 2021

NTNU
Norwegian University of
Science and Technology

Private Identification Schemes for 5G IoT Networks

Kenneth Gabrielsen

Master of Science in Communication Technology
Submission date: July 2021
Supervisor: Stig F. Mjølsnes
Co-supervisor: Ruxandra F. Olimid

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

Title:	Private Identification Schemes for 5G IoT Networks
Student:	Kenneth Gabrielsen

Problem description:

This work will investigate the security problem of *private identification*, in particular where mobile user equipment reports its identity to the core network. This problem is clearly exhibited by IMSI catcher active attacks. Recently, the 5G standard has introduced a public key cryptography-based solution for identity privacy. However, simpler resource-constrained Internet-of-Things devices may not easily realize public-key cryptography, therefore more lightweight solutions should be sought.

Analyze and compare state-of-the-art lightweight cryptography schemes for private identification in IoT mobile networks. The study should include concrete performance analyses related to real-world devices and protocols, both based on published reports and own experimental testing.

Date approved:	2021-03-18
Responsible Professor:	Stig F. Mjølsnes, IIK
Supervisor:	Stig F. Mjølsnes, IIK
Co-supervisor:	Ruxandra F. Olimid IIK

Abstract

This thesis explores the problem of *Private Identification* pertaining to the introduction of *Internet of Things* (IoT) devices to the new *5G Mobile network*. The current solution in 5G for identity privacy relies on public-key cryptography, and it is uncertain if this is reliable for IoT devices. This thesis specifies requirements for a private identification protocol that sufficiently guarantees the *privacy of a subscriber*, without it impacting IoT use cases in terms of *performance*. An analysis of proposed *lightweight alternatives* is then done to cover the informal security requirements for a suitable solution. To evaluate performance, an experimental framework was created that greatly simplifies the process of benchmarking cryptographic primitives and schemes. This is then used to gather experimental results on the performance of the current 5G solution and proposed alternatives. Deductions are then made based on the informal analysis and experimental results of the schemes. The 5G solution is found to be sufficiently performant for Massive IoT use cases without high mobility. All but one of the symmetric solutions had sufficient identity privacy for mobile networking. However, the remaining scheme did not have sufficient reliability. The proposed schemes also suffer from a lack of understanding of how identification is done in 5G-AKA, which limits compatibility and cryptographic flexibility.

Sammendrag

Oppgaven tar utgangspunkt i problemet med privat identifisering for *Internet of Things* (IoT) enheter når de skal brukes i det nye 5G mobilnettet. Løsningen i 5G for å holde identifiserbar informasjon hemmelig benytter seg av offentlig-nøkkel kryptografi. Med tanke på ytelse, så er det usikkert om dette er en gunstig løsning for IoT enheter. Krav til en gunstig protokoll som ivaretar privat identifisering i 5G mtp. ytelse for IoT blir først spesifisert. Disse kravene blir brukt som utgangspunkt i analyse av en rekke symmetriske protokoller som kan erstatte den nåværende 5G løsningen. For å evaluere ytelsen til disse protokollene, så ble et eksperimentelt rammeverk laget. Rammeverket forenkler arbeidet med referansemåling av kryptografiske protokoller. Analysen og de eksperimentelle resultatene blir da brukt som grunnlag for å svare på oppgaveproblemet. Løsningen brukt i 5G for privat identifisering er tilstrekkelig i ytelse for å håndtere Massiv IoT bruksområder med liten mobilitet. Alle untatt en av de symmetriske protokollene har derimot sikkerhetsvakheter som gjør dem utsatt for en aktiv angreper. Den gjenstående protokollen er ikke påligelig nok å bruke i mobilnettet. De symmetriske forslagene lider også av en utilstrekkelig forståelse av hvordan identifisering blir gjort i 5G-AKA protokollen, som begrenser kompatibiliteten og den kryptografiske fleksibiliteten til 5G-AKA.

Preface

It is assumed that the reader possesses basic knowledge of cryptography. Simple knowledge of the mobile network may also help at times.

This thesis required working with multiple programming languages, different hardware, and a bunch of libraries and software. Initially, I did not know how to program with low-level hardware, so that was definitely a learning process. That required separating the experimental framework into two design stages, where I first just made something that worked. The next version is a lot more streamlined, so if anyone else wants to do something similar, they could use this work as a basis to hopefully dodge most of the issues I encountered.

At times I struggled with a lack of direction and scope with the thesis. My supervisors Stig Frode and Ruxandra were definitely helpful in giving me a sense of direction in those moments. Thank you.

I could not have completed this thesis without such a strong support group. I am grateful to my parents for the love and understanding they gave me during these difficult times. My friends' support was also a much-needed relief when I was tired of everything and close to giving up. Thank you.

Contents

List of Figures

List of Tables

List of Acronyms

5G-AKA	5G Authentication and Key Agreement.
DDoS	Distributed Denial-of-Service.
ECC	Elliptic-Curve Cryptography.
ECIES	Elliptic Curve Integrated Encryption Scheme.
GUTI	Globally Unique Temporary Identifier.
HN	Home Network.
HNI	Home Network Identifier.
IMSI	International Mobile Subscriber Identity.
IoT	Internet of Things.
KDF	Key Derivation Function.
PID	Private Identification.
PKC	Public-Key Cryptography.
RNG	Random Number Generator.
SN	Serving Network.
SUCI	SUbscription Concealed Identifier.
SUPI	SUbscription Permanent Identifier.
TMSI	Temporary Mobile Subscriber Identity.

UE User Equipment.

USIM Universal Subscriber Identity Module.

VN Visiting Network.

Chapter 1

Introduction

The Internet of Things (IoT) growth has been staggering over the last few years, and this trend is expected to continue. Statista forecasts that there were 30.7 billion IoT devices in 2020, which is expected to grow to more than 75 billion by 2025 [Sta]. Using these numbers, Safaei et al. anticipate that there will be 9 smart devices *per person* by 2025 [SMBE17]. Such growth spurs a significant increase in sent data that may involve *identifiable information* about individuals or the device itself.

Meanwhile, the 5G network is in the process of being rolled out in several countries. In Britain, it is expected that the new mobile network will cover 90% of the population by 2027 [OF18]. 5G also contains new security specifications on how to handle user identity privacy. Contrary to previous mobile network generations, it ensures the confidentiality of a user's identity by using public-key cryptography [rGPP18]. This is a viable solution for commodity mobile devices [JNNN17]. But it is also expected that the 5G network will facilitate communication between IoT devices. Public-key solutions typically do not work well with computationally constrained devices, which may impede the onboarding of IoT to the 5G Mobile Network. This problem has spurred on several lightweight symmetric proposals for handling identity privacy in the 5G network.

1.1 Motivation

To facilitate further understanding of recently proposed symmetric protocols and how they fit with the 5G security specifications is of main importance in this thesis. Their different means to achieve identity privacy for the 5G network create interesting trade-offs in regards to security and performance. Understanding their differences and drawbacks could be helpful for designing better solutions that achieve identity privacy. Jimenez et al. also highlights the need to evaluate the performance of the current 5G solution for identity privacy on commodity IoT devices [JNNN17].

1.2 Research

This section details the relevant thesis research questions and associated research objectives.

The main research goal is to comprehensively assess recent lightweight protocols suitable to maintain identity privacy in mobile networking [Cho21, KDM18, PGGM20, GSZ19, Bra20]. This assessment includes both security analysis and performance analysis adhering to mobile network user privacy requirements and IoT performance requirements. As such, the current 5G solution for this problem is also evaluated for comparison.

1.2.1 Research Questions

- **RQ1.** To what degree is the current 5G identity privacy scheme suitable for IoT devices?
- **RQ2.** To what degree do the lightweight alternatives substitute the current solution in regards to both security and performance?
- **RQ3.** What are the possible trade-offs and differences between these lightweight solutions?
- **RQ4.** Do the proposed solutions minimally interfere with current 5G security specifications and architecture?

1.2.2 Research Objectives

- **RO1.** Analyze the current 5G Authentication and Key Agreement (5G-AKA) protocol and its identification scheme and create a set of requirements suitable for a 5G Private Identification Protocol.
- **RO2.** Extend this protocol to cover requirements for onboarding IoT devices in regards to performance and security.
- **RO3.** Do security analysis of proposed lightweight schemes for private identification in mobile networking.
- **RO4.** Create an experimental framework for testing the proposed schemes.
- **RO5.** Use the framework to evaluate proposed scheme performance on commodity IoT devices.
- **RO6.** Use scheme security analysis and experimental results to answer research questions.

1.3 Methodology

The methodology used for this thesis can be divided into four parts. The first is research into recent private identification protocols, their applicability to mobile networks, and their applicability for lightweight embedded devices. This is done through a literature study of relevant papers, technical specifications regarding 5G, and publications with the relevant scheme proposals.

The second is the development of an experimental framework for evaluating the performance of the proposed schemes.

The third is an experiment carried out by using the aforementioned framework. The experiments are done on several IoT devices varying in computational power, reflecting the current IoT market. This will evaluate the schemes based on their performance.

The fourth is a discussion of the applicability of the schemes using the literature study and the results from the experiments, where they are matched according to a proposed set of necessary security and performance requirements for a 5G IoT Private Identification Protocol.

1.4 Outline

This thesis consists of 6 chapters, including this one.

Chapter 2 presents the Private Identification problem and solution, the Private Identification protocol, in a general setting. Some trivial solutions are also presented.

Chapter 3 presents the problem in mobile networks, the historical solutions, and the current solution proposed for the 5G mobile network. It extends the standard protocol requirements to be applicable in mobile networking.

Chapter 4 presents the problem in the context of IoT and mobile networking, where the protocol requirements are extended again to be applicable for IoT devices using the mobile network. The proposed schemes and security analysis is also done here.

Chapter 5 details the experimental framework and its functionality. It also presents the experimental setup using the framework.

Chapter 6 details the experimental results gathered using the framework.

Chapter 7 Summarizes the security analysis and experimental results and answers the research questions.

Chapter 8 Concludes the thesis by summarizing the accomplished work and points out some future research that could be done.

Chapter 2

Private Identification

This chapter gives an overview of the Private Identification problem, along with an explanation of a proposed general protocol to handle the problem, and relevant security requirements for it.

2.1 The Private Identification Problem

This problem pertains to the scenario where a client entity, with no secure communication established, needs to register its identity to a server entity over a communication system. This is either just done to register said client entity or as the setup for authentication to facilitate further communication.

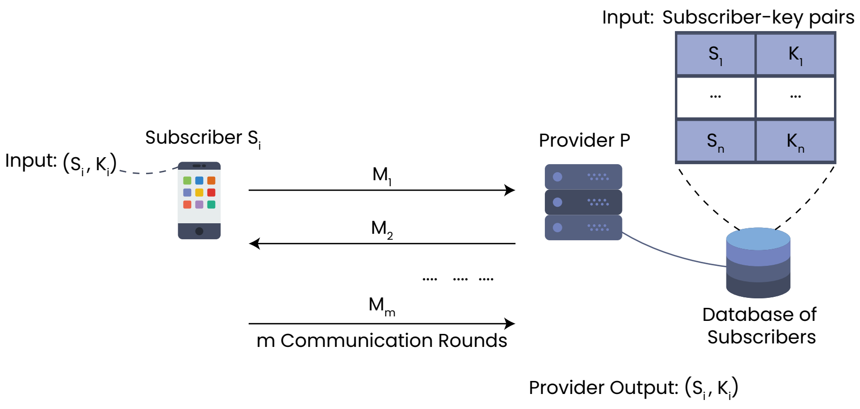


Figure 2.1: Subscriber sends concealed S_i to provider P, who then derives the identity. Figure inspired from protocol overview in [MO17b].

The client entity is known as the subscriber, and the server entity is known as the provider. This relationship is one-to-many, where a provider has to handle several subscribers concurrently. Subscriber i knows of its identity S_i and may also possess a cryptographic subscriber key K_i that could be used to establish secure communication with the provider. The provider has a register of all the subscriber identities and their associated subscriber key, as seen in Figure 2.1.

The subscriber key cannot initially be used to establish communication, as the provider would have no way of knowing which cryptographic key to use for decryption. The subscriber needs to transmit its identity so that the provider can select the proper subscriber key.

However, transmitting this identity would compromise the privacy of the subscriber. The identity could, for instance, be used to track the location of the subscriber.

2.1.1 The Private Identification Problem

Mjølsnes and Olimid define this problem, specifically for mobile networks, as the Private Identification Problem [MO19] [MO17b].

How can we construct efficient and scalable private identification mechanisms in (mobile) communication systems? More concrete, how can a device identify itself to the (mobile) network) while never disclosing its permanent identity to an adversary?

2.1.2 The Private Identification Protocol

Mjølsnes and Olimid define the Private Identification Protocol (PIP) in symmetric settings [MO17b]:

1. Input:
   - Each subscriber $S_i, 1 \leq i \leq n$ owns a pair (S_i, K_i)
   - Provider P owns the pairs associated to all subscribers $(S_i, K_i)_{1 \leq i \leq n}$
2. A subscriber S_i runs one or more instances of PIP with the service provider P
3. Output:
   - P learns the identity S_i
   - For an adversary A, PIP is secure.

They also state the following informal security requirements, quoted directly [MO17b]:

- **Privacy**. The protocol must not leak any information that can help identify the subscriber or learn the private shared key.
- **Unlinkability**. Two or more message exchanges cannot be linked to the same subscriber
- **Protection against location disclosure or tracking**. Tracking a subscriber's location should not be possible.
- **Protection against cloning and impersonation**. The subscribers must not be prone to cloning or impersonation after one or more runs of the private identification protocol. Impersonation performed by replay attacks could be up to some point avoided by subsequent authentication mechanisms.
- **Efficiency**. The private identification protocol must comply with speed, computational power, number of rounds, etc., with the requirements.
- **Scalability**. The identification protocol must be scalable to a substantial amount of subscribers (hundreds of millions).

Thesis terminology

This thesis uses slightly different abbreviations for simplicity:

- Private Identification (PID).
- Private identification Problem becomes PID Problem.
- Private Identification Protocol becomes PID Protocol.

2.2 Trivial Solutions

This section describes some trivial solutions to private identification. They each have some significant flaw that prevents them from being used.

2.2.1 Key Search

A *Key Search* solution trivially creates a private identification scheme with a computational trade-off. A subscriber S_i chooses a random value ,say r_i, and encrypts it using the subscriber key K_i. The subscriber then transmits r_i and its encrypted value to provider P. Provider P will then iterate through all (S_i, K_i) subscriber-key pairs and decrypt the encrypted r_i using each key K_i, checking if the sent r_i value matches with the decrypted value. If they match, the corresponding S_i is chosen as the subscriber identity.

This is a computationally expensive solution that is linearly bound, as it iterates through all subscribers. Variations of this solution may not necessarily use the subscriber key. Hashing could also be used, where S_i is hashed with value r_i, the hashed value, and r_i are transmitted. The Provider P would then need to iterate through all identities and do the same hashing to identify the correct S_i.

Solutions may also choose to improve upon existing scalability by using different data structures. The identities could be organized into binary trees, which would improve the identity search from linear to logarithmic. However, this incurs some penalties. Some additional parameters that could be used to identify the subscriber must also be sent over the communication channel. These additional parameters are necessary for properly indexing the relevant data structure used. Another consideration is added complexity to the solution, which could be significant for existing architectures that have to be revamped to accommodate new data structures.

2.2.2 Public-key Cryptography

Public-Key Cryptography (PKC), also known as *Asymmetric Cryptography*, can ensure confidentiality of the identity and authentication of the provider. In a PKC cryptosystem, there are two separate keys used for encryption and decryption. A user X generates a *Public Key* used for encryption, which is kept accessible to anyone, and a *Private Key* kept secret, used for decryption. The user Y then uses the public key of X to encrypt a message. This message can only be decrypted and read by X. The security of a PKC solution is derived from the hardness and intractability of certain mathematical problems.

Preliminary

Take, for instance, integer factorization. Given two very large prime numbers, p and q, where $p < q$. They are each quantified by k_p and k_q bits. Say both of them are 1536 bits long. Then, $p \times q = N$, where N is roughly of size $k_p + k_q$. This will produce a 3072-bit number. This multiplication is done in milliseconds for a commodity computer, and even with no optimization, the time complexity is $\mathcal{O}(k_p \times k_q)$, a roughly quadratic increase of operations for each bit. However, finding the prime factors p and q from N is computationally expensive. The factorization, even with optimized factorization algorithms like *Pollard Rho*, has a time complexity of $\mathcal{O}(\sqrt{N})$, the number of operations growing exponentially with each bit [Pol75]. For the same commodity computer, it would take trillions of years to find the factorization. In fact, the largest factorized number, RSA-250, in the aforementioned format, is "only" 829-bits long. The factorization utilized approximately 2700 CPU-core years using a conventional high-end server CPU (*2.1Ghz Intel Xeon Gold 6130*) as a reference [BGG$^+$20].

RSA is a PKC cryptosystem that builds on this idea of factoring being a hard problem [RSA78]. The user first finds any two suitably large prime numbers p and q, which form the basis of a private key. The modulo N, made by multiplying p and q, is used as a basis for the public key. The public key is made accessible, and the private key is kept secret. The different security strengths are then denoted with N-bit RSA. Using the example from the previous paragraph would make them 3072-bit RSA keys, and it would have the equivalent security of a 128-bit symmetric key in classic security settings [Bar20].

Application to the Problem

The provider P is in possession of a public-private key pair (PK_P, SK_P). Each subscriber is registered with and can use PK_P for encryption. A subscriber S_i uses PK_P to encrypt its identifier: $Enc_{PK_P}(S_i)$, and sends the encrypted identity to the provider. The provider decrypts using SK_P to get $S_i = Dec_{SK_P}(Enc_{PK_P}(S_i))$.

Although this offers an elegant solution to the problem, it comes with the necessity of a *Public Key Infrastructure* (PKI), which could be burdensome for certain applications of the problem. The encryption process may also be burdensome for computationally constrained devices without lowering the security strength to inadequate levels. The public keys of each provider also have to be provisioned securely, and it may be difficult to update the public key of a provider and delegate the new key to each subscriber. The security is also reliant on a single private key, and if compromised, it would require manual intervention with every subscriber device to update to a new provider public key.

2.2.3 Pseudonyms or Temporary Identifiers

Instead of a subscriber sending its long-term identifier S_i, the subscriber sends a pseudonym for S_i, say TID_i, which acts as a temporary identifier: the provider stores all temporary identifiers and their long-term identifier associations. Hence, TID_i is used as the index to determine the actual subscriber identity S_i.

Some solutions partially apply this solution by having the provider periodically refresh a pseudonym for a subscriber and send it to the subscriber after establishing secure communications. This is the case for mobile networks. A different application of this solution category is to have the pseudonym change for every identification phase. This is accomplished by the subscriber and provider sharing temporary parameters that update after every identification. These temporary parameters are then used to derive the pseudonym. However, this requires the subscriber and provider to have the parameter values synchronized. If the subscriber's parameters start to deviate from the provider's, de-synchronization will happen, as the provider cant derive the identity. To remedy this, a re-synchronization mechanism is necessary. This is often a burdensome process that causes privacy leakage. Furthermore, these solutions can usually be forced to de-synchronize by an adversary.

Chapter 3

Private Identification for Mobile Networks

The subscriber-provider model is used in mobile networks to ensure secure subscriber-provider communication across heterogeneous mobile networks. This chapter will give necessary preliminaries for how a mobile network operates. The amendments to the PID Protocol necessary for mobile networking are described. How private identification is handled by mobile networks, both legacy and current in 5G, is described. Finally, the private identification scheme used in 5G is analyzed.

3.1 Preliminaries

This section is necessary for understanding the mobile network in context to the problem of private identification.

3.1.1 Mobile Communications Network Architecture

This subsection describes the architecture of a *Mobile Communications Network*. See Figure 3.1 as a guideline. This communication network contains multiple distinct providers. Hence, it is important to note that a subscriber could now communicate with a different provider, then route the communication to the subscriber's provider. This communication needs to be facilitated through a three-tier architecture.

Mobile Network

A communication network that provides services like voice and data to subscribers over a wireless link. Network coverage is achieved by organizing land areas into *cells*, where one cell is served by a transceiver, known as a *base station*.

User Equipment

A subscriber uses its User Equipment (UE) to connect to the base station with the strongest signal. Also known as *Mobile Equipment* when referring to mobile units. IoT Devices will eventually be more prevalent on the 5G Mobile Network.

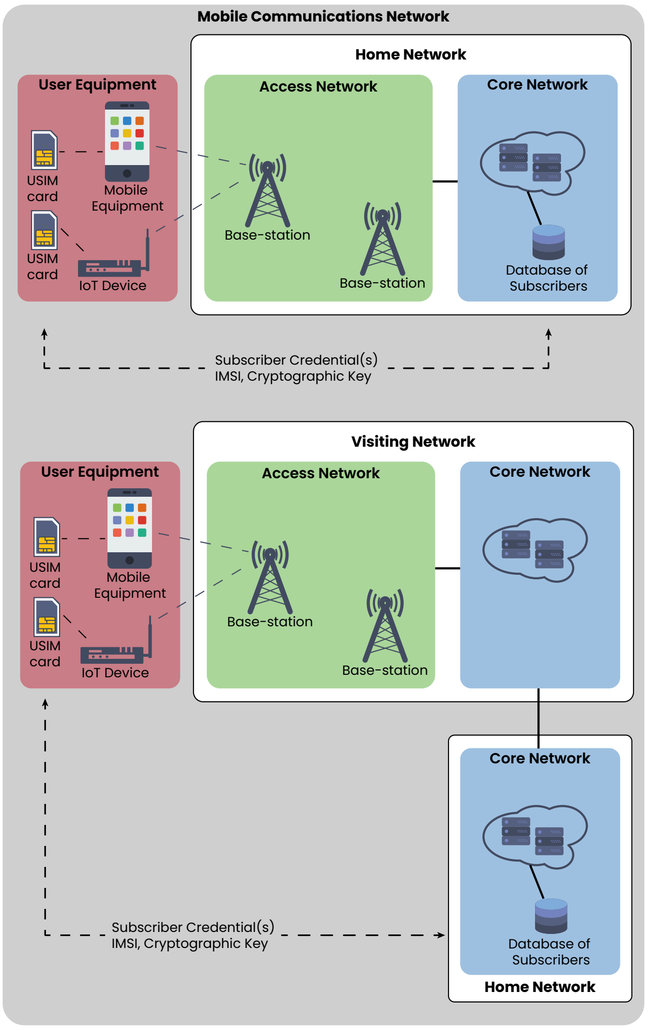


Figure 3.1: Three-tier mobile network architecture. Top: Access via Home network where Serving Network is HN. Bottom: Roaming via a visiting network. Figure adapted from existing figure in [MO19].

Universal Subscriber Identity Module Card

The Universal Subscriber Identity Module (USIM) card is a tamper-proof IC card within the UE. It has basic cryptographic capabilities and stores *subscriber credential(s)* that matches the profile of one subscriber.

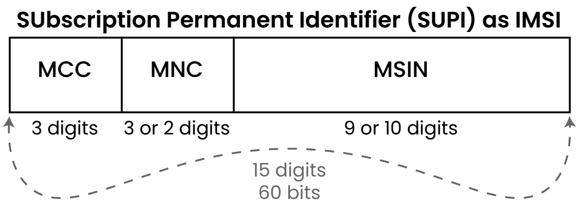


Figure 3.2: SUPI structure. All digits are 4 bit BCD encoded.

Network Operator

The provider to a subscriber in the context of mobile networks.

Home Network

The Home Network (HN) home environment of the subscriber. The UE establishes end-to-end communication with the HN. Subscriber credentials are stored here for every subscriber under the operator. The HN is responsible for providing service to the subscriber. The Core network comprises multiple physical components that can communicate with each other to provide subscriber services.

Serving Network

The UE communicates with a base station, which is part of the Serving Network (SN). It is used as an access point to the HN. Usually belongs to the same operator as the subscriber, which would make it the home network. It is responsible for routing the communication to the home network.

Visiting Network

If UE has roaming enabled, then the UE can establish connections with serving networks that belong to different operators. In that instance, the serving network is known as the Visiting Network (VN).

IMSI/SUPI

The International Mobile Subscriber Identity (IMSI) uniquely identifies a subscriber for all mobile networks. It is stored in the USIM and HN. The HN uses it as a tag to look up subscriber credential(s) in the subscribers' database. In the 5G network, the identifier is known as SUbscription Permanent Identifier (SUPI), for which there are two main types, one of them being The IMSI. This thesis considers the SUPI and IMSI to be equivalent without loss of generality.

Figure 3.2 visualizes the structure of the SUPI. The *Mobile Country Code* (MCC) indicates the geographic region, and *Mobile Network Code* (MNC) identifies an operator. Together, they uniquely identify a mobile network operator. The *operator uses the Mobile Subscription Identification Number* (MSIN) to identify a subscriber within its own network uniquely.

Subscriber Key

A cryptographic key on the USIM and in the HN. Upon UE establishing a connection with HN, the HN will use the received IMSI to lookup the subscriber key. This key is then used for further authentication and subsequent secure communication.

3GPP

Third Generation Partnership Project (3GPP) is an umbrella term that unites 7 telecommunication standard development organizations. They are responsible for releasing all mobile network-related standard specifications.

3.1.2 Subscriber Identification preceding 5G

This subsection describes how identification was accomplished before 5G, for which private identification was insecure. 4G LTE is used as an example without loss of generality, as this problem persists until 5G.

TMSI/GUTI

The Temporary Mobile Subscriber Identity (TMSI) is used as a temporary identifier in mobile networks such that it can be sent instead of the IMSI when connecting to a network. This identifier is periodically refreshed and is impossible to link back to the subscriber's IMSI. This allows the subscriber to initialize communications without compromising its identity. The TMSI is contained within a Globally Unique Temporary Identifier (GUTI) in LTE. This is simply a wrapper with some extra signaling information. The 5G mobile network uses 5G-GUTI as the temporary identifier.

Initial Request

Upon first connecting to any mobile network, the UE has not been provisioned with a TMSI. Hence, it must transmit its IMSI in the clear over the radio access link. This would allow any passive eavesdropper to capture the IMSI.

IMSI Catcher

IMSI catching is a viable attack against all mobile networks preceding 5G [MO17a]. It works by utilizing two base stations. For instance, two affordable SDR devices. One base

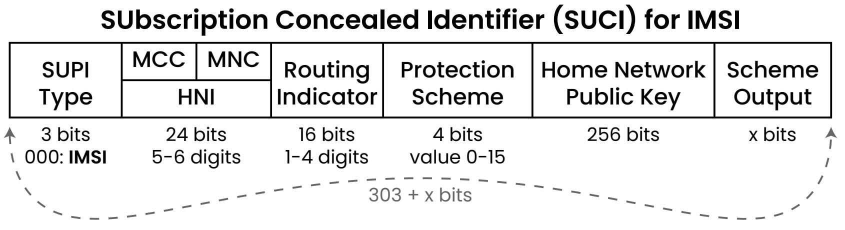


Figure 3.3: SUCI structure.

station jams a real serving network's signal in the vicinity, while another base station masquerades as the jammed serving network. In effect, any subscriber in the area will attempt to connect to the fake base station instead of the real one. However, this base station does not have a valid TMSI for the subscriber. This causes the subscriber to send its IMSI in plaintext to the fake base station, "catching" the IMSI. If the subscriber has its UE set to roaming mode, it will greatly simplify this attack to just requiring one base station.

3.1.3 5G-AKA

This subsection describes the relevant parts of the 5G-AKA. It was initially specified in release 15 of the 5G Security Specifications by 3GPP, June 2018, which will henceforth be referred to as TS 33.501 [rGPP18]. Clause 6.1.3 details the two different authentication procedures: EAP-AKA and 5G-AKA. For this problem, we focus on 5G-AKA without loss of generality.

SUCI

IMSI catchers do not work in 5G because instead of the UE sending a SUPI in clear upon request, it will be encrypted and sent in a SUbscription Concealed Identifier (SUCI).

The structure is visualized in Figure 3.3. SUPI type refers to either IMSI or NAI. The *Home Network Indicator* (HNI) is composed of the MCC and MNC, uniquely identifying the subscriber's network operator. This credential also resides within the USIM of every subscriber. The routing indicator is used for signaling purposes within the home network and is not relevant to the problem. The *Scheme Output* contains the encrypted SUPI and additional output necessary for other security requirements. The *Protection Scheme* determines which scheme should be used to produce the scheme output. The *Home Network Public Key* is contained within the USIM and is the public key used for communication with the home network.

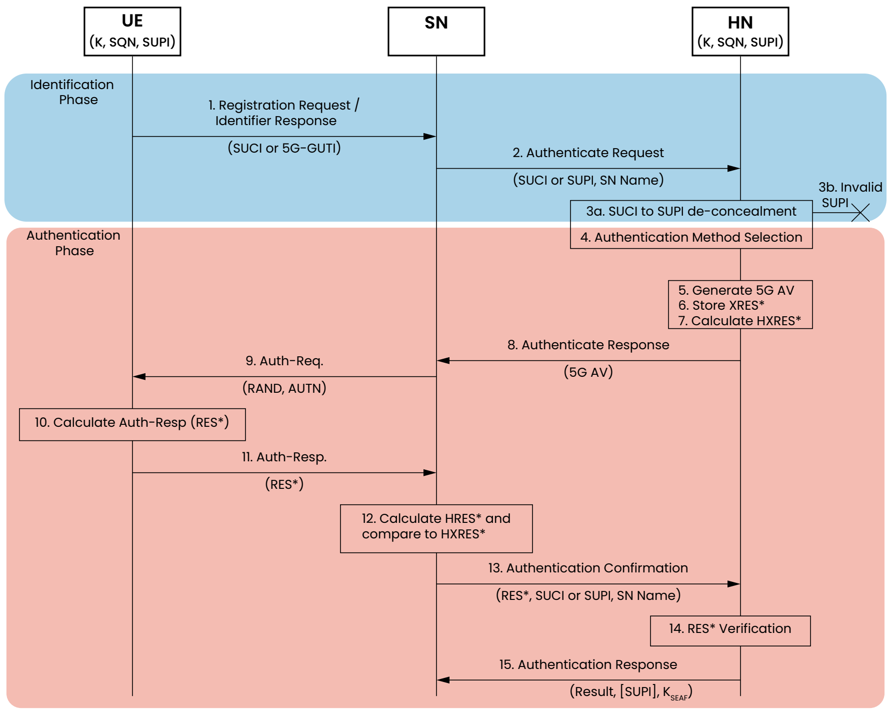


Figure 3.4: 5G AKA with identification. Figure adapted from sequence diagram in TS 33.501 [rGPP18].

Identification Phase

Figure 3.4 visualizes the entire 5G-AKA process for a UE when it initiates a new connection to a serving network. The Identification Phase is of main concern in this thesis.

The goal of the identification phase is to acquire the SUPI of a subscriber without compromising the subscriber's privacy. The SUPI needs to be encrypted and sent to the HN. The process is described below:

1. The UE sends a Registration Request to the SN. If the UE is already allocated a temporary identifier 5G-GUTI, then it will send that to the SN. Otherwise, it needs to produce a valid SUCI with one of the 5G protection schemes and send that.
2. Check HNI and redirect message to the HN, along with unique id for the sending SN. If SN receives 5G-GUTI, it will be mapped to the SUPI.
3. HN utilizes a *Subscription Identifier De-concealing Function* (SIDF) to decrypt the SUCI scheme output to its SUPI. This is known as de-concealing the SUCI. If the result is invalid, it gets discarded, and authentication halts.

Authentication Phase

The goal of the authentication phase is to establish secure communications with the UE and HN and mutually authenticate them. The necessary prerequisites for starting this phase are all of the subscriber's credentials acquired by using the SUPI from the identification phase as a lookup tag. This thesis will not go in-depth on the details of this phase without much loss of generality.

Protection Schemes

The generation of the SUCI using different protection schemes allows the identification phase of the 5G-AKA protocol to be decoupled from the authentication phase. This distinction enables cryptographic flexibility by accomodating different cryptographic profiles to be used to encrypt the SUPI. Currently, only three profiles are in use:

- NULL Scheme, no protection. Used only for emergency calls when there is no established connection.
- ECIES Profile A
- ECIES Profile B

3.1.4 PID Protocol 5G Adaption

This subsection describes the necessary adaptions to the standard PID protocol necessary for mobile networking, specifically for 5G. A 5G PID protocol should be a sufficient slot-in for a protection scheme in 5G-AKA.

3GPP have identified the following essential requirements related to user privacy in TS 33.102 clause 5.1.1 [rGPP20]:

- **User Identity Confidentiality**: The property that the permanent user identity (IMSI) of a user to whom a service is delivered cannot eavesdrop on the radio access link
- **User Location Confidentiality**: The property that the presence or arrival of a user in a certain area cannot be determined by eavesdropping on the radio access link.
- **User Untraceability**: The property that an intruder cannot deduce whether different services are delivered to the same user by eavesdropping on the radio access link.

The Standard PID protocol and the informal security requirements are stated in Chapter 2. The first two requirements overlap with the existing PID Protocol requirements on subscriber confidentiality and location tracking. User untraceability is not considered in this problem, as it is inherent with 5G-AKA by provisioning pseudonyms periodically.

This thesis suggests adding the following requirements to a suitable 5G PID Protocol:

- **SUCI Unlinkability**. Two or more SUCI messages can not be linked together, as this could be used to deduce the presence of a subscriber and allow for location tracking.
- **SUCI Integrity Protection**. The SUCI scheme output should be integrity protected, preferably by a MAC also to allow authentication.
- **SUCI Replay Protection**. It should not be possible to replay a previously transmitted SUCI and go through the identification phase. This could be used to deduce the presence of a subscriber.
- **Proper SUCI Format**. The PID protocol should produce the scheme output contained in the SUCI, not the SUCI itself. See Figure 3.3.
- **Resistant to de-synchronization**. If the PID protocol relies on synchronization, it should minimize de-synchronization and provide a re-synchronization mechanism.

Some additional security requirements that could be considered depends on whether or not the 5G PID Protocol is contained within a new AKA protocol meant as a replacement for 5G-AKA:

The underlying AKA protocol should satisfy the following requirements to enable a secure 5G PID Protocol.

- **Mutual Authentication**. It should mutually authenticate the HN and Subscriber.
- **MiTM Protection**. It should not be possible to impersonate the serving network.
- **Decoupled Identification**. It should be possible to apply different PID protocols for the identification phase by allowing different protection schemes to produce the SUCI scheme output.

3.2 5G ECIES

This section describes the standard protection scheme used in 5G-AKA to produce the SUCI scheme output. 5G-AKA uses the Elliptic Curve Integrated Encryption Scheme (ECIES) for this purpose. ECIES utilizes both asymmetric and symmetric cryptography. Asymmetric cryptography is used to establish a shared secret. Symmetric cryptography is

Table 3.1
5G ECIES Profiles
Profile **A** and Profile **B**

ECIES Profile Parameters	**Cryptographic Arguments**
Elliptic Curve	**A**: Curve25519 **B**: secp256r1 (NIST P-256)
KDF	ANSI-X9.63-KDF
Hash	SHA-256
MAC	HMAC-SHA-256
Symmetric Cipher	AES-128 in CTR Mode

then used with that shared secret as a key to establishing communications. 5G ECIES is described in *Annex C* of TS.33.501 [rGPP18].

5G-ECIES uses two different profiles referred to as profile A and profile B, both technically qualifying as two distinct protection schemes. The specifications for these profiles can be seen in Table 3.1. The only significant difference between them is their choice and use of elliptic curves.

3.2.1 Asymmetric Cryptography Components

ECC

Elliptic-Curve Cryptography (ECC) generates public and private key pairs for use in public-key cryptocraphy. ECC keys have a substantially smaller memory footprint than RSA, with the equivalent of a 256-bit public key offering the same security as a 128-bit symmetric key, which would be a 3072-bit public key in RSA. [Bar20]. ECC relies on the intractability of the *Elliptic Curve Discrete Logarithm Problem* (ECDLP) [Mil86, Kob87]. It is computationally infeasible to derive the discrete logarithm of a random element on the curve based on a publicly known base point. The security is based on the ability to perform point multiplications on the curve but the inability to derive the multiplicand given the original point and resulting point. This practically makes it a one-way function. The curves used are standardized to maximize efficiency and security.

ECDH

Elliptic-curve Diffie-Hellman (ECDH) is the key agreement protocol used in ECIES that allows two parties, say, Alice and Bob, each with their own ECC key pair, to establish a shared secret over an insecure channel [BCR+18]. The public keys represent two different base points on a curve, say A and B. While the private keys, when converted to integers, represent scalar values a and b. Alice and Bob then exchange public keys and keeps their private keys secret. Alice then uses Bob's public key B and performs the scalar multiplication $a \times B$. Bob does the same with Alice's public key and computes $b \times A$. Both parties will arrive at the same point C. This new point C is called the ECDH key of the two parties.

3.2.2 Symmetric Cryptography Components

Key Derivation

The ECDH key is not directly used as a shared secret key for symmetric communication. Instead, the ECDH key is fed through a key derivation function (KDF) to produce the final shared secret key. This is to even out the entropy of the ECDH key.

Symmetric Encryption, Authentication and Integrity

A block cipher like AES then uses the shared secret key to encrypt any message to be transmitted between the two parties. The message's authentication and integrity can also be achieved by using the KDF with the ECDH key, producing an additional key used to create a Message Authentication Code (MAC).

ECDH Key-reuse

ECDH allows one party to reuse their generated keys, as long as the other party will always use new ephemeral ECC key pairs. This is done in 5G-AKA. The home network generates a static ECC key pair and keeps its private key secret within the core network. The Home Network Public Key is put into the USIM card of the subscriber during registration. It is the UE's responsibility to always use ephemeral key pairs when initiating identification.

3.2.3 ECIES at UE

This subsection describes how the UE utilizes ECIES to encrypt the MSIN. This process is visualized in Figure 3.5.

1. Ephemeral Key Pair Generation

The UE uses ECC key generation to generate the Ephemeral UE public key and Ephemeral UE private key. The public key is part of the scheme output. This is either using Curve25519 or NIST P-256.

2. Key Agreement

The ephemeral private key and the static HN public key are fed into the *Key Agreement* function. This converts the ephemeral key into a scalar. A scalar multiplication with the public key over one of the two curves in either Profile A or B produces the *Ephemeral Shared Key*.

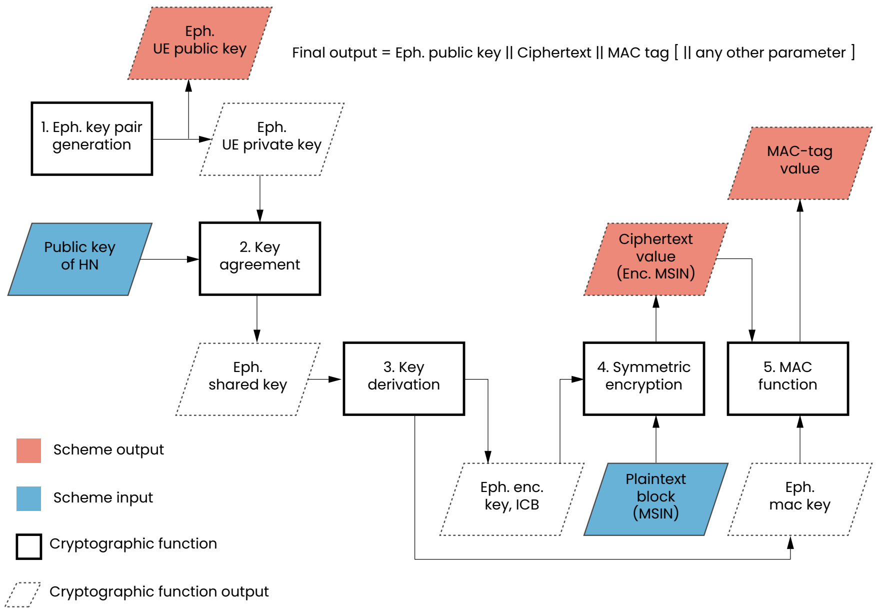


Figure 3.5: ECIES encryption process for UE. Figure adapted from Annex C in TS 55.301 [rGPP18].

3. Key Derivation

The Ephemeral shared key is fed into the chosen key derivation function. ANSI-X9.63-KDF using SHA256 as base hashing function derives an ephemeral encryption key and an ephemeral MAC key.

4. Symmetric Encryption

The ephemeral encryption key is fed into the chosen symmetric encryption function. AES-128 in Counter mode is used with the ephemeral encryption key to encrypt the MSIN of the subscriber.

5. MAC Function

The encrypted MSIN, along with the ephemeral MAC key, is fed into the MAC function. The MSIN is integrity protected by producing a MAC tag with the encrypted MSIN as input using HMAC-SHA256. The ephemeral MAC key is used as the key.

Final Output

The final output of the scheme is visualized in Figure 3.6. It is the concatenation of the Ephemeral public key, the encrypted MSIN, and the MAC tag ensuring MSIN integrity.

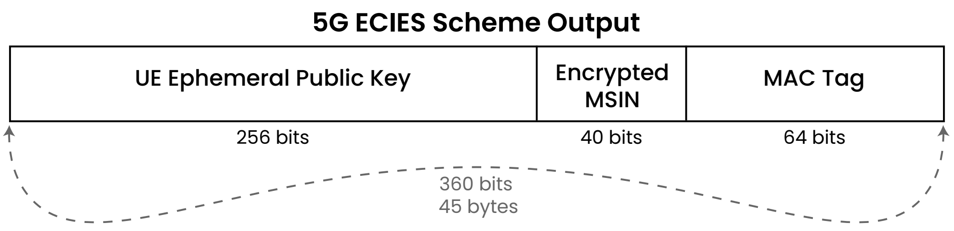


Figure 3.6: ECIES Scheme output for UE. Figure adapted from Annex C in TS 33.501 [rGPP18]

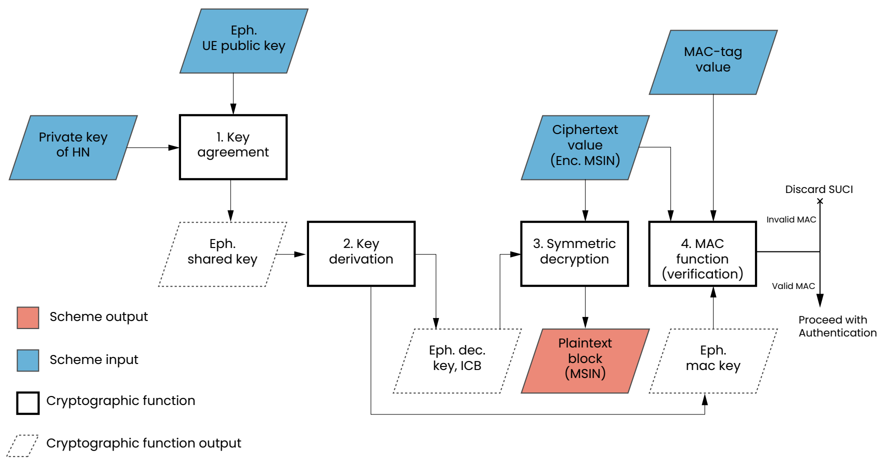


Figure 3.7: ECIES decryption process for HN

This is put into the *Scheme Output* field of the SUCI, seen in Figure 3.3. The protection scheme bits are set to 0x1 for Profile A and 0x2 for profile B of ECIES. The SUCI is then filled with the other parameters corresponding to the credentials stored on the USIM and sent to the nearest serving network. The SN will then redirect the SUCI to the home network.

3.2.4 ECIES at HN

This section describes the decryption process for ECIES done by the HN to retrieve the MSIN from the SUCI. The cryptographic primitives used by the UE at each step will be the same for the HN and are decided by the protection scheme field of the SUCI. This process is visualized in Figure 3.7.

1. Key Agreement

The HN skips key generation as it always keeps a static ECC key pair. The received SUCI contains the UE ephemeral public key and is fed into the key agreement function along with the private key of the HN. This produces an Ephemeral shared key identical to the key produced by the UE.

2. Key Derivation

The Ephemeral shared key is fed into the chosen key derivation function. It will use the same KDF as the UE to derive identical encryption and integrity keys.

3. Symmetric Decryption

The encrypted MSIN is decrypted by the symmetric decryption function, which will also be AES-128 in counter mode.

4. MAC Function

The MAC tag sent by the UE is fed into the MAC function's verification function to ensure the integrity and authenticity of the MSIN.

Completion

If the MAC tag is invalid, the request is aborted. Now that the HN has a valid MSIN, it has successfully identified the subscriber. The HN can now proceed with the authentication phase of 5G-AKA. Once this is accomplished, the UE is assigned a 5G-GUTI as a pseudonym to the SUPI. This will prevent any subsequent SUCI identification phases with the same serving network.

3.3 5G-AKA ECIES PID Protocol compatibility

This section describes the viability of ECIES and, by extension, the 5G-AKA identification phase and its compatibility with what would be considered a sufficient PID Protocol.

3.3.1 ECIES Security

5G ECIES guarantees sufficient 128 bits of classical security to conceal the SUPI. The provider is also authenticated in the process, and the subscriber, by extension, is authenticated in 5G-AKA.

Post-Quantum threat

ECC is unfortunately susceptible to quantum computers, and by extension, ECIES. ECDLP is intractable to classical computers because there is no conventional algorithm for efficiently figuring out the discrete logarithm, i.e., anything less than exponential time per bit.

This changes for quantum computers, where algorithms exist that can accomplish this task in polynomial time. [Sho94]. Global Risk institute released a quantum threat timeline report in 2020, wherein they asked 44 quantum computing experts from both industry and academia key questions to predict when quantum computers become powerful enough to pose a threat to cybersecurity. [Ins] Approximately 86% (36/44) think it's 50% likely to happen within 20 years, with 12 of them feeling that it is actually between 95%-99% in that timeframe. This suggests that the common consensus for quantum computers to become a viable threat to classic security is within 20 years.

Chosen SUCI Attacks

An adversary can capture previously sent SUCIs from other subscribers and replay this SUCI [KDM18]. The timing of the response back from the HN could be used to infer whether or not that subscriber is in the vicinity.

This is possible because the computed MAC does not account for any temporal parameters also sent in the SUCI, which allows for SUCI replay attacks.

3.3.2 SUCI Overhead

The SUCI overhaed is inferred from using Figure 3.3 and related Figure 3.6 to total the bit size of a SUCI set to use ECIES. In total, 663 bits are sent in the identification phase from the UE.

3.3.3 Performance

This issue is related to onboarding IoT devices to the 5G network, a topic expanded in the next chapter. ECIES is actually computationally slow for lightweight constrained devices. This is due to PKC just being computationally heavy in general. The supported hardware is not there to do point multiplication and scalar multiplication efficiently for many IoT devices.

Chapter 4

Onboarding IoT and Proposed Schemes

This chapter introduces the relevant IoT requirements and adapts the 5G PID protocol to cover IoT devices. The proposed schemes are then introduced, and security analysis is performed using the adapted PID protocol.

4.1 Massive and Critical IoT

This thesis will consider two main segments of the IoT application domain: Massive IoT and Critical IoT.

4.1.1 Critical IoT

Critical IoT covers all use cases that require low latency and high uptime. Typically anything that requires a constant, data-rich connection to function. Instances of use cases could be telemedicine, First Responder Technology, and factory automation. This thesis will specifically cover the latency requirements. Schulz et al. document the latency requirements for several Critical IoT use cases [SMK+17]. The latency thresholds vary greatly between each use case. But the average mobility of the use case is of specific concern to this thesis. This is because most identification phases should be followed by a change of serving networks or de-synchronization caused by mobility. The use cases with high mobility typically demand a latency threshold of around 10-100 milliseconds, but some even as low as 0.5 microseconds to 10 milliseconds.

4.1.2 Massive IoT

Massive IoT covers all use cases that require high scalability of devices, low cost, and low power consumption. Typical use cases include smart cities, Industrial IoT, and smart homes. It is essentially driven by scale and cost rather than speed. The communication overhead should therefore be minimized, and the performance overhead of any scheme should be small to maximize battery life.

4.2 Revised Threat Model

The *Dolev-Yao* threat model is frequently applied by the schemes looked at in this thesis. This allows an active adversary to eavesdrop, forge, replay, delay and mess with the order of the exchanged messages. A passive adversary can only eavesdrop. This is a natural choice accounting for both passive and active adversaries over an unsecured communication channel [Cer01]. This is a sound threat model for amobile network that only accounts for commodity mobile devices.

However, for IoT devices, the threat of side-channel attacks is a significant concern that this threat model does not account for. And depending on how the scheme works, compromising the subscriber credentials of just one subscriber in the network could compromise the privacy of a significant number of other subscribers. This is exemplified in the HashXOR section. It must be assumed it is trivial for an adversary to be physically close to the device. It has been shown that secrets have been extracted from USIM with these means, albeit this is due to them being unprotected, which is a straightforward weakness [ZYSQ13] [LYS$^+$15]. A more concerning result is shown by Udvarhelyi Et al. [UvWBS21]. Their results show that side-channel attacks could be performed on 32-bit embedded devices without sufficient knowledge of the underlying hardware architecture.

Another issue is that it is much easier for an active adversary with physical access to the device to cause a de-synchronization attack. The adversary could force the device to crash. If a rogue base station facilitates the registration procedure again with the UE, the SUCI will always be generated. As IoT devices are usually not equipped with the best tools for RNG, it could be likely that some numbers are re-used, compromising the security of any schemes hiding long-term secrets with RNG.

4.2.1 PID Protocol 5G IoT Adaption

This subsection describes the necessary adaptions to the 5G PID Protocol necessary to onboard IoT devices for mobile networking. It inherits the PID requirements specified in the subsection3.1.4.

- **Minimal SUCI**. The length of the transmitted SUCI should be minimized by reducing the PID Protocol output size.
- **No shared subscriber credentials**. Due to the increased probability of compromised subscriber credentials in an IoT device due to side-channel attacks, a PID protocol should not have shared credentials between two or more subscribers that may impact the privacy of other subscribers if leaked.
- **Lightweight Cryptography**. This is specifically important for Critical IoT applications to minimize latency but could be ceded for most Massive IoT use cases.

This is because a pseudonym will circumvent the identification phase in most cases, and the usage of the PID protocol is only necessary for initial connection and re-synchronization.

– **High Scalability**. The device density for massive IoT is unprecedented, and the scalability of the PID protocol becomes a crucial factor to handle when SUCI's are processed by the home network.

The rest of chapter will now detail the proposed lightweight schemes, and an analysis using the 5G PID Protocol derived in this section.

4.3 HashXOR

HashXOR is a lightweight private identification scheme proposed as an alternative to ECIES in 5G-AKA by Choudhury [Cho21]. This is achieved by the UE only doing two random number generations, three hashing operations, and three XOR operations. Most of the computational work is left to be done by the home network through key search.

4.3.1 Scheme Description

HashXOR requires the USIM to be registered with two new subscriber credentials, the Cluster Identity (CI) at 128-bit and Imsi Position (IP) at 24-bit. This is because the IMSIs on the home network are grouped into clusters (root nodes), organized into binary trees for efficient searching for a leaf node. These leaf nodes contain linked lists of IMSIs. Hence, the CI is used to select the appropriate root node, and the IP is used to select the appropriate leaf node containing a linked list where one of the IMSIs will be the subscriber's IMSI.

The identification phase starts with the UE generating two random 128-bit numbers, *R1* and *R2*. Two distinct hashing functions, *H1* and *H2*, are used. The CI is used as a shared secret key, as it is used to conceal the R1 number under transmission for all the subscribers in that cluster. R1 is hashed, and the result is used to conceal the IP. The IMSI is concealed by XOR — denoted by $\bigoplus$ — with a random number, acting as a one-time pad. An adversary can't attain the IMSI itself of a subscriber through either passive or active attacks.

The UE sends the following 6 parameters as the SUCI for identification to the SN, which will redirect the message in its entirety to the HN, stepwise doing this to recover the IMSI:

1. Home Network Identifier (HNI), used by SN to redirect the SUCI to the HN.

2. $C1 = H2(R1) \bigoplus IP$. Conceal IP with hashed R1 value.
3. $B1 = H1(R1)$. Conceals R1 and is used for key search in cluster list.
4. $A1 = R1 \bigoplus CI$. Conceals CI. The HN recovers both R1 and CI by iterating through the HN's list of CIs and checking if the equality $B1 == H1(A1 \bigoplus CI)$ is true. The recovered R1 is used to recover IP as well. the CI and IP are used to navigate to the leaf node containing subscriber IMSI. ($\bigoplus = XOR$)
5. $B2 = H1(R2)$. Conceals R2 and is used for key-search in the linked list contained in the leaf node.
6. $A2 = R2 \bigoplus IMSI$. Conceals IMSI. The HN recovers IMSI by iterating through the linked list and checking if $B2 == H1(A2 \bigoplus IMSI)$ is true.

256 clusters are enough to account for all the possible IMSIs in the network. And 16777216 leaf nodes in total are contained in them. Assuming that the IMSI database is fully congested and all IMSIs are uniformly distributed across each leaf node, there will be 256 IMSIs contained in each leaf node. An average number for a typical home network must be assumed to be significantly lower. As such, these two subscriber credentials would, in most scenarios, uniquely identify a subscriber in a location if they got compromised.

4.3.2 SUCI Linkability Attack

This thesis proposes the following attack on SUCI unlinkability requirement.

This attack assumes that the adversary has one or more CIs for a network. It is assumed that the CI could be obtained by side-channel attacks on any device in the network. Since the SUCI is computed on the UE, it will actually be trivial for an adversary with full access to a device to recover its CI. Moving the SUCI computation to the USIM is one mitigation to this. Still, the USIM has to accommodate two distinct hashing functions and the memory to store the results.

Normally a side-channel attack would obviously only compromise the security of that device, but this is not the case for this scheme. One out of every 256th device participating in the network could have its privacy compromised if an adversary obtains a single CI.

The attack is performed in the following manner:

- The adversary has known CI in the HN and is either eavesdropping on the network or actively disrupting it by acting as a fake base station, provoking subscribers to send their SUCIs.

- the subscriber with the same CI starts identification procedure with the SN, sending the scheme's SUCI.
- The adversary captures the SUCI and checks if it possesses the CI with the following equality check: $B1 == H1(CI \bigoplus A1)$.
- The adversary also obtains IP with $ip = C1 \bigoplus H2(R1)$.
- The adversary possesses both IP and CI for a subscriber, which is highly likely to identify the device uniquely.

SUCI unlinkability is compromised from this attack because any two different SUCIs sent by the same device can be linked.

4.3.3 Chosen SUCI Attack

This is possible because the transmitted SUCI is not integrity protected. This allows any adversary to re-use the parameters in the SUCI with a new SQN and, depending on the network response and timing, insinuate whether the subscriber is located on the network or not.

4.3.4 DDoS Attack

A Distributed Denial-of-Service (DDoS) attack is possible because the scheme is reliant on 5G-AKA authentication and replay protection. As such, the UE is authenticated in the 10th step of the AKA process. If an adversary replays a SUCI, the HN still has to search for the correct IMSI in the binary tree and linked list. As stated in the scheme paper, the average execution time to map SUPI from SUCI on the HN is $T_{HN} = 514t_{xor} + 512t_{hash} + 280t_{tr}$ Where hash are the two hashing functions, and t_{tr} represents node traversal time. This attack can be scaled up to be launched by several devices at once against the HN, which could impede mobile services.

4.3.5 PID Protocol Compatibility

This subsection highlights the incompatibility of the scheme per the desired traits of a Private Identification Protocol for 5G.

The sent SUCI is only partially in the proper format, as the IMSI is encrypted instead of the MSIN.

Architectural Changes

The scheme requires changes in both the USIM and the HN by accomodating two new subscriber credentials, CI and IP. The HN also has to re-structure the IMSI database to binary search trees and linked lists. The proposal paper includes some details on the implementation where SHA256 is used. However, the scheme requires two distinct hashing functions. This requires an additional cryptographic hashing function to be introduced as well.

4.3.6 Scalability

HN searching for IMSI is improved from the usual linear key search to logarithmic key search using binary trees. However, the HN still has to do some computational work on average, as highlighted in the preceding section.

4.3.7 SUCI Overhead

For this computation, it is assumed that the hashing operations output 256 bits, as their work uses SHA-256 as an example and does not make any mention of truncating the hash output.

The assembled SUCI consists of the concealed CI (128 bits), hashed R1 (256 bits), hashed R2 (256 bits), concealed IMSI (64 bits in their work), and concealed IP (24 bits). A 24-bit HNI is also needed to redirect the SUCI. In total, the SUCI consists of 752 bits.

4.4 PPSE-AKA

The Privacy Preservation and Security Efficient (PPSE-AKA) protocol is proposed by Parne et al. as an alternative for 5G-AKA [PGGM20]. They claim that it preserves the user's identity and protects the shared secret key. This is accomplished using only symmetric primitives. It relies on a lookup tag to compute the SUPI from the SUCI. The SUCI is integrity protected by MAC and is also ensured non-repudiation and non-reusability.

4.4.1 Scheme Description

This thesis will describe parts of the first phase of this protocol, which ensures identification for a UE initially connecting to a network or when synchronization is lost.

The USIM is provisioned with a randomly generated parameter $Simcode_{rand}$, timestamp/sequence number T_{UE}, and cryptographic key K_i. The HN also stores all these parameters for each subscriber. The random parameter is used as a lookup tag to the cryptographic key K_i, which is used to derive a delegation key DK_i used to de-conceal the SUPI.

In very brief details, the UE and HN perform the following steps for identification:

- UE derives the delegation key DK_i using T_{UE} and K_I with HMAC-SHA256.

- UE computes a one-time activation code $Actcode_{rand} = LCS_n(T_{UE} \bigoplus Simcode_{rand})$. Where LCS_n is left circular shift by n, and n is the value of the first eight-digit of the SUCI converted to decimal.
- UE computes SUCI with a reversible symmetric function, say, AES in counter mode. Using ephemeral key DK_i. The SUPI is thus encrypted, and T_{UE} is used as a nonce such that SUCI is unique. MAC is also computed.
- SUCI, $Actcode_{rand}$, T_{UE}, HNI, and MAC parameters are sent to SN, which checks HNI and redirects the message to the correct HN.
- HN checks if T_{UE} is within a threshold value for replay attack protection and discards the message if that is the case.
- HN recovers $Simcode_{rand}$ by using right circular shift,
- HN uses $Simcode_{rand}$ to lookup cryptographic key K_i associated with that subscriber. HN then derives the ephemeral key DK_i using T_{UE} sent by the UE.
- The SUPI is de-concealed using the reversible symmetric function, T_{UE} and DK_i by decrypting the SUCI.

Phase 2 deals with ensuring authentication for n connections with the same SN and is not relevant to the identification.

4.4.2 SUCI Linkability Rttack

This thesis proposes an attack on the SUCI unlinkability requirement.

It is trivial for even a passive adversary to associate any SUCI with any other SUCI sent from the same device when the first phase is executed. This is because the $Simcode_{rand}$ parameter uniquely identifies a subscriber, and it is easy for an adversary to acquire it simply by collecting SUCI, T_{UE}, and $Actcode_{rand}$ from the identification phase of a subscriber. The adversary simply computes $Simcode_{rand} = RCS_n(T_{UE} \bigoplus Actcode_{rand})$.

This issue is exacerbated by an active attacker that can cause this identification phase at will with a fake base station that drowns out the SN. This would be categorized as a SUCI catcher and would collect SUCIs and check if they can be linked with the derived $Simcode_{rand}$ parameter. If any two SUCis share $Simcode_{rand}$, then a unique subscriber would be located in that physical area. This allows for simple location tracking.

4.4.3 PID Protocol Compatability

This protocol is intended to replace 5G-AKA and does not properly differentiate between the identification phase and authentication phase. There are no multiple protection schemes available to generate the SUCI like in the original 5G-AKA.

The protocol requires the USIM to be equipped with an additional new parameter $Simcode_{rand}$.

In this publication, the SUCI is regarded as an encrypted SUPI. This doesn't seem right for several reasons:

- The SUCI format is specified by TS 33.501. The encrypted MSIN and any additional security-related parameters should be the scheme output seen in Figure 3.3.
- Only the MSIN should part of the SUPI should be encrypted. The MCC and MNC parameters are sent as the HNI of SUCI to identify the correct home network.
- In subsequent authentications with the same serving network, the SUCI is treated as a pseudonym.

4.4.4 SUCI Overhead

The publication contains details on how large in bits the first message is, which could be regarded as the actual SUCI of the protocol. However, this message lacks HNI ($\geq 24 bits$) to route the SUCI to the HN. In total, the SUCI is then 576 + 24 = 600 bits.

4.5 Braeken's Scheme

In Braeken's paper, they propose an alternative to the 5G-AKA protocol as a whole, not just the identification phase [Bra20].

Security is maintained through synchronization by sequence number N between the UE and HN. This synchronized state is provisioned with the USIM, where the USIM will contain two new identity-related parameters compared to the original 5G-AKA protocol, which will, in conjunction, act as a replacement for the temporary identifier 5G-GUTI. Synchronization is maintained if the UE's N value does not pass a certain threshold value set by the operator.

4.5.1 MiTM Attack

A *Man-in-The-Middle Attack* is possible. As pointed out by other papers, this scheme oversimplifies the communication with the HN by completely removing the SN as a middleman [Cho21] [PGGM20]. This allows an active adversary to perform a MiTM attack on subscribers using a fake base station.

4.5.2 PID Protocol Compatibility

The scheme does not take into account serving networks or visited serving networks. This scheme acts as a replacement for the 5G-AKA protocol itself. Still, it offers no means of cryptographic flexibility like the current 5G-AKA protocol, which is done through specifying protection schemes to construct SUCIs. This scheme is also not backward-compatible with older generations of mobile networks, relying on TMSI not used in this scheme.

4.5.3 SUCI Overhead

It is impossible to derive the specific SUCI overhead for this scheme, as no specific mentions of bit size are given for the temporary identifiers and certain cryptographic functions.

4.6 SEL-AKA

Gharsallah et al. propose the SEL-AKA protocol as an alternative to the current 5G-AKA protocol [GSZ19]. It uses only symmetric primitives to achieve this, and there is no shared state between the UE and HN, so synchronization errors are not an issue.

This thesis will not delve into the implementation details of this scheme due to it being both imagined as a 5g-AKA protocol replacement and having a fatal security flaw associated with identification.

4.6.1 SUCI Linkability Attack

A significant flaw with this protocol is that the SUCI is sent with a *Ref* parameter used to de-conceal the SUPI. However, as Choudhury and Parne et al. noticed, this ref parameter is unique to each subscriber and is sent in clear [Cho21] [PGGM20]. This allows a passive adversary to easily link all SUCIs produced by this protocol to one device. Hence SUCI unlinkability is not achieved.

4.6.2 PID Protocol Compatability

It is designed as an alternative to 5G-AKA and not as a slot-in for a protection scheme. The identification phase and authentication phase are not distinctly separated like in the current 5G-AKA, which allows for additional protection schemes to produce the SUCI.

The SUCI is also in the wrong format by assuming that the SUCI is the same as encrypted SUPI.

4.6.3 SUCI Overhead

The SUCI in this protocol is the first message sent by the UE. It consists of the ref (64 bits), encrypted SUPI (60 bits), nonce (32 bits), and MAC (64 bits), and HNI (24 bits). In total, this amounts to 248 bits.

4.7 Khan et al. Scheme

The symmetric scheme Post Quantum IDentification (PQID) is proposed by Khan et al. as a post-quantum protection scheme alternative to ECIES used in 5G-AKA. This scheme uses only the cryptographic algorithms that are already available on the USIM and in the HN [KDM18].

4.7.1 Scheme Description

A brief overview of how the scheme works is provided in this thesis.

This stateful symmetric scheme relies on synchronization between the UE and HN with a shared sequence number. Two new temporary identification parameters, A and B, are embedded into the USIM of the UE. The HN creates these parameters with a single master key. A is used to construct a Confidentiality key CKID. B is used to check the integrity of the SUCI.

The scheme consists of an identification phase and an update phase. The identification phase produces a SUCI, which contains the concealed SUPI and a MAC to ensure integrity and replay protection of the scheme output. The HN uses the master key to de-conceal the SUPI. After identification, a shared secret key is established between UE and HN, using the subscriber key and a random number sent to the UE. This allows the update phase to begin. In this phase, the HN produces the new A and B parameters for the UE by using the master key. They are encrypted with the new shared secret key and sent to the UE. This allows the UE to be ready for a new identification phase. 5G-AKA proceeds as usual after this, and the UE is assigned a 5G-GUTI.

4.7.2 Scheme Flaws

Choudhury points out the following flaws of the scheme [Cho21].

- A single master key K_{HN} is used for concealing the SUPI of every registered subscriber, but there is no mechanism to update this master key.
- A nonce N is used twice with K_{HN} in a XOR and hash operation to produce parameters A and B : $A = K_{HN} \bigoplus N$ and $B = h(K_{HN}, N)$. These parameters, along with the nonce, are transmitted in the clear with every SUCI transmission by any UE. An adversary could exploit this to compromise the master key K_{HN}.
- Two sequence numbers $SQN_{ID_{UE}}$ and $SQN_{ID_{HN}}$ are introduced to the HN and UE to guard against replay attack. However, this opens up for inherent issues like loss of synchronization during mobile resets and system crashes.

- IF the UE and HN's sequence numbers vary too much, de-synchronization happens. There is no mechanism in the scheme to re-synchronize.

4.7.3 PID Protocol Compatability

This scheme requires two additional temporary parameters to be instantiated on the USIM and HN. IT also requires the 5G-AKA identification response to be sent with an additional parameter.

4.7.4 SUCI Overhead

The SUCI is in the proper format and amounts to 303 bits without the scheme output. The scheme output consists of a MAC, encrypted MSIN, a concealed confidentiality key, and a concealed Sequence number. These parameters are respectively 64, 40, 128, and 48 bits. In total, this creates a SUCI message with 583 bits.

4.8 NewHope

A section on the post-quantum key exchange protocol *NewHope* is included to highlight the issues of public key post-quantum cryptography. It is not designed or supposed to be used as a standalone private identification scheme. However, as a cryptographic primitive, it could be extended to achieve that purpose.

NewHope is a post-quantum key exchange protocol proposed by Alkim et al. in late 2016 [ADPS15]. It is based on a mathematical problem known as *Ring Learning With Errors*, assumed to be hard to solve even for quantum computers. It was selected as a round-two contestant in the NIST Post-Quantum Cryptography Standardization Program but did not make the third round [CSD17b].

The protocol only offers unauthenticated key exchange, as the authors believe the key exchange protocol should be decoupled from authentication for better cryptographic flexibility. To achieve authentication, the authors assume that proven pre-quantum signatures could be used in the foreseeable future. This could then be used to authenticate a NewHope public key.

Unfortunately, keys can not be reused or cached. Therefore, both parties need to generate and use ephemeral key pairs for each exchange. The AKA process of Newhope has different parties doing distinct operations, the server and the client. The server is responsible for generating a public and private key. The public key is then sent to a client, using it as a seed to create a shared secret and client public key, which it sends back to the server. The server then uses its private key and the client public key to derive the shared secret. The following steps explain how provider authentication and shared secret derivation could happen in 5G-AKA using NewHope:

- The UE sends a SUCI with no scheme output and the protection scheme tag set to use NewHope.
- The HN generates a NewHope key pair and signs the NewHope public key using the Home Network Public Key. The NewHope public key is sent back, along with the digital signature.
- The UE authenticates the NewHope Public key as originated from its home network and uses it as input to derive a shared secret and its own public key. The UE does not need to store a private key, as the NewHope client function to derive a shared secret does so ephemerally.
- The UE uses the shared secret and a symmetric function to encrypt the MSIN.
- The SUCI scheme output is then finally made by concatenating the encrypted MSIN and UE NewHope public key. the SUCI is then sent again to the HN.
- The HN derives the shared secret using the UE's public key and decrypts the MSIN.

4.8.1 SUCI Overhead

A SUCI message, excluding the scheme output, contains 303 bits. The NewHope client public key consists of 2048 bytes, and the encrypted SUPI is 40 bits. In total, the final SUCI message sent contains 16727 bits.

However, this assumes the usage of a classic digital signature. In reality, it would only make sense to use NewHope with a post-quantum digital signature. This requires modifying the Home Network public key to accommodate that. Hence, the SUCI size would be significantly larger.

This protocol also requires two communication rounds. First, get the HN to generate its ephemeral key pair, then for the UE to use those keys. This adds significant latency.

Chapter 5

Experimental Setup

This chapter covers the selected hardware used in the experiment and the selected software to create the experimental framework and do the experimentation.

5.1 Hardware Selection

32-bit embedded devices with varying computational capability and memory are the main target for this thesis. The devices reflect what would constitute mid to high end of performance for IoT at a cheap entry point. Jimenez Et al. evaluated the performance of the ECIES scheme on a handful of Android devices that reflect the average device on the market [JNNN17]. This study seeks a similar baseline for the average 32-bit constrained embedded device while also looking at performance for proposed symmetric solutions.

Evaluation of the schemes following the proposals use 8-bit embedded devices with clock speed as low as 16Mhz, for instance, the Arduino Uno device [Cho21]. This device is a reasonable estimate for use in low-end IoT. This experiment's relevant cryptographic libraries implement standard cryptographic primitives that optimize for 8-bit platforms and include measured execution times in their documentation [WW]. Those results are used as an alternative for testing against 8-bit devices in this experiment.

Table 5.1 details the IoT devices used for evaluation and their specifications.

All the devices support the Arduino Platform, simplifying the development time of the experimental framework with broad library support and extensive documentation. All devices use their distinct development framework, and as such, device flexibility and choice were kept in mind when designing the experimental framework in this study.

The laptop was used for serial communication with the embedded devices, both for uploading code to the devices and acquiring experimental results. Any device suitable for development work could be substituted for this purpose.

Table 5.1
Device Specifications

Specifications	**ESP32**	**ESP8266**	**Nano 33 IoT**
Development Board	Adafruit HUZZAH ESP32 Feather	Adafruit HUZZAH ESP8266 Feather	Arduino Nano 33 IoT
Micro Processor	Xtensa LX6 Dual-core	Xtensa L106 Single-core	SAMD21 Cortex-M0+
Clock Speed	240 MHz	80 MHz	48 MHz
Flash Memory	4 MiB	4 MiB	256 KiB
SRAM	520 KiB	64 KiB	32 KiB

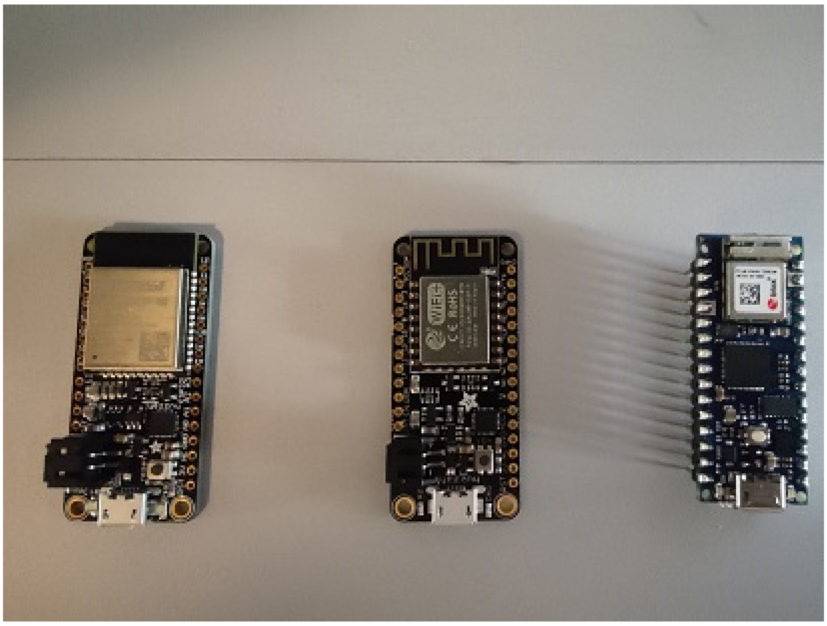


Figure 5.1: The development boards used in the experiment, from left to right: Huzzah ESP32, Huzzah ESP8266, and Arduino Nano 33 IoT.

5.2 Software

This section describes everything used to develop the experimental framework, carry out experimentation, and process the results.

PlatformIO in Visual Studio Code

PlatformIO is an embedded development platform with an extensive embedded platform, framework, board, and library support [Pla]. It has a multi-platform and multi-architecture build system that enables easy development and management over several boards that may differ in architecture, framework, and platform choice.

PlatformIO IDE is an extension in VScode was used for this project. It integrates nicely to provide a rich IDE for flexible embedded development.

C++

C++ code is written, compiled, and uploaded to the devices. The code includes both the setup of the experimental framework and all evaluated cryptographic functions.

Python

The Python programming language (v.3.9) [Pyt] allowed scripts to be made for serial communications with the embedded devices and treatment of the results from that serial communication.

Arduino platform

Using this platform as a foundation for library and code support allows the experimental framework to be used on as many devices as possible [WW]. It has extensive library support, particularly for cryptographic libraries that implement standard cryptographic primitives. It was also the only common platform for the hardware.

YAML

The experimental framework uses YAML headers to serialize meta information in each experimental session between the host computer and device.

5.3 Experimental Framework

The experimental framework was designed with these goals in mind:

- Simplify and streamline evaluation and testing of cryptographic functions on embedded devices.

– Create interfaces that allow for simple testing and evaluation of execution time of cryptographic functions.

– Allow for easy board management, with extensive configuration options, library, debugging, and tool support.

– Simplify serial communication and data manipulation.

– Allow for both simple and advanced logging of results.

Two design cycles were necessary for creating the experimental framework:

– The first design cycle had all necessary code set up for the pipeline of testing cryptographic functions, serialization-deserialization of results and meta-information from the device to the host computer, and data manipulation to prepare the results.

– The second design cycle streamlines this process by abstracting the code into independent classes that each performs one of the aforementioned tasks. The code was structured sequentially in a Jupyter notebook to provide step-by-step instructions to quickly get started with benchmarking and data manipulation of the results. All functions were also documented.

The code for the experimental framework is on github and contains 2 branches [Gab21].

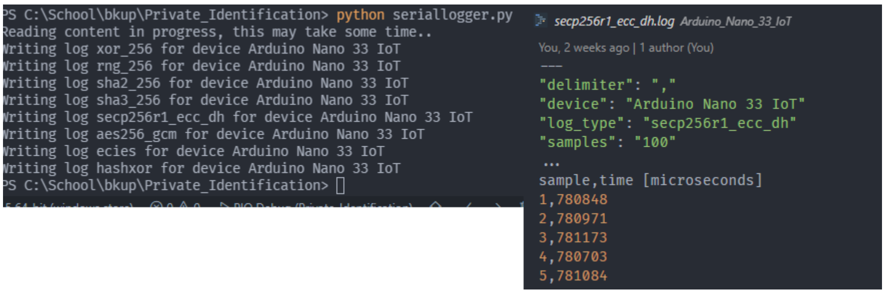
– **Master**. Contains code used in this thesis for experimentation, which uses the first design cycle of the framework. It also includes step-by-step instructions for how the experimental results were derived.

– **Skeleton.** Contains the barebones experimental framework synthesized from the second design cycle, along with a few examples of cryptographic primitives being evaluated.

It has the following functionality:

– Able to quickly change between evaluating different boards on the fly seamlessly.

– Setting up unit tests of any code, which could, for instance, be test vectors for cryptographic primitives to ensure that the cryptographic functions produce the desired output.



```
PS C:\School\bkup\Private_Identification> python seriallogger.py
Reading content in progress, this may take some time..
Writing log xor_256 for device Arduino Nano 33 IoT
Writing log rng_256 for device Arduino Nano 33 IoT
Writing log sha2_256 for device Arduino Nano 33 IoT
Writing log sha3_256 for device Arduino Nano 33 IoT
Writing log secp256r1_ecc_dh for device Arduino Nano 33 IoT
Writing log aes256_gcm for device Arduino Nano 33 IoT
Writing log ecies for device Arduino Nano 33 IoT
Writing log hashxor for device Arduino Nano 33 IoT
PS C:\School\bkup\Private_Identification> 
```

secp256r1_ecc_dh.log Arduino_Nano_33_IoT

You, 2 weeks ago | 1 author (You)

```
---
"delimiter": ","
"device": "Arduino Nano 33 IoT"
"log_type": "secp256r1_ecc_dh"
"samples": "100"
...
sample,time [microseconds]
1,780848
2,780971
3,781173
4,780703
5,781084
```

Figure 5.2: A session of benchmarking and resulting output

- Benchmarking of multiple cryptographic functions at once. This could also be configured to only output the average execution time of the benchmarked functions.
- Ability to extend the logging functionality to include additional meta-information or new results through additional columns.
- Serialization of both results and meta-information about the experiment in log files.
- De-serialization on the host computer using Python and subsequent data manipulation in a step-by-step process in a Jupyter Notebook.
- Easy data presentation and comparisons of the evaluated cryptographic functions, as all cryptographic functions logged under one device are dynamically put into a comparison table.
- Ability to estimate a scheme's execution time based on benchmarked cryptographic primitives used in the scheme.

Figure 5.2 shows to the left the host PC capturing the serialized benchmark results for several cryptographic primitives against one device. This is done dynamically after setting some variables. To the right, one of those log files can be seen. It also includes meta-information contained in a YAML header. The meta-information is used to specify the CSV log file format and include some additional information necessary when working with the data.

Appendix A contains a walkthrough of how to set up the experimental framework and use it.

5.4 Experimentation

This section describes how the experimental framework was set up to test the schemes and primitives for this thesis.

Two additional Arduino libraries were necessary for the project:

- **Arduino Crypto Library** [WW]. It contains a Random Number Generator (RNG), all the symmetric primitives, and the NewHope code.
- **micro-ecc** [Mac21]. Contains fast ECDH implementation with NIST P-256 as one of the supported curves.

5.4.1 Experimental Methodology

This is roughly the step-by-step process that was followed to gather experimental results on a cryptographic primitive or scheme:

1. Implement the cryptographic function will be benchmarked. If it is a cryptographic primitive, this is simply done by importing the implementations by aforementioned libraries.
2. Create and use unit test for the cryptographic function. Usually consisting of input and correct output, to ensure correctness of the function.
3. Create wrapper function(s) necessary for timing one iteration of the cryptographic function. These wrapper functions do necessary setup and teardown for each benchmark sample.
4. Use the benchmarking function and wrapper functions to time the cryptographic function with 100 samples.
5. Serialize meta-information (number of samples, device and primitive being benchmarked) along with benchmark results of all 100 samples.

6. Deserialize and parse meta information with python script and store the results as a log file.

The log files are written sequentially with a sentinel value for starting and stopping, so several cryptographic functions can be benchmarked at one run of the python script.

5.4.2 Cryptographic Scheme Implementations

The experiment's goal was to acquire insight into the execution time of ECIES encryption and the proposed symmetric solutions encryption on selected 32-bit embedded devices. The experimental framework was developed to enable this. Due to time constraints and technical problems, only the ECIES, Newhope, and HashXOR schemes could be implemented fully on the devices. However, most of the standard cryptographic primitives used by the other proposed schemes were tested for validity, and their execution time was evaluated. Those results can be used to create estimates for the execution time of the schemes.

ECIES Implementation

This experiment's implementation profile of ECIES differs from the two profiles used in 5G-AKA but guarantees the same or higher security at a similar execution time.

secp256pr1, also known as NIST P-256, is the elliptic curve used, similar to 5G ECIES profile B. AES is also used, albeit AES-256 in Galois Counter Mode simplifies integrity protection. This also removes the need for a distinct H MAC computation. SHA256 is used for key derivation, not ANSI X9.63 Key Derivation Function (KDF) with SHA256.

HashXOR Implementation

HashXOR was implemented using SHA256 and SHA3-256 cryptographic hash functions. This differs slightly from the proposal's own experimental results, which only used SHA256. This is because the scheme creators state that two distinct hashing functions should be used in the encryption, but how this is done is not explicitly stated in their own experimental results, as it lacks implementation details.

Chapter 6

Experimental Results

This chapter details all the experimental results derived from using the experimental framework. How these results were derived can be seen in a step-by-step process in Appendix B.

6.1 Primitives

The tested primitives' average execution time can be seen in Table 6.1. Keys are also set so they can be referred to in computing estimates for other schemes.

Table 6.1
Average Primitive Execution Time
Seconds

Primitive		**Device**		
Key	Name	ESP32	ESP8266	Nano IoT
t_{XOR}	XOR	3.00×10^{-6}	7.00×10^{-6}	1.10×10^{-5}
t_{RNG}	RNG	3.10×10^{-5}	9.20×10^{-5}	1.70×10^{-4}
$t_{BLAKE2s}$	BLAKE2s	1.90×10^{-5}	8.30×10^{-5}	1.44×10^{-4}
t_{SHA2}	SHA256	2.80×10^{-5}	1.07×10^{-4}	2.61×10^{-4}
t_{SHA3}	SHA3-256	1.90×10^{-4}	5.50×10^{-4}	1.34×10^{-3}
$t_{sym.enc}$	AES256 Enc.	1.50×10^{-5}	1.79×10^{-4}	3.24×10^{-4}
$t_{sym.dec}$	AES256 Dec.	9.00×10^{-6}	2.44×10^{-4}	5.25×10^{-4}
t_{GCM}	AES-256-GCM	1.53×10^{-4}	1.03×10^{-3}	2.57×10^{-3}
t_{ECDH}	NIST P-256 ECC-DH	0.17	0.978*	0.781

The ESP8266 device has one abnormal result, NIST P-256 ECC+DH (both key generation on the IoT device and shared secret derivation). It seems to have a higher execution time than the supposedly weaker Nano IoT device. This can be explained by the fact that the ESP8266 has an inbuilt hardware watchdog timer. This timer will periodically interrupt the main code such that system maintenance code can be run, which takes roughly 500 milliseconds. This will halt the ECC+DH function because it simply takes too long. This is an issue in the implemented ECIES scheme as well.

6.2 Implemented Schemes

The schemes implemented on the device for testing were limited to HashXOR using SHA3 + SHA256, ECIES, and NewHope. The NewHope implementation is contained in the Arduino Crypto Library and is based on the cited GitHub repository [SN18].

Table 6.2
Average Implemented Scheme Execution Time
Seconds

	Device		
Scheme	ESP32	ESP8266	Nano IoT
HashXOR SHA3	3.11×10^{-4}	8.56×10^{-4}	2.18×10^{-3}
NewHope	1.12×10^{-2}	NaN	7.73×10^{-2}
ECIES	1.70×10^{-1}	9.79×10^{-1}	7.84×10^{-1}

The average execution time is listed in Table 6.2. Execution time for NewHope on ESP8266 could not be derived because of the significant stack size required, causing issues when attempting to run the code. Arduino Crypto Library minimized the necessary memory requirements, but it still requires 10.2KB of memory to run NewHope on a client device [WW].

6.3 Estimated Schemes

This section describes the process of estimating a scheme execution time based on its cryptographic primitives. In this case, the schemes are estimated using table 6.1. PPSE is not estimated due to its inherent handling of SUCI as a pseudonym and concealed initial identifier. This makes it difficult to assess which phases should be included and omitted partly due to time constraints with the thesis.

Table 6.3 is based on similar work done by Choudhury et al., although adapted to the cryptographic primitives used in this thesis [Cho21]. The keys in this table are used to look up the average execution time in Table 6.1.

Table 6.3
Scheme Cost Using Primitives

Scheme	**Primitive computations**
Khan	$t_{RNG} + 3 \cdot t_{SHA2} + t_{sym.dec}$
Braeken	$7 \cdot t_{SHA2}$
SEL-AKA	$T_{RNG} + 3 \cdot t_{SHA2} + t_{sym.enc}$
HashXOR BLAKE2S	$2 \cdot t_{RNG} + 3 \cdot t_{XOR} + 2 \cdot t_{BLAKE2s}$
HashXOR SHA3	$2 \cdot t_{RNG} + 3 \cdot t_{XOR} + 2 \cdot t_{SHA3}$
ECIES	$T_{ECDH} + T_{SHA2} + T_{GCM}$

Table 6.4
Average Estimated Scheme Execution Time
Seconds

	Device		
Scheme	ESP32	ESP8266	Nano IoT
Khan	1.61×10^{-4}	7.85×10^{-4}	1.77×10^{-3}
Braeken	1.96×10^{-4}	7.49×10^{-4}	1.83×10^{-3}
SEL-AKA	1.30×10^{-4}	5.92×10^{-4}	1.28×10^{-3}
HashXOR BlAKE2s	1.18×10^{-4}	3.95×10^{-4}	7.78×10^{-4}
HashXOR SHA3	2.89×10^{-4}	8.62×10^{-4}	1.98×10^{-3}
ECIES	1.70×10^{-1}	9.79×10^{-1}	7.84×10^{-1}

HashXOR blake2S is included to see the effect of using a more performant hashing function compared to SHA3. ECIES and HashXOR using SHA3 are also estimated theoretically to see the difference between the theoretical and actual implementations.

Finally, we use Table 6.3 and Table 6.1 to derive Table 6.4

Chapter 7 Discussion

This chapter summarises the informal scheme analysis using the final 5G PID Protocol specifications synthesized in Chapter 4. This information is organized into Table 7.1. Observations about the performance of the schemes are also made using the experimental results from Chapter 6. Finally, these observations are used to answer the research questions specified in Chapter 1.

7.1 Observations

This section will detail observations made from the informal scheme analysis and then detail performance evaluations using the experimental results.

7.1.1 Scheme Analysis

See Table 7.1 for an overview of how the different schemes vary regarding 5G PID Protocol requirements. This table was derived from Chapter 4 and the proposed scheme subsections. Further explanation on how this table is derived is also explained throughout this section.

Symmetric Scheme Weaknesses

PPSE-AKA and SEL-AKA were vulnerable to passive eavesdropping due to their fixed reference parameters. By default, this problem is exacerbated by an active adversary that can, for instance, jam devices and force devices to perform the identification phase. Hence, they do not possess *User Location Confidentiality*.

In a similar vein, *HashXOR* has a *Linkability attack* that limits the unlinkability of the SUCI. In most instances, the SUCI's will easily be linked. However, the prerequisites of this attack might be a strong assumption, wherein the adversary gets access to subscriber credentials.

Table 7.1
Analysis of Proposed Schemes Overview

	Asymmetric		Symmetric				
Proposals:	**ECIES (5G)**	**NewHope**	**HashXOR**	**PPSE-AKA**	**Braeken**	**SEL-AKA**	**Khan**
Year of publication:	2018	2016	2020	2021	2020	2019	2018
Security							
User Identity Confidentiality	Yes	No	Yes	Yes	Yes	Yes	Yes
User Location Confidentiality	Yes	No	No	No	Yes	No	Yes
User Untraceability	Yes	No	Yes	Yes	Yes	Yes	Yes
SUCI Integrity Protection	Yes	No	No	Yes	Yes	Yes	Yes
SUCI Replay Protection	No	No	No	Yes	Yes	Yes	Yes
SUCI Unlinkability	Yes	Yes	No	No	Yes	No	Yes
Mutual Authentication	Yes	No	Yes	Yes	No	Yes	Yes
MiTM Protection	Yes	No	Yes	Yes	No	Yes	Yes
Attacks/weaknesses:							
Linkability Attack	No	No	Yes	Yes	No	Yes	No
De-synchronization	No	No	No	No	No	No	Yes
Chosen SUCI Attack	Yes	Yes	Yes	No	No	No	No
Other (comment)		Unauth. Auth.	DDOS Attack				Nonce Reuse
Vulnerable to							
Passive Adversary	No	No	No	Yes	No	Yes	No
Active Adversary	No	Yes	Yes	Yes	Yes	Yes	No
PID Protocol Compatibility							
Decoupled Identification	Yes	No	Yes	No	No	No	Yes
SUCI Protection Scheme	Yes	No	Yes	No	No	No	Yes
5G-AKA Replacement	No	No	No	Yes	Yes	Yes	No
Proper SUCI Format	Yes	No	No	No	No	No	Yes
Scalability	Constant	Constant	Log.	Constant	Constant	Constant	Constant
SUCI Size (bits)	663	16727	752	600	Unknown	248	583

*Vulnerable to attacks

The *Braeken* scheme did not model the serving network into their 5G-AKA protocol and is inherently vulnerable to *MiTM attacks.*

The *Khan Et al.* scheme has no re-synchronization method and is inherently vulnerable to forced de-synchronization attacks. This could limit the reliability of the scheme.

SUCI Format Compatibility

Only one of the proposed schemes had a proper format for the SUCI. There seems to be an insufficient understanding of the protection scheme paradigm that 5G-AKA uses. Hence, the schemes did not properly account for how it fits with 5G-AKA by assuming that the SUCI is just the encrypted SUPI, thus accounting for no additional signaling information.

Decoupled Identification

Of all the proposed alternatives to the 5G-AKA protocol as a whole, none of them properly accounted for having a distinct identification phase that could be easily replaced. This is done in 5G-AKA using the protection scheme. This issue is also reflected in the SUCI format compatibility. These protocols, therefore, lack the cryptographic flexibility that 5G-AKA has with the protection schemes.

7.1.2 Performance

The performance results are based on the average execution time of both implemented and estimated schemes in Table 6.2 and Table 6.4. The scalability and SUCI size in Table 7 is also considered.

5G ECIES Execution Time

The average execution time for 5G ECIES on these devices may be *sufficiently low* for some massive IoT use-cases where identification will only *rarely* occur. The ESP32 device took "only" 0.17 seconds to create the SUCI necessary for identification. The Nano IoT device took 0.78 seconds. Unfortunately, the results from the ESP8266 were unreliable for ECIES, but should in general, be somewhere between the ESP32 device and Nano IoT device.

For critical IoT applications, ECIES would incur too high latency to be used. To satisfy some of the use cases in this category, it would require ECIES to have an average execution time of fewer than 5 milliseconds.

NewHope

At first glance, the execution time of NewHope is favorable, with the ESP32 device creating a shared secret in only 11.2 milliseconds and the Nano IoT device in 77.3 milliseconds.

But this does not account for the added latency of two communication rounds, which is only when the scheme is unauthenticated. For this to be made authenticated, it would require additional cryptographic operations where the NewHope key is signed using a post-quantum signature scheme. This would significantly increase the average execution time.

Furthermore, the memory requirements of NewHope are extremely significant. With just NewHope, the SUCI size is 16727 bits. The scheme also requires significant memory on the device, as 10.2KB of memory was required to use NewHope properly. And this does not even account for adding authentication with post-quantum signatures.

Symmetric Scheme Performance

All the symmetric schemes had favorable execution times and were quite similar. The only significant difference comes from which cryptographic hashing operation was used. The impact of using SHA3 on the ESP8266 and Nano IoT were remarkably high.

All the schemes except HashXOR also had favorable scalability. This is due to HashXOR having a very high average amount of cryptographic primitives used in every key search on the HN, which exacerbates DDoS attacks.

In terms of SUCI size, all symmetric schemes and ECIES were also similar. SEL-AKA does stand out with a smaller SUCI size, but the consequence is insufficient security.

7.2 Answers to Research Questions

This section will answer the research questions highlighted in Section 1.2.1.

RQ1. To what degree is the current 5G Identity privacy scheme suitable for IoT devices?

Based on observations made about ECIES, it seems suitable for use in Massive IoT applications that *rarely* need additional identification phases using SUCI. These identification phases could happen because of mobility and de-synchronization with the serving network. In normal cases, the SUCI identification phase should happen once, and a pseudonym with no performance impact is used in subsequent identification.

A 32-bit embedded device with the equivalent power of the ESP32 device or more could be used as a gateway device, where more constrained devices communicate efficiently with this device to relay information to the 5G Network.

RQ2. To what degree do the lightweight alternatives substitute the current solution in regards to security and performance?

Based on the observations made about the symmetric schemes, the performance is favorable for both Massive IoT and critical IoT in terms of execution time and scalability.

However, the security is still insufficient, where all of the schemes are vulnerable to an active adversary.

RQ3. What are the possible trade-offs and differences between these lightweight solutions?

The symmetric schemes all accomplish their performance and security goals in different ways. However, they all end up having some form of inherent weakness.

Both PPSE-AKA and SEL-AKA do a security-performance tradeoff by using reference parameters that could be used to identify the subscriber.

HashXOR seems to trade security for more efficient performance by introducing a shared cluster identity parameter as a new subscriber credential, but this opens up to a new attack that compromises the SUCI unlinkability of the scheme.

Braeken and Khan et al.'s scheme could be said to trade security and reliability for more efficient performance due to their synchronized parameters.

Only Khan et al. scheme accounted for the existing 5G architecture by minimizing the introduction of new cryptographic primitives and having the proper SUCI format.

In general, it seems that all of the 5G-AKA schemes only prioritized performance instead of user privacy and reliability.

RQ4. Do the proposed solutions minimally interfere with current 5G security specifications and architecture?

None of the proposed 5G-AKA alternatives decoupled the identification phase from the authentication phase. Hence, there are no mechanisms for cryptographic flexibility where a different protection scheme could be used to construct the SUCI.

Similarly, none of the schemes barring Khan et al. had a proper SUCI format, which would make it an incompatible protection scheme for 5G-AKA. SUCI was, for instance, made to be equivalent to an encrypted SUPI.

All solutions also introduced additional subscriber credentials to be stored in the USIM and HN.

In general, it seems that the main issue stems from a lack of understanding of what a protection scheme in 5G-AKA should accomplish.

Chapter 8 Conclusion

This chapter concludes the thesis by summarizing the work done and what research was accomplished. Finally, some future work is recommended.

8.1 Summary of Thesis

5G identity privacy is now more crucial than ever, as this mobile network is set to support IoT devices. However, the identity privacy scheme used in 5G is new and relies on public-key cryptography. This made it uncertain how reliable identification would be on IoT devices. Hence, several lightweight symmetric alternatives were proposed.

This thesis takes root in the Private Identification problem for 5G mobile networking and specifies requirements for a 5G PID Protocol. The current 5G solution and proposed alternatives are then evaluated according to these requirements. An experimental framework was made to simplify the evaluation and was promptly used in this thesis.

This framework went through two design cycles, and the result is a sequential and well-documented process for benchmarking cryptographic primitives and schemes. The experimentation used the framework to benchmark actual scheme implementations and estimate scheme performance from evaluated cryptographic primitives.

The results from all the analyzed schemes and the associated experimental results were then used to create observations used to answer the research questions. The answers indicate that the current 5G solution for identity privacy, ECIES, is manageable for massive IoT use cases where concealed identification will rarely happen. However, the symmetric schemes either had insufficient identity pricacy or insufficient reliability. Furthermore, due to an insufficient understanding of identification in 5G-AKA, they suffer from a lack of cryptographic flexibility.

8.2 Future Work

The experimental framework created in this thesis could be used to evaluate the proposed primitives in the NIST Lightweight weight cryptography contest on commodity IoT devices [CSD17a]. It could also be used in the NIST Post-quantum contest [CSD17c].

The evaluations of the proposed schemes could be used to avoid pitifalls for designing a new Private Identification Protocol suitable for 5G.

The specifications for a 5G PID Protocol could be adapted and extended to cover other use cases.

References

[ADPS15] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - a new hope, 2015.

[Bar20] Elaine Barker. Recommendation for key management:: Part 1 - general. Technical Report NIST SP 800-57pt1r5, National Institute of Standards and Technology, Gaithersburg, MD, May 2020.

[BCR$^+$18] Elaine Barker, Lily Chen, Allen Roginsky, Apostol Vassilev, and Richard Davis. Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography. Technical Report NIST Special Publication (SP) 800-56A Rev. 3, National Institute of Standards and Technology, April 2018.

[BGG$^+$20] F. Boudot, P. Gaudry, A. Guillevic, N. Heninger, E. Thomé, and P. Zimmermann. Comparing the difficulty of factorization and discrete logarithm: A 240-digit experiment, 2020.

[Bra20] An Braeken. Symmetric key based 5G AKA authentication protocol satisfying anonymity and unlinkability. *Computer Networks*, 181:107424, 2020.

[Cer01] Iliano Cervesato. The Dolev-Yao Intruder is the Most Powerful Attacker. In *Proceedings of the Sixteenth Annual Symposium on Logic in Computer Science | LICS'01*, pages 16–19. IEEE Computer Society Press. Short, 2001.

[Cho21] Hiten Choudhury. HashXor: A lightweight scheme for identity privacy of IoT devices in 5G mobile network. *Computer Networks*, 186:107753, 2021.

[CSD17a] Information Technology Laboratory Computer Security Division. Lightweight Cryptography | CSRC | CSRC. https://csrc.nist.gov/projects/lightweight-cryptography, January 2017.

[CSD17b] Information Technology Laboratory Computer Security Division. Round 3 Submissions - Post-Quantum Cryptography | CSRC | CSRC. https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions, January 2017.

[CSD17c] Information Technology Laboratory Computer Security Division. Round 3 Submissions - Post-Quantum Cryptography | CSRC | CSRC. https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions, January 2017.

[Gab21] Kenneth Gabrielsen. Kennethgab/Experimental-Framework. https://github.com/Kennethgab/Experimental-Framework, July 2021.

[GSZ19] Ikram Gharsallah, Salima Smaoui, and Faouzi Zarai. A Secure Efficient and Lightweight authentication protocol for 5G cellular networks: SEL-AKA. In *2019 15th International Wireless Communications Mobile Computing Conference (IWCMC)*, pages 1311–1316, 2019.

[Ins] Global Risk Institute. Quantum Threat Timeline Report 2020. https://globalriskinstitute.org/publications/quantum-threat-timeline-report-2020/.

[JNNN17] Enrique Cobo Jimenez, Prajwol Kumar Nakarmi, Mats Naslund, and Karl Norrman. Subscription identifier privacy in 5G systems. In *2017 International Conference on Selected Topics in Mobile and Wireless Networking (MoWNeT)*, pages 1–8. IEEE, 2017.

[KDM18] Haibat Khan, Benjamin Dowling, and Keith M Martin. Identity Confidentiality in 5G Mobile Telephony Systems. *Security Standardisation Research*, page 23, 2018.

[Kob87] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[LYS$^+$15] Junrong Liu, Yu Yu, François-Xavier Standaert, Zheng Guo, Dawu Gu, Wei Sun, Yijie Ge, and Xinjun Xie. Small Tweaks Do Not Help: Differential Power Analysis of MILENAGE Implementations in 3G/4G USIM Cards. In Günther Pernul, Peter Y A Ryan, and Edgar Weippl, editors, *Computer Security – ESORICS 2015*, Lecture Notes in Computer Science, pages 468–480. Springer International Publishing, 2015.

[Mac21] Ken MacKay. Kmackay/micro-ecc, June 2021.

[Mil86] Victor S. Miller. Use of Elliptic Curves in Cryptography. In Hugh C. Williams, editor, *Advances in Cryptology — CRYPTO '85 Proceedings*, Lecture Notes in Computer Science, pages 417–426, Berlin, Heidelberg, 1986. Springer.

[MO17a] Stig Frode Mjølsnes and Ruxandra-Florentina Olimid. Easy 4G/LTE IMSI Catchers for Non-Programmers. *235-246*, 2017.

[MO17b] Stig Frode Mjølsnes and Ruxandra Florentina Olimid. The Challenge of Private Identification. *iNetSec*, page 16, 2017.

[MO19] Stig F. Mjolsnes and Ruxandra F. Olimid. Private Identification of Subscribers in Mobile Networks: Status and Challenges. *IEEE Communications Magazine*, 57(9):138–144, 2019.

[OF18] Edward J. Oughton and Zoraida Frias. The cost, coverage and rollout implications of 5G infrastructure in Britain. *Telecommunications Policy*, 42(8):636–652, September 2018.

[PGGM20] Balu L. Parne, Shubham Gupta, Kaneesha Gandhi, and Shubhangi Meena. PPSE: Privacy Preservation and Security Efficient AKA Protocol for 5G Communication Networks. In *2020 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, pages 1–6. IEEE, 2020.

[Pla] PlatformIO. PlatformIO is a professional collaborative platform for embedded development. https://platformio.org.

[Pol75] J. M. Pollard. A monte carlo method for factorization. *BIT Numerical Mathematics*, 15(3):331–334, September 1975.

[Pyt] Python. Welcome to Python.org. https://www.python.org/.

[rGPP18] 3rd Generation Partnership Project. Security architecture and procedures for 5g systems (ts 33.501 version 15.0.0 release 15). *3GPP*, 2018.

[rGPP20] 3rd Generation Partnership Project. 3g security security architecture (ts 33.102 version 16.0.0 release 16). *3GPP*, 2020.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[Sho94] P.W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

[SMBE17] Bardia Safaei, Amir Mahdi Monazzah, Milad Bafroei, and Alireza Ejlali. Reliability Side-Effects in Internet of Things Application Layer Protocols, December 2017.

[SMK$^+$17] Philipp Schulz, Maximilian Matthe, Henrik Klessig, Meryem Simsek, Gerhard Fettweis, Junaid Ansari, Shehzad Ali Ashraf, Bjoern Almeroth, Jens Voigt, Ines Riedel, Andre Puschmann, Andreas Mitschele-Thiel, Michael Muller, Thomas Elste, and Marcus Windisch. Latency Critical IoT Applications in 5G: Perspective on the Design of Radio Interface and Network Architecture. *IEEE Communications Magazine*, 55(2):70–78, February 2017.

[SN18] Peter Schwabe and Michael Naehrig. Newhopecrypto/newhope-usenix. https://github.com/newhopecrypto/newhope-usenix, December 2018.

[Sta] Statista. Number of IoT devices 2015-2025. https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/.

[UvWBS21] Balazs Udvarhelyi, Antoine van Wassenhove, Olivier Bronchain, and François-Xavier Standaert. On the Security of Off-the-Shelf Microcontrollers: Hardware Is Not Enough. In Pierre-Yvan Liardet and Nele Mentens, editors, *Smart Card Research and Advanced Applications*, volume 12609 of *Lecture Notes in Computer Science*, pages 103–118. Springer International Publishing, 2021.

[WW] Rhys Weatherley and Brandon Wiley. Arduino Cryptography Library: Arduino Cryptography Library. https://rweather.github.io/arduinolibs/crypto.html.

[ZYSQ13] Yuanyuan Zhou, Yu Yu, François-Xavier Standaert, and Jean-Jacques Quisquater. On the Need of Physical Security for Small Embedded Devices: A Case Study with COMP128-1 Implementations in SIM Cards. In Ahmad-Reza Sadeghi, editor, *Financial Cryptography and Data Security*, Lecture Notes in Computer Science, pages 230–238. Springer, 2013.

Appendix A

Setting up and using the framework

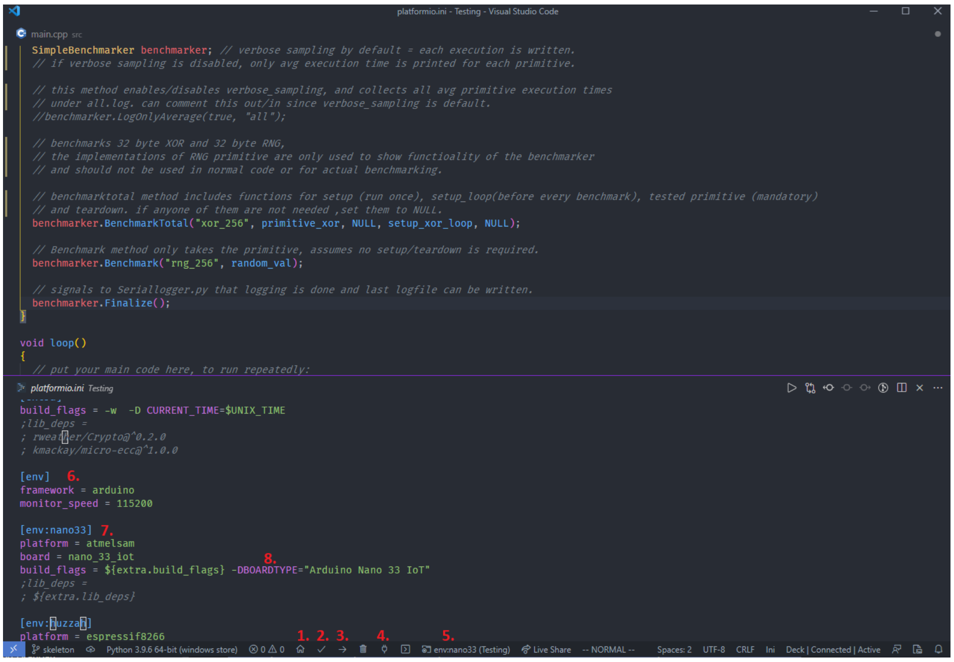
prerequisites:

- USB driver(s) for development board(s)
- Visual Studio Code
- Python v3.6+

The development board's USB driver is required for the host computer to detect it over USB. For an Arduino device, the driver can easily be installed using the Boards Manager utility in the Arduino IDE. Windows 10 may also automatically recommend installing the associated USB driver when you first connect the board.

The PlatformIO IDE is integrated into Visual Studio Code as an extension. It can be installed through the extension marketplace. Once installed and VSCode is reloaded, the IDE will be available through both the status bar and command palette. See Figure A.1 for an example view of the IDE and details of the project. it contains the following numbered elements:

1. The home button that opens up PlatformIO home overview, where you can create and open projects, add new boards, libraries, etc..
2. The build button that builds selected code for the chosen environment in 5.
3. The upload button that uploads the last build to the device.
4. A serial monitor that will display what the device prints.
5. The current environment/board chosen to develop and deploy code will be the current device connected over USB to your host computer.
6. The common environment for all devices.



```cpp
    SimpleBenchmarker benchmarker; // verbose sampling by default = each execution is written.
    // if verbose sampling is disabled, only avg execution time is printed for each primitive.

    // this method enables/disables verbose_sampling, and collects all avg primitive execution times
    // under all.log. can comment this out/in since verbose_sampling is default.
    //benchmarker.LogOnlyAverage(true, "all");

    // benchmarks 32 byte XOR and 32 byte RNG,
    // the implementations of RNG primitive are only used to show functioality of the benchmarker
    // and should not be used in normal code or for actual benchmarking.

    // benchmarktotal method includes functions for setup (run once), setup_loop(before every benchmark), tested primitive (mandatory)
    // and teardown. if anyone of them are not needed ,set them to NULL.
    benchmarker.BenchmarkTotal("xor_256", primitive_xor, NULL, setup_xor_loop, NULL);

    // Benchmark method only takes the primitive, assumes no setup/teardown is required.
    benchmarker.Benchmark("rng_256", random_val);

    // signals to Seriallogger.py that logging is done and last logfile can be written.
    benchmarker.Finalize();
}

void loop()
{
    // put your main code here, to run repeatedly:
```

```ini
build_flags = -w  -D CURRENT_TIME=$UNIX_TIME
;lib_deps =
; rweather/Crypto@^0.2.0
; kmackay/micro-ecc@^1.0.0

[env]   6.
framework = arduino
monitor_speed = 115200

[env:nano33]  7.
platform = atmelsam
board = nano_33_iot                           8.
build_flags = ${extra.build_flags} -DBOARDTYPE="Arduino Nano 33 IoT"
;lib_deps =
; ${extra.lib_deps}

[env:huzzah]
platform = espressif8266
```

Figure A.1: PlatformIO project in VScode

7. The specific environment for a device.

8. A BOARDTYPE variable being set in the environment that can be used for logging.

Click the home icon in VSCode's status bar to open PlatformIO's overview. From this overview, create a simple new project, find your development board, and select it. This will install the necessary toolchain for compiling code and create a configuration file necessary for your device. This project can remain unused.

The experimental framework will be opened as a new project in PlatformIO from the home overview in VSCode.

Get the experimental framework from Github. Here is the link to the skeleton branch, containing only the experimental framework:

https://github.com/Kennethgab/Experimental-Framework/tree/skeleton

This could be downloaded and unzipped, or use *git clone* and change the branch to *skeleton*.

The platformio.ini file contains the necessary configuration details for compiling code to your board. The previous project created for your device also has a platformio.ini file. Take the content from this file, the specific configuration details for your board, and append it to the experimental framework's platormio.ini. Your device can now be selected as an environment to build and connect towards from VScode's status bar.

TO uniquely identify your device in code, you need to define the *BOARDTYPE* variable with a unique name representing your board. Do so like it is done for the other device environments by adding a build flag with *-D BOARDTYPE="Your device"*

Unit tests can also be made, and the logic for that is put up in the tests folder. To run unit tests against a device, the command palette must be opened *(CTRL+SHIFT+P)*, and the "*PlatformIO: Run Test..*" command must be executed.

Everything you need to do on the host computer is sequentially ordered in a Jupyter notebook. This is an environment where you can execute python code in cells and get output for each cell. The data_treatment.ipynb file is a notebook that contains step-by-step instructions for going through the framework. A print of this file is added at the end of this appendix.

The python libraries used on the host computer, installed either through the notebook or through the *requirements.txt* file:

– **pandas**. used to manipulate data extracted from devices
– **pyserial**. Used to communicate with the device over USB using Python.
– **pyyaml**. Used for loading and writing YAML files, which reads the log headers for meta-information.

The *src/main.cpp* file is the entry point for writing code on the development board. this contains some example code for benchmarking primitives, notably including a class *SimpleBenchmarke*r that simplifies the process. Using this class can be seen in Figure A.1, where it will evaluate an XOR and RNG function through two different interfaces.

Several functions could be initialized to start testing a new cryptographic primitive or scheme, but only one is mandatory for the SimpleBenchmarker class to work.

– A setup function. This runs once in the benchmark to set up any necessary preliminaries like static buffers that the primitive function could use.
– A loop setup function. This function runs before every new benchmark sample and could be used to re-initialize the input for the benchmarked primitive function.

- A primitive function. This is the code that will be benchmarked and is mandatory to include.
- a teardown function. Runs at the end of the benchmarking session once.

Once a primitive is set up and ready to be benchmarked, the data_treatment notebook file can be followed step-by-step to capture the results.

Python walkthrough

The next pages of this appendix will contain the printout of the data_treatment notebook in the skeleton branch. For a detailed insight into what each function does, check the relevant python files contained in the repository. All functions in the framework are documented.

data_treatment_theoretical

June 22, 2021

[1]:
```
import sys

# make sure all relevant libraries are installed
!{sys.executable} -m pip install pandas pyserial pyyaml
```

```
Requirement already satisfied: pandas in c:\users\kenne\appdata\local\packages\p
ythonsoftwarefoundation.python.3.9_qbz5n2kfra8p0\localcache\local-
packages\python39\site-packages (1.2.4)
Requirement already satisfied: pyserial in c:\users\kenne\appdata\local\packages
\pythonsoftwarefoundation.python.3.9_qbz5n2kfra8p0\localcache\local-
packages\python39\site-packages (3.5)
Requirement already satisfied: pyyaml in c:\users\kenne\appdata\local\packages\p
ythonsoftwarefoundation.python.3.9_qbz5n2kfra8p0\localcache\local-
packages\python39\site-packages (5.4.1)
Requirement already satisfied: pytz>=2017.3 in c:\users\kenne\appdata\local\pack
ages\pythonsoftwarefoundation.python.3.9_qbz5n2kfra8p0\localcache\local-
packages\python39\site-packages (from pandas) (2021.1)
Requirement already satisfied: numpy>=1.16.5 in c:\users\kenne\appdata\local\pac
kages\pythonsoftwarefoundation.python.3.9_qbz5n2kfra8p0\localcache\local-
packages\python39\site-packages (from pandas) (1.20.3)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\kenne\appdata\
local\packages\pythonsoftwarefoundation.python.3.9_qbz5n2kfra8p0\localcache\loca
l-packages\python39\site-packages (from pandas) (2.8.1)
Requirement already satisfied: six>=1.5 in c:\users\kenne\appdata\local\packages
\pythonsoftwarefoundation.python.3.9_qbz5n2kfra8p0\localcache\local-
packages\python39\site-packages (from python-dateutil>=2.7.3->pandas) (1.16.0)
WARNING: You are using pip version 21.1.1; however, version 21.1.2 is available.
You should consider upgrading via the 'C:\Users\kenne\AppData\Local\Microsoft\Wi
ndowsApps\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\python.exe -m pip
install --upgrade pip' command.
```

[2]:
```
# import all functions used for processing data
import pandas as pd
from SerialLogger import write_logs
from Analyzelogs import load_log, categorize_logfiles, calculate_all, \
 calculate_cost, extract_avg

# display values in dataframe table setting to 2 significant digit
```

```
pd.set_option('display.float_format', '{:.2E}'.format)
```

```
[ ]: # run this cell each time you log primitive(s) to write the .log files
     # from serial communication
     write_logs("./test_results/")
```

```
[ ]: # categorize device name here, do multiple if multiple devices.
     #


     root_dir = "./test_results" # TODO set to your workspace root or use relative␣
      ↪path
     device_dir = "your_device_folder" # TODO set to the folder containing logfiles

     device_dict = categorize_logfiles(f"{root_dir}/{device_dir})
```

```
[ ]: # create index, the full name of each primitive being benchmarked
      #TODO fill in with your primitives
     index = ["XOR" , "RNG"]

     # create corresponding list to the index that contains the raw log name of␣
      ↪primitive
     #TODO fill in with your own primitive log_type names matching index, add more␣
      ↪primitives
     primitives = ["xor_256", "rng_256"]

     # for each primitive in the list, calculate the avg
     # for each primitive and store in a a list/column by the order given in␣
      ↪primitives
     # so devices_avgs[0] = xor avg, devices_avg[1] = rng avg
     device_avgs = extract_avg(device_dict, primitives)

     # create primitive execution time table, column for each device
     # TODO put in more devices if you want

     primitives_avg = pd.DataFrame({"DEVICE" : device_avgs},
     index=index
     )
     print(primitives_avg)

     # store result in .file ./tables/primitives_avg.tex
     primitives_avg.to_latex("./tables/primitives_avg.tex")
```

```
[ ]: #calculate implemented scheme avg execution time
     # just like primitive avg execution time

     implemented_schemes_index = ["ECIES"]
     implemented_schemes = ["ecies"]

     device_avg_schemes = extract_avg(device_dict, implemented_schemes)

     # assemble dataframe again, can also do this in previous cell
     # all in one go instead.

     implemented_schemes_avg = pd.DataFrame({"DEVICE" : device_avg_schemes},
     index = implemented_schemes_index)
      print(implemented_schemes_avg)

      # store result in a .tex file
      implemented_schemes_avg.to_latex('./tables/implented_schemes_avg.tex')
```

```
[ ]: # calcualte theoretical scheme avg

     # index to use in assembled dataframe
     scheme_index = ["Scheme1", "Scheme2"
      ]

     # primitives and amounts they're used for each scheme
     # stored as tuples in a list

     scheme1_cost = [["SHA256",3], ["XOR", 2]]
     scheme2_cost = [["SHA3-256", 7],  ["RNG"], 3]
     # put these in a list for iteration
     schemes = [scheme1_cost, scheme2_cost]

     # calculate cost series using AVG execution time dataframe
     # to lookup primitive execution time

     device_theoretical_schemes = calculate_all(primitives_avg, schemes, "DEVICE")

     # assemble dataframe/table by using columns as costs
     # for each device, and scheme names as index

     schemes_theoretical = pd.DataFrame({"DEVICE" : device_theoretical_schemes},
     index=scheme_index)
     print(schemes_theoretical)
      # store result
      schemes_theoretical.to_latex("./tables/theoretical_schemes_avg.tex")
```

Appendix B
Step-by-step derivation of experimental results

This appendix shows the python notebook for how the experimental results were derived. the code for this is from the master branch contained in the repo:

https://github.com/Kennethgab/Experimental-Framework

For detailed information into implementing all primitives, schemes, and functions, check the Github repository.

data_treatment_actual

June 22, 2021

```
[ ]: import sys# make sure all relevant libraries are installed
     !{sys.executable} -m pip install pandas pyserial pyyaml
```

```
[5]: # import all functions used for processing data
     import pandas as pd
     from SerialLogger import write_logs
     from Analyzelogs import load_log, categorize_logfiles, calculate_all, \
      calculate_cost, extract_avg
```

```
Reading content in progress, this may take some time..
Writing log xor_256 for device Arduino Nano 33 IoT
Writing log rng_256 for device Arduino Nano 33 IoT
Done!
```

```
[ ]: # run this cell each time you log primitive(s) to write the .log files
     # from serial communication. store in test_results folder.
     write_logs("test_results/")
```

```
[5]: # Categorizing all the expermimental results on cryptographic primitives
     #  for ESP32 into pandas dataframes.  stored in dicts.

     root_dir = "./test_results"
     esp32_dir =  "Adafruit_Feather"
     esp8266_dir = "Adafruit_Huzzah"
     nano_dir =  "Arduino_Nano_33_IoT"


     # put all logfiles into dicts by each device, each logfile a pandas dataframe
     #  with columnssample and time [microseconds]
     esp32 = categorize_logfiles(f"{root_dir}/{esp32_dir}")
     esp8266 = categorize_logfiles(f"{root_dir}/{esp8266_dir}")
     nano_iot = categorize_logfiles(f"{root_dir}/{nano_dir}")

     time_col = "time [microseconds]"
     esp32_xor = esp32["xor_256"][time_col]
```

```
print(round(esp32_xor.mean()))
```

```
3
```

[6]:
```
# Constructing the measured primitive table using the mean execution time in␣
 ↪microseconds
index = ["XOR", "RNG", "BLAKE2s", "SHA256", "SHA3-256","AES256 Enc.",
 "AES256 Dec.", "AES-256-GCM", "NIST P-256 ECC+DH"]

primitives = ['xor_256', 'rng_256','blake2s','sha2_256', 'sha3_256'
,'aes256_enc','aes256_dec','aes256_gcm', 'secp256r1_ecc_dh']

# for each primitive in the list, calculate the avg execution time and put that␣
 ↪in a list.
# returns a column of average execution time for each device
esp32_avgs = extract_avg(esp32, primitives)
esp8266_avgs = extract_avg(esp8266, primitives)
nano_avgs = extract_avg(nano_iot, primitives)

# display values in dataframe table setting to 2 significant digit
pd.set_option('display.float_format', '{:.2E}'.format)

# create primitive execution time table, columns for each device average␣
 ↪execution time.
primitives_avg = pd.DataFrame({"ESP32" : esp32_avgs,
"ESP8266" : esp8266_avgs, "Nano IoT" : nano_avgs}, index=index)
print(primitives_avg)

# store in file ./tables/primitives_avg.tex
primitives_avg.to_latex('./tables/primitives_avg.tex')
```

```
                     ESP32  ESP8266  Nano IoT
XOR               3.00E-06 7.00E-06  1.10E-05
RNG               3.10E-05 9.20E-05  1.70E-04
BLAKE2s           1.90E-05 8.30E-05  1.44E-04
SHA256            2.80E-05 1.07E-04  2.61E-04
SHA3-256          1.90E-04 5.50E-04  1.34E-03
AES256 Enc.       1.50E-05 1.79E-04  3.24E-04
AES256 Dec.       9.00E-06 2.44E-04  5.25E-04
AES-256-GCM       1.53E-04 1.03E-03  2.57E-03
NIST P-256 ECC+DH 1.70E-01 9.78E-01  7.81E-01
```

[7]:
```
#  implemented scheme average execution time

# index for dataframe/table implemented scheme average excution time
implemented_schemes_index = ["HashXOR SHA3", "NewHope", "ECIES" ]
# keys to use in dictionary to get right dataframe
```

```
implemented_schemes = ["hashxor", "newhope_client", "ecies"]

# create 3 lists containing avg execution time of lists for each device
#  in form [hashxor_avg, newhope_avg, ecies_avg]
esp32_avg_schemes = extract_avg(esp32, implemented_schemes)
esp8266_avg_schemes = extract_avg(esp8266, implemented_schemes)
nano_avg_schemes = extract_avg(nano_iot, implemented_schemes)

# assemble dataframe where primitive is index and columns are avg execution␣
 ↪time of schemes by device.
implemented_schemes_avg = pd.DataFrame({ "ESP32" : esp32_avg_schemes,
"ESP8266" : esp8266_avg_schemes, "Nano IoT" : nano_avg_schemes},
 index=implemented_schemes_index)

print(implemented_schemes_avg)

 # store processed data in file
implemented_schemes_avg.to_latex('./tables/implemented_schemes_avg.tex')
```

```
                ESP32  ESP8266  Nano IoT
HashXOR SHA3 3.11E-04 8.56E-04  2.18E-03
NewHope      1.12E-02      NaN  7.73E-02
ECIES        1.70E-01 9.79E-01  7.84E-01
```

[13]:

```
# theoretical scheme average execution time
# khan: rand, 3 xor, 4 hash, 1 sc.dc
# braeken: 7 hash
# SEL-AKA: rand, 3 hash, sc.en
# ecies: 1 hash, 1 ecc+dh, 1 aes256/gcm
# hashxor: 2 rng, 3 xor, 2 distinct hashing functions

# index to use in assembled dataframe of theoretical scheme avg execution time
scheme_index = ['Khan', "Braeken", "SEL-AKA",  "HashXOR BlAKE2s",
 "HashXOR SHA3", "ECIES"]

# primitives and amounts they're used for each scheme, used to calculate scheme
#  cost by table lookup in avg
# primitive execution time table
braeken_prims = [["SHA256", 7]]
sel_aka_prims = [["RNG", 1], ["SHA256",3], ["AES256 Enc.", 1]]
khan_prims = [["RNG", 1], ["XOR", 3], ["AES256 Dec.", 1], ["SHA256", 4]]
hashxor_2 = [["RNG", 2], ["XOR", 3], ["SHA256", 1], ["BLAKE2s", 1]]
ecies_prims = [["SHA256", 1], ["NIST P-256 ECC+DH", 1], ["AES-256-GCM", 1]]
hashxor_prims = [["RNG", 2], ["XOR",3], ["SHA256", 1], ["SHA3-256", 1]]


# scheme costs put in a list so they can be iterated through,
# same order as index so they're matching
schemes = [khan_prims, braeken_prims, sel_aka_prims,
  hashxor_2, hashxor_prims, ecies_prims ]

# use scheme costs and goes through all of them, returns
#  list of avg theoretical execution time for the schemes in the same order as␣
↪index.
esp32_schemes = calculate_all(primitives_avg, schemes, "ESP32")
esp8266_schemes = calculate_all(primitives_avg, schemes, "ESP8266")
nano_schemes = calculate_all(primitives_avg, schemes, "Nano IoT")


# assemble dataframe/table by using columns as costs
#  for each device, and scheme names as index.
schemes_theoretical = pd.DataFrame({"ESP32" : esp32_schemes,
 "ESP8266" : esp8266_schemes, "Nano IoT" : nano_schemes}, index=scheme_index)
schemes_theoretical.to_latex("./tables/theoretical_schemes_avg.tex")
print(schemes_theoretical)
```

```
                  ESP32  ESP8266  Nano IoT
Khan            1.61E-04 7.85E-04  1.77E-03
Braeken         1.96E-04 7.49E-04  1.83E-03
SEL-AKA         1.30E-04 5.92E-04  1.28E-03
HashXOR BlAKE2s 1.18E-04 3.95E-04  7.78E-04
HashXOR SHA3    2.89E-04 8.62E-04  1.98E-03
ECIES           1.70E-01 9.79E-01  7.84E-01
```

[73]:
```
# calculate percentage error between theoretical schemes and implemented schemes

def percentage_difference(a,b):
    delta = abs(a - b);
    return round((delta/(a+b))*100,2)

def percentage_error(a, b):
    delta = abs(a - b)
    return round((delta/abs(b))*100,2)

def percentage_change(a,b):
    delta = a -b
    return round((delta/abs(b))*100, 2)

# look at error differences between scheme theoretical and implemented



theoretical = schemes_theoretical.at["ECIES", "ESP32"]*1e6
implemented = implemented_schemes_avg.at["ECIES", "ESP32"]*1e6
```

```
print("theoretical: ",theoretical)
print("implemented: ",implemented)
error = percentage_error(theoretical, implemented)
difference = percentage_difference(theoretical, implemented);
change = percentage_change(implemented, theoretical)
print(f"error: {error}%\ndifference: {difference}%\nchange: {change}%")
```

```
theoretical:  170006.0
implemented:  170049.0
error: 0.03%
difference: 0.01%
change: 0.03%
```

[ ]:

```

```