In-network processing for 5G crypto using FPGA-based NICs

Ritika Nagar
2020112

BTP report submitted in partial fulfillment of the requirements for the Degree of B.Tech. in Computer Science & Engineering
on 9th May 2023

BTP Track: Research

BTP Advisor
Dr. Rinku Shah

BTP Co-advisor
Dr. Sumit Darak

Indraprastha Institute of Information Technology
New Delhi

Student's Declaration

I hereby declare that the work presented in the report entitled "**In-Network Crypto Primitives,**" submitted by me for the partial fulfillment of the requirements for the degree of Bachelor of Technology in Computer Science & Engineering at Indraprastha Institute of Information Technology, Delhi, is an authentic record of my work carried out under the guidance of **Dr. Rinku Shah**. Due acknowledgments have been given in the report for all material used. This work has not been submitted elsewhere for the reward of any other degree.

..... **Place & Date: New Delhi, 29 Nov 2023**

Ritika Nagar

Certificate

This is to certify that the above statement by the candidates is correct to the best of my knowledge.

..... **Place & Date: New Delhi, 29 Nov 2023**

Dr. Rinku Shah

Abstract

The latest advancements in the field of 5G telecommunications have proposed the requirements of high speeds (~1 Gbps per user) and low latency (< 1 ms) for a new world of possible applications such as AR/VR, autonomous driving, enhanced mobile broadband, and dense deployments of IoT devices. To achieve such high-performance requirements for the 5G network components and applications, solutions such as kernel bypass techniques and offload to programmable data plane hardware have been proposed. The management plane of the 5G network implements security algorithms to ensure confidentiality and integrity within the wireless network, between the wireless and the mobile core (wired), and between the components within the mobile core network. The 5G network implements the New Generation Encryption Algorithms (NEA) and New Generation Integrity Algorithms (NIA) to support ciphering and integrity, respectively. Some research has proposed to offload the 5G components, such as the Access and Mobility Function (AMF), New Generation Node B (gNB), and User Plane Function (UPF), to the programmable data planes (PDPs) to accelerate the performance and reduce power consumption. With the offloading of security-related network functions like AMF and gNB, it becomes crucial to offload the cryptographic algorithms implemented by these functions to avoid hypervisor/kernel stack traversal, ensure better throughput, and lower network latencies.

Our project aims to design an in-network 5G security solution that promises high speed, low processing latency, scalability, dynamic reconfigurability, and reduces power consumption by leveraging FPGA-based network hardware.

Keywords: 5G, in-network processing, cryptography algorithms, FPGA, performance acceleration

Acknowledgments

Firstly, I am expressing my sincere gratitude towards my Project Advisor, Dr. Rinku Shah, for her continued support, guidance, and patience during the project. She has provided us with the technical support required during the project and has guided my focus during uncertain times. We would also like to thank Dr. Sumit Darak, project co-advisor, for his valuable time and input.

Contents

Some Useful abbreviations and definitions :

AKA: Authentication and Key Agreement

NAS: Non-Access Stratum

AS: Access Stratum

USIM: Universal Subscribers Identity Module

UDM: UDM is primarily in charge of generating AKA authentication credentials, storing and managing all user identity information (SUPI) in the 5G system, and decrypting SUCI

SUPI: Subscribers Permanent Identifier this is generated and stored in both UDM and USIM

SUCI: Subscribers Concealed Identifier

GUTI: Globally Unique Temporary UE Identity

AMF: AMF is one of the essential network elements in the 5G core network, which is responsible for registration management, connection management, reachability management, and mobility management in 5GS, as well as NAS message ciphering and integrity protection

MME: Mobility Management Entity; Network Function in 4G core network

S-GW: Serving/Signaling Gateway

P-GW: PDN Gateway

HSS: Home Subscriber Network

Introduction

The traditional telecommunication infrastructure (e.g., 3G, 4G) was based on fixed-function networking hardware components. Any changes to the device feature required hardware vendor intervention and long turn-around times. The recent fifth generation (5G) telecommunication system follows the software-defined networking paradigm, where the 5G network components are classified into — the control plane and the data plane; where the control plane is responsible for deciding forwarding policies, monitoring, and management functions, while the data plane follows the instructions from the control plane and forwards the mobile user's data to/fro the Internet. The control plane components run as software (VMs or containers) over commodity servers and configure the data plane hardware switches to follow forwarding policies.

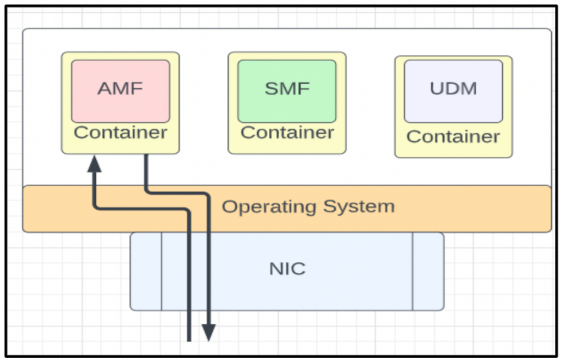
Applications such as AR/VR, autonomous driving, enhanced mobile broadband, dense deployments of IoT devices, highspeed entertainment in a moving vehicle, and delay-sensitive video applications require high throughput (~1 Gbps/user), very low processing latencies (<1 ms) with stringent quality of service (QoS).
A high-performance and low-cost UPF is necessary to meet these requirements.

Traditional Network Components are used to run on commodity servers. They consumed higher energy and precious compute cycles, increasing latencies during communication between control plane network functions and communication between control plane and data plane functions.
With the help of Programmable hardware, researchers were able to offload some of the control plane network functions to the data planes, which resulted in [1]:

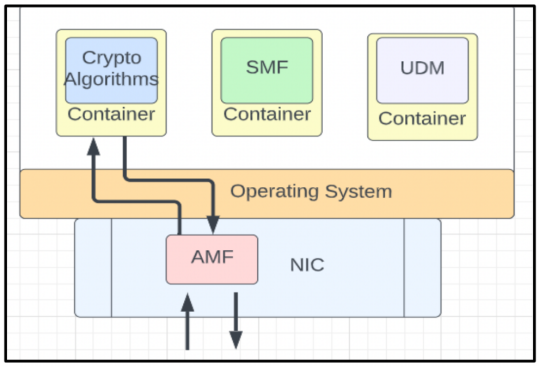
1. Improved performance by reducing the amount of computation done in the control plane.
2. Reduced Complexity of the network by separating the control and data plane .
3. Increased scalability, allowing the data plane to handle more traffic without hampering the performance of the control plane due to independence.

Traditionally, the devices in the data plane used to follow specific instructions i.e., were fixed to

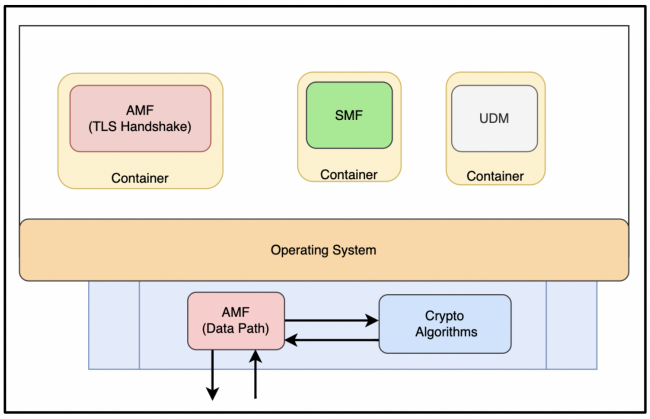
what set of actions they could perform, but with the developments in the programmable data planes(PDP), we could now program hardware devices in the data plane and hence add some functionality to these hardware devices as per the requirements, unlike the fixed function devices.



No Offload



Partial Offload



Complete Offload

With programmable hardware, researchers have offloaded some control plane functions to achieve better performance. Offloading of security-related Network functions like AMF and the gNB has been achieved by researchers. However, because the cryptographic ciphering and integrity algorithms that are used by these functions are still in the control plane, the network functions have to call them repetitively from the control plane which increases CPU cycles and results in higher latencies and inefficient use of energy.

This gives rise to the problem of partial offloading, where offloading the network security functions remains inefficient. These network functions depend on other security functions present in the control plane, reducing throughput and efficiency. Our project aims to solve the problem of partial offloading so that the remaining cryptographic functions can be offloaded to the data plane, ensuring high speeds,

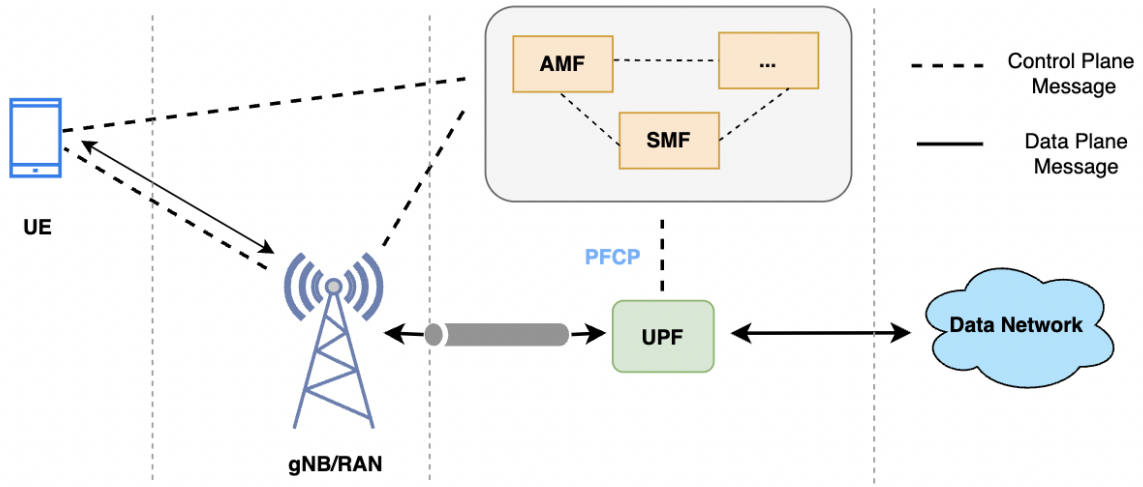
Lower latency, hardware reusability, scalability, and dynamic reconfigurability by using FPGA-based network hardware.

Background

5G network Structure

In the 5G network structure, we can broadly divide the network structure into two parts where one is the Access network, and the other is Core network.
The access network is designed to ensure connectivity with the end users. The Radio Access Network(RAN) provides connectivity service to the user and helps the end user connect to the Core Network.
The Core network contains all the 5G network functions which interact with each other and help with the 5G functionalities of the network, like authenticating the user, providing and managing security keys before connection to the network, etc.

The core network can further be divided into two parts where the control plane has various functions like the Access and Mobility Function(AMF), the Session Management Function(SMF), the Authentication Server Function(AUSF), and various other functions while in the Data Plane we have functions like the User Plane Function(UPF).



The AMF is responsible for authenticating the device before it can send any data on the network. UE first authenticates itself on the AMF before transmitting any data. With successful authentication with the AMF, UE sends data to the RAN.

The RAN is responsible for connecting the UE with the Core Network, sending the data to the

internet through the UPF. UPF is an important network function in the 5G network as it is responsible for processing data packets. The efficient working of the UPF is one of the reasons for the low latencies and higher throughput of the 5G network.

5G Network Functions and Secure Attachment Procedure

A few of the 5G network components are derived from the traditional 4G components, where the functionality of 4G components is further broken down into smaller functional components. Below is a small table to map the relationship between 5G vs 4G network components.

Tabular representation of relationships between some of the essential 5G and 4G components:

	MME	**S-GW**	**P-GW**	**HSS**
AMF	✔			
SMF	✔		✔	
UPF		✔	✔	
AUSF				✔
UDM				✔

In a 5G network, each User Equipment(UE) has a Unique Subscriber Identity Module(USIM) which stores a provisioned Subscribers Identifier known as SUPI. Each USIM stores the Subscribers Permanent Identifier(SUPI), which is never transmitted over the network in plain text to avoid any privacy breach due to the leak of SUPI over the network. The Subscriber Concealed Identifier(SUCI) is used to conceal the SUPI during transmission to prevent leaking of SUPI as per 5G security guidelines.

To initiate Authentication, during the initial registration, the UE sends SUCI and the connection request to the AMF. The AMF forwards the SUCI to the AUSF and the UDM for receiving the SUPI from SUCI.

The UDM uses SIDF to conceal the SUCI and return the SUPI to the AUSF, which then returns the SUPI information with the response message for the incoming connection request. The AMF generates a GUTI, an alternative for temporarily communicating SUPI details over the network; the AMF maps the GUTI value to a SUPI value and uses this mapping for a future incoming connection request from the UE.

On authentication completion, the NAS layer security procedure is initiated. The NAS layer protocol manages the connection between the UE and the core network.

For enabling NAS security, the AMF selects the required ciphering and integrity algorithms for the communication according to the provided security capabilities of the UE and sends a NAS

Security Mode Command to the UE. The UE completes the security process with NAS Security Complete Message. All the NAS layer communications are now ciphered and integrity protected after the NAS security is complete.

After the AS security, The network assigns an IP address with the Attach Accept message and contains UE Security Capabilities in the Attach Accept message to initiate the AS security mechanism. For initiating the AS security mechanism, the gNB generates the AS Security Mode Command message based on the selected security algorithms in the Attach Accept message and sends it to the UE. The UE acknowledges the message with an AS Security Mode Complete message and completes the attachment process.

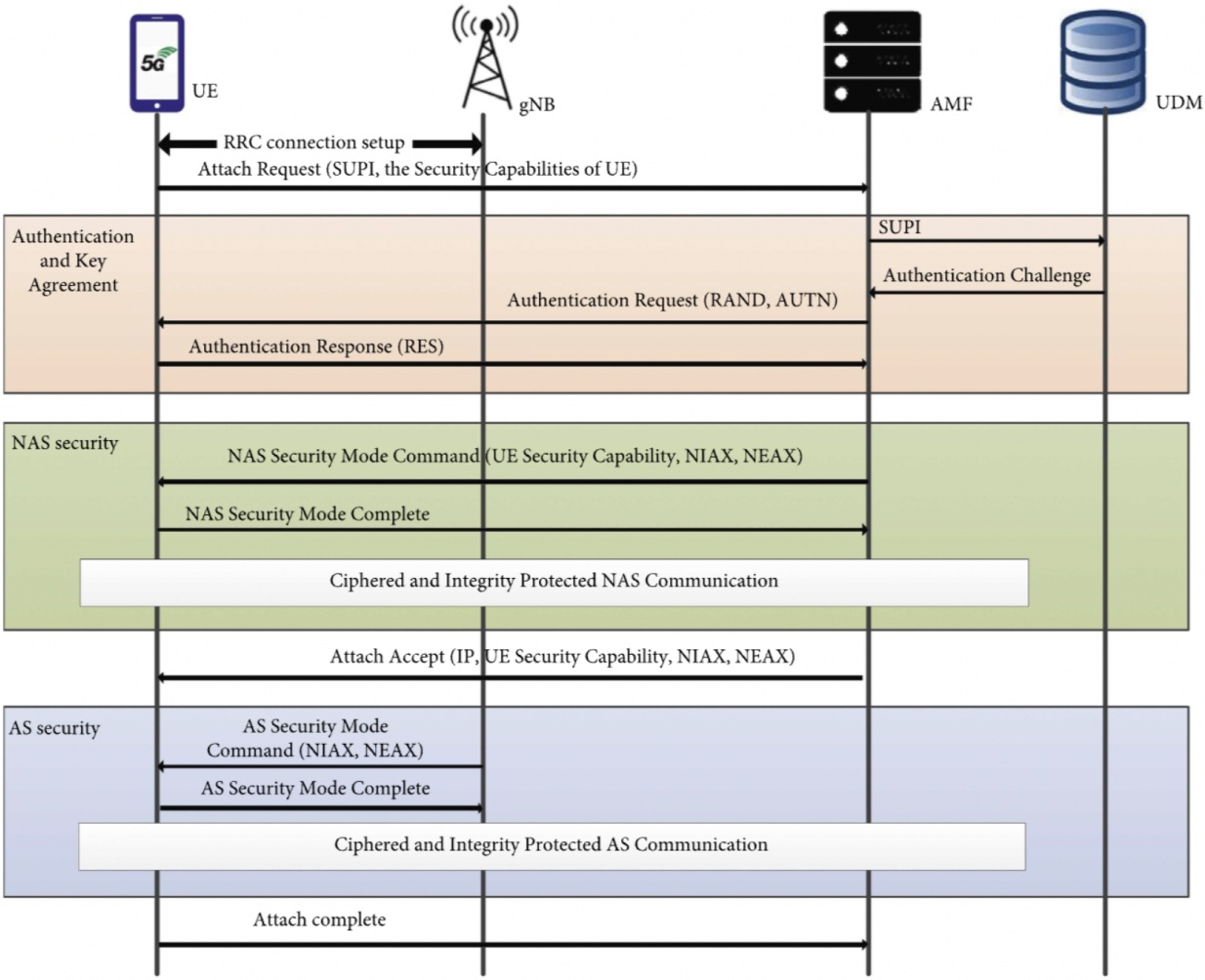


Figure 2 [image source]

5G Security

Requirements of the Network Functions

5G networks have TLS and application layer security to ensure security within the operator domain.
All 5G network functions should support TLS with server and client-side certificates.
The following algorithms should be supported by all 5g network functions [2].

TLS suite and supported algorithms

Purpose\TLS version	TLS 1.0	TLS 1.1	TLS 1.2
Key Exchange	ECDHE, RSA	ECDHE RSA	ECDHE RSA
Authentication (Certificates)	RSA ECDSA	RSA ECDSA	RSA ECDSA,
Encryption	AES 128 CBC, AES 256 CBC 3DES EDE CBC	AES 128 CBC, AES 256 CBC 3DES EDE CBC	CHACHA20 POLY1305, AES 128 GCM, AES 256 GCM, AES 128 CBC, AES 256 CBC
Hash	SHA	SHA	SHA256, SHA384, SHA

Security Requirements of 5G Network Functions

	NEAx(x=0,1,2)	NEA3	NIAx(x=0,1,2)	NIA3	Auth via SUCI	SUPI Handling	RRC Confidentiality	RRC Integrity	NAS Confidentiality	NAS Integrity	GUTI
UE	Required	optional	Required	optional			Optional	Required	Optional	Required	Required
AMF	Required	optional	Required	optional	Required	Required			Optional	Required	Required
UDM					Required	Required					
AUSF						Required					
gNB	Required	optional	Required	optional			optional	optional			
SIDF					Required						
SEAF					Required						

Ciphering and Integrity Protection

In the 5G network a new set of algorithms called the New Generation Algorithms are used. For confidentiality purposes, we have New Generation Encryption Algorithms (NEAx), while for Integrity, we have New Generation Integrity Algorithms (NIAx). These algorithms are derived from traditional algorithms like AES-CTR, Snow-3G, etc. but are used with slight modifications.

Ciphering Algorithms Derived from

NEA0 -
NEA1 Snow-3G
NEA2 AES-CTR
NEA3 ZUC algorithm

Integrity Algorithms Derived from

NIA0 -
NIA1 Snow-3G
NIA2 AES
NIA3 ZUC algorithm

Structure of NEAx/NIAx algorithms:

The ciphering mechanism for the 5G network involves generating a key stream which is then XORed with the plaintext block to create a ciphertext block. The above mention keystream is generated from the NEA and the NIA algorithms, which is then used to XOR the plaintext and produce a ciphertext; the same Keystream is used to convert the ciphertext to plaintext while decrypting the ciphertext on the receiver end.
The NEA/NIA algorithms generate the keystream and have specific inputs like COUNT, BEARER, DIRECTION, and LENGTH. Along with the KEY, these go as inputs to the NEA/NIA block and produce a KEYSTREAM.

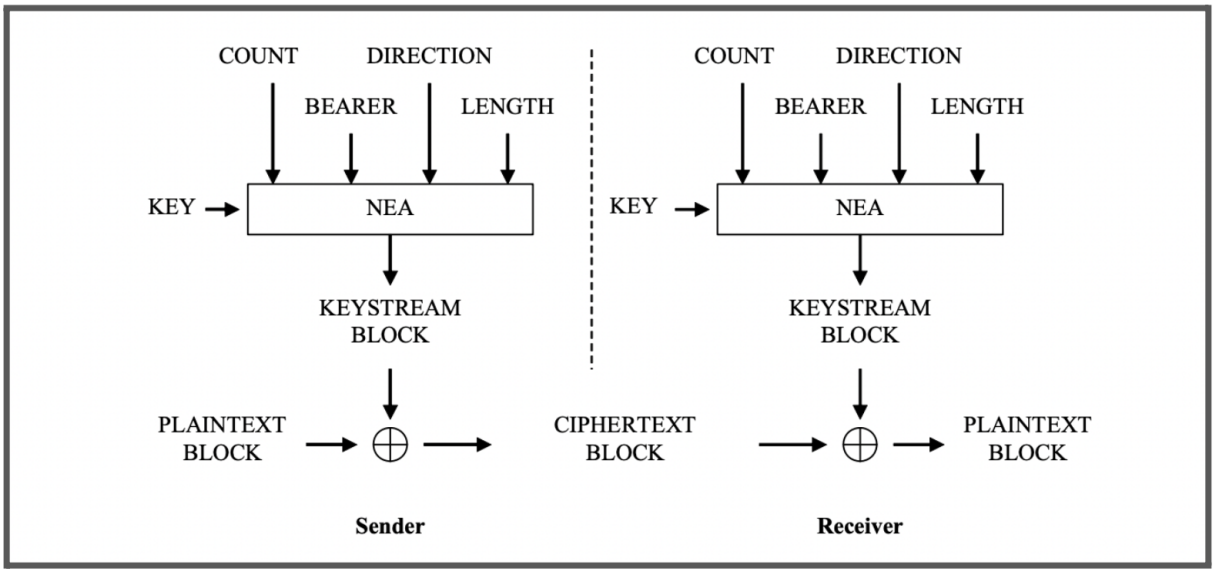


Figure 3 [image source]

The working of the NEA/NIA algorithms can be further broken down for a better understanding. The block diagram of the AES algorithm is shown below, where the AES requires a Key and a Keystream block to produce the output block, which is then XORed with the plaintext block to get the ciphertext. In the internal working of the NEA/NIA algorithms, the input apart from the KEY is used to generate the KEYSTREAM required for the underlying AES algorithm in the case of the NEA2 algorithm. This derived keystream has then used an input for the underlying algorithm like AES, Snow3G, etc.

The initial Keystream for the underlying algorithm is generated using the following structure: the leftmost bits are the most significant bits.

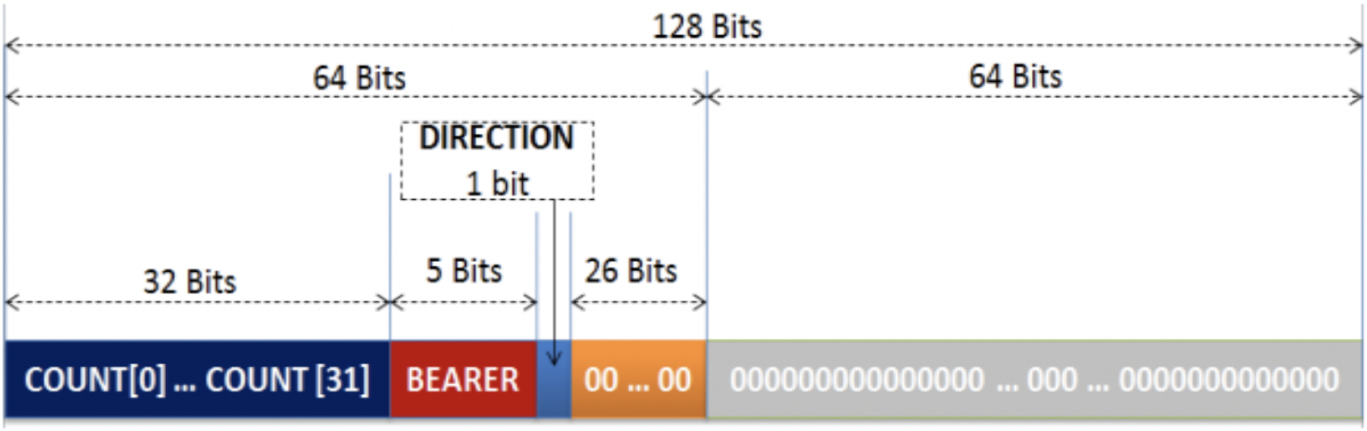


Figure 4 [image source]

The Integrity protecting algorithms follow the same structure as that of the NEAx algorithms but instead of the ciphertext, the integrity algorithms produce a 32-bit message authentication code (MAC) used for Integrity evaluation at the Receiver end. Similar to MAC, the algorithm produces an Expected MAC(XMAC) at the receiver's end.
The MAC is usually appended at the end of the messages before sending them out in the network.

AES and AES CTR

Advanced Encryption Standards or AES is a 16-byte block cipher

- The algorithm has an sp network or substitution and permutation network.
- The algorithm uses keys of sizes 128,192 and 256 bits which correspond to 10, 12, and 14 rounds(initial round key addition is excluded)
- The plaintext is XORed with the round key.
- Substitution of bytes from s box(lookup table).
- Shifting rows to add permutation.
- Multiplication with a 16-byte constant matrix
- In the final round mix column step is skipped.

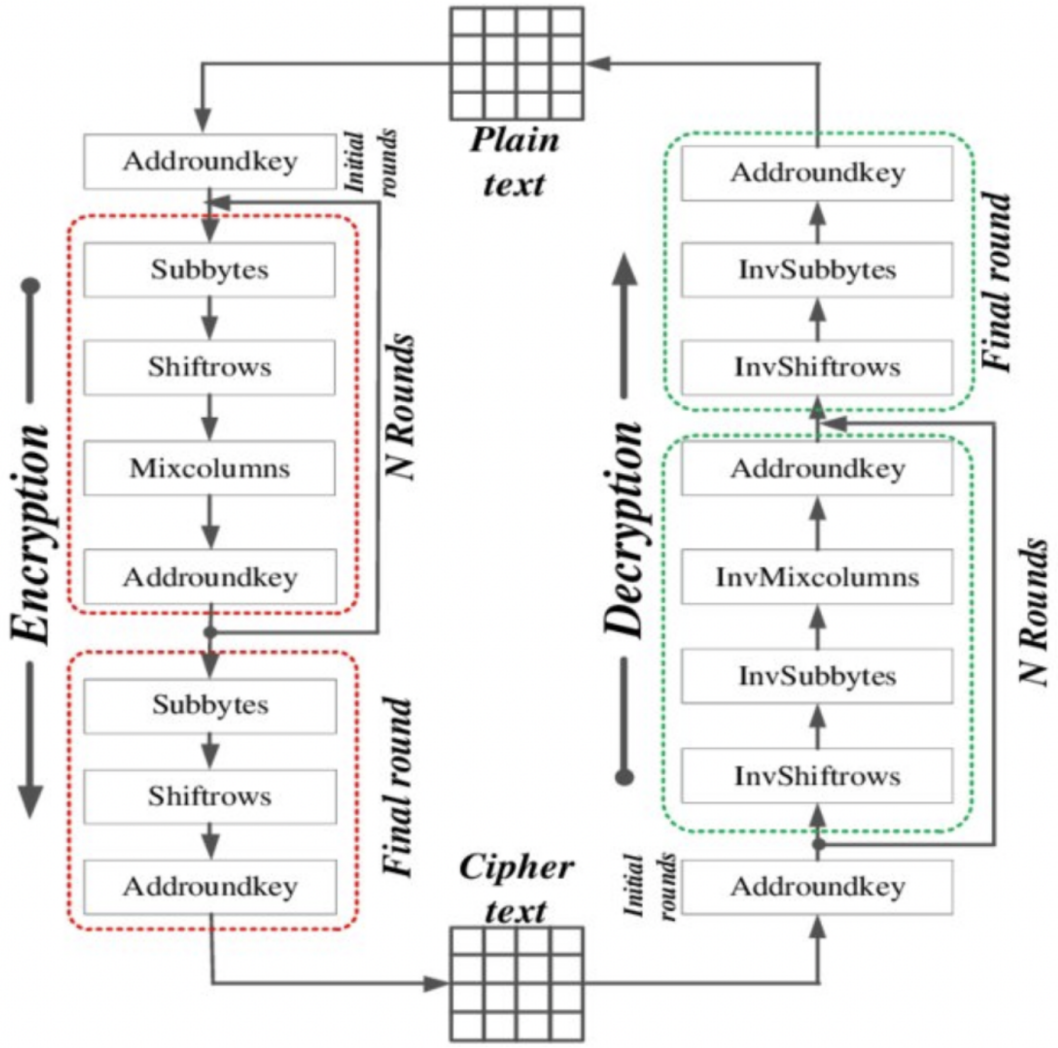


Figure 6 [Image source]

The AES algorithm has various wrappers which keep the same basic AES at the core and just change the structure of the Algorithm to derive a new algorithm for the different use cases.

AES CTR block can be used to run multiple AES blocks in parallel with all of them being independent of each other unlike the traditional AES ECB or AES CBC which are dependent on their previous ciphertext for input.

The structure of the AES CTR mode is as follows:

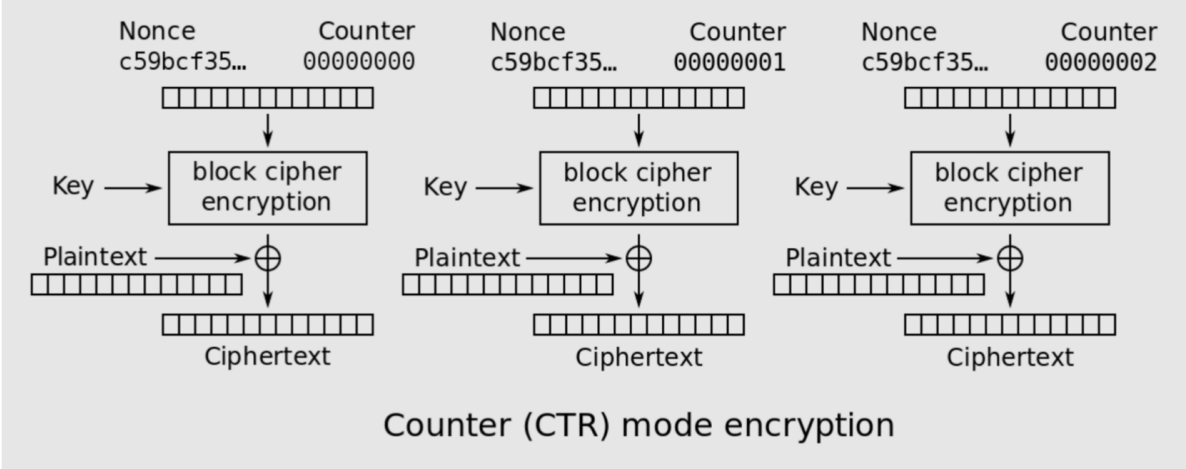


Figure 7 [image source]

Key features of the algorithm:

- Allows parallelism
- Each block is Independent of one another
- This algorithm has fast decryption as the XOR operation is done in the end and we can precompute output from cipher blocks without plaintext.
- All counters under one key should have a different value.
- No padding is done in this mode we use only MSB in case of partial blocks hence the exact length of the plaintext is revealed

Work Done in the Past Semester

Literature Review and Study

Throughout the duration of the semester I studied and understood various research papers and 5G 3GPP security specifications.

The References of the Research Papers are attached.[3][4][5][6]

Study about 5G Structure, Network Functions, and basics of 5G

I researched and gained information and knowledge about

- 5G Network structure
- 5G network functions
- How communication between UE happens with the Core Network
- How authentication of a UE is done via AMF
- Security requirements of Network Functions
- Basic Definitions like Home Environment and RAN and how are they interconnected.
- Functionality of Security Network functions and various constraints they hold ● Applications of 5G network

Study and Research on Workings of NEA/NIA algorithms

The internal working of the NEAx and NIAx algorithms used for ciphering and Integrity protection of 5G data which is explained above in the report was learned during the duration of the BTP. I learned about various network components and Network functions of 5G like AMF, AUSF, SIDF, UDF, etc.

The BTP project requires me to implement the NEA2 and ZUC-based algorithm on FPGA by the end of the entire BTP duration. Before working with the code, I researched the 128NEA2 algorithm from Security Specs. The NEA2 was a ciphering algorithm based on the AES CTR algorithm and to implement it I gained a sufficient understanding of the AES algorithm before I could head on to Understand the AES CTR implementation.

The following headers dive deeper to explain the AES and the AES CTR algorithm, but alongside, I also understood how the AES CTR was modified to generate and use the NEA2 algorithm according to the required algorithm structure.

The NEAx algorithms were developed during the development of the 4G standards for ciphering and Integrity.

NEA3-ZUC Algorithm

Before working on the code, I studied about 128 NEA3 algorithms from security specs. To gain sufficient knowledge of the algorithm, I read the 3GPP specifications and other relevant technical documents related to the algorithm. NEA3 is a security algorithm used for encrypting the data in 5G networks. It is based on the ZUC stream cipher algorithm and uses it to generate a keystream for encryption and integrity protection of user data in 5G networks.

To successfully complete the BTP project, I demonstrated proficiency in FPGA design, software development and a thorough understanding of the ZUC algorithms.

Implementation of the algorithm on FPGA

After understanding the algorithm and its design, I implemented it using C.

Implementing these algorithms on an FPGA involves designing and developing the necessary hardware and software components to enable their operation on the FPGA platform.

Implementing the ZUC algorithm on an FPGA using HLS code involved several stages of development, simulation, synthesis, and verification. I started the process by understanding the algorithm's design, requirements, and constraints. Once the algorithm is understood, the next step is to choose a hardware description language such as Verilog or VHDL, or in this case, HLS.

The HLS code is written using C, after which the code is optimized for performance and resource utilization.

Once the code is written and optimized, it is simulated to ensure correctness and performance. During simulation, the code is tested using different inputs to ensure it produces the correct output and meets the performance requirements.

After simulation, the code is synthesized into a hardware design. During synthesis, the code is translated into a set of hardware components that can be loaded onto the FPGA. The synthesized code is then loaded onto the FPGA, allowing it to execute the algorithm in hardware.

Finally, the implementation is verified by testing its functionality and performance on the FPGA. The verification process involves testing the implementation with different inputs and analyzing its outputs. If any issues are found, the implementation is iterated and improved based on the results.

Results

Apart from the Research and Studies about the 5G Security Standards. I also benchmarked the reading for ZUC algo on the Lab server. I created a testbed on the server with a Docker container where I was supposed to test and find out the latency and throughput values. After writing the code for the ZUC algo, I set a docker file for building the container. The docker containers were bound to one CPU while running to prevent context switch during the runtime to get inaccurate results. Then the ZUC code inside the container was run using a task set to ensure further the process was bound to a CPU. During the runtime, the CPU usage was checked using the top command to ensure the Test was utilizing 100% CPU during the execution of the code.

	Docker results: 1 CPU core			
	Results on Container			
Packet Size	Throughput (in MBps)	Througput(in Mbps)	Latency (in s)	latency per pkt(us)
64	15.415	123.32	4.154	4.068
128	29.874	238.51	4.296	4.293
256	54.542	436.33	4.694	4.692
512	92.012	745.33	5.566	5.496
1024	143.64	1149.12	7.129	7.124
1464	171.46	1375.34	8.546	8.535

Evaluation

Testbed Setup:

Architecture: x86_64
CPU op-mode(s): 32-bit, 64-bit
CPU(s): 32
On-line CPU(s) list: 0-31
Thread(s) per core: 2
Core(s) per socket: 16
Model name: AMD Ryzen 9 5950X 16-Core Processor
CPU MHz: 2192.454
CPU max MHz: 3400.0000
CPU min MHz: 2200.0000

Software used: Docker and GCC compiler.

Analysis:

Here we are performing the computation in CPU without involving the communication latency i.e all the computations are happening over one CPU core, and data is transmitted from the same machine, the above figures will become significant when we do involve the communication latency.

```
# taskset 0x1 ./main1464

 ______BenchMarks______

 Latency: 9.265468
Throughput(Bytes/Sec):1580060499.912147
Throughput(Packets/Sec):1079276.297754
# taskset 0x1 ./main1464

 ______BenchMarks______

 Latency: 9.384980
Throughput(Bytes/Sec):1559939392.518684
Throughput(Packets/Sec):1065532.371939
# taskset 0x1 ./main1464

 ______BenchMarks______

 Latency: 9.448267
Throughput(Bytes/Sec):1549490504.449123
Throughput(Packets/Sec):1058395.153312
# taskset 0x1 ./main1464
```

The benchmark reading was taken multiple times to ensure no erroneous reading was taken. The screenshot of the benchmark for 1464 byte-sized packet as input to AES GCM

Work Done in the Current Semester

The literature review in the previous semester helped me understand the problem statement. After the literature review, I decided to implement the cryptographic algorithms required to complete the crypto library for our project.

The 5G architecture required the implementation of the following algorithms:

Ciphering Algorithms Derived from

NEA0 -
NEA1 Snow-3G
NEA2 AES-CTR
NEA3 ZUC algorithm

Integrity Algorithms Derived from

NIA0 -
NIA1 Snow-3G
NIA2 AES
NIA3 ZUC algorithm

We decided on Implementing the NEA2, NIA2, NEA3, and NIA3 algorithms and
divided the work among ourselves.

Motivation:

Currently, the security algorithms can be offloaded to dedicated hardware chips present on the SmartNIC. These algorithms are implemented on FPGA, operating efficiently and achieving a throughput of 100GBps. However, due to evolving requirements, the rigidity of fixed-function ASICs becomes impractical, necessitating programmable and flexible hardware accelerators. FPGAs, being both flexible and programmable, emerge as a suitable solution.

As TLS incorporates multiple algorithms simultaneously, the Edge Server must support a variety of these algorithms concurrently. To address this, we propose a framework that dynamically determines which algorithms to place on the FPGA, based on changing requirements derived from specific metrics.

To optimize FPGA resource utilization in a multi-algorithm environment, we plan to work with two versions of each security algorithm: one optimized for throughput and another for accommodating multiple algorithms. The allocation of more hardware resources

enhances performance but limits the number of algorithms that can be placed on the FPGA.

The capability to dynamically reconfigure the FPGA (DPR) without disrupting operations allows on-the-fly algorithm changes based on traffic proportions. Therefore, we are developing a Scheduler that determines FPGA algorithm configurations based on monitored metrics fetched by a controller program.

These metrics are obtained from incoming packets and hardware utilization, guiding the Scheduler's decisions regarding adding, replacing, or reconfiguring specific hardware IPs for particular algorithms. These metrics also serve as feedback for FPGA modifications.

To further refine the Scheduler's performance, we aim to establish the proportion of hardware IPs needed for each algorithm by analyzing incoming traffic patterns. To achieve this, a testbed for experimentation is being enabled, enabling us to build a Scheduler capable of dynamically reconfiguring the FPGA based on packet requirements.

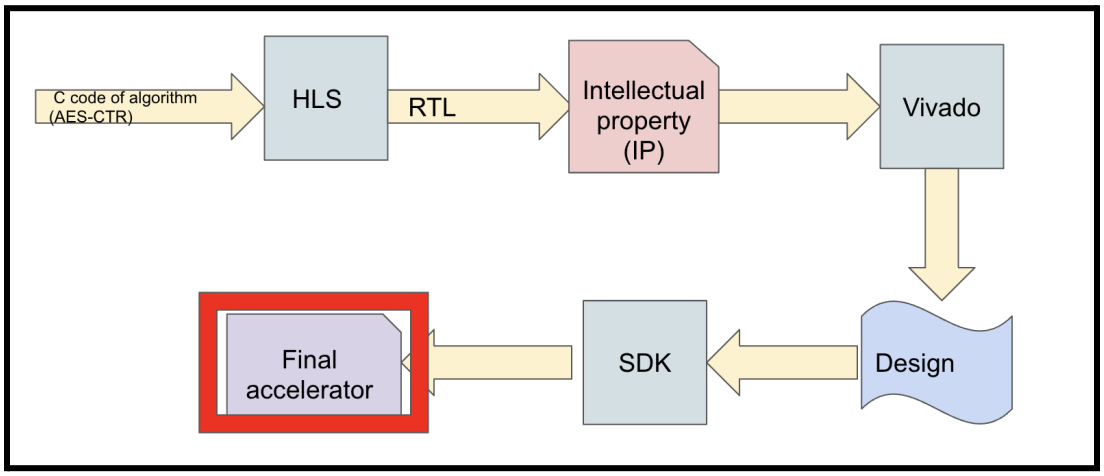
NEA2 Algorithm

The NEA2 algorithm was based on the AES CTR algorithm, and hence, I started out by implementing the C Code for the algorithm.

I took around three weeks to implement the AES CTR algorithm in C code and verified the algorithm with standard test cases from NIST Documentation [9].

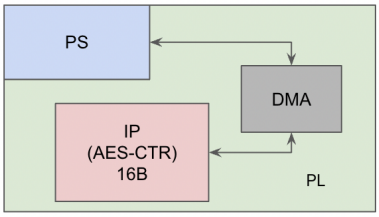
After verification from the Test Cases, I moved on to implement the HLS code from the AES-CTR C code, where I was required to clean the C code to make it HLS compatible by removing the function calls from the Libraries that were not compatible during the compilation of HLS code on FPGA hence I removed the functions like memcpy and malloc and rewrote code for malloc, memcpy, memcmp and some other derived function from scratch.

After the code was modified, I started parallelizing the C code and implementing pragmas in the C code to generate HLS code, and after some help from our lab mates, I was able to implement AES CTR and generate HLS code for it.



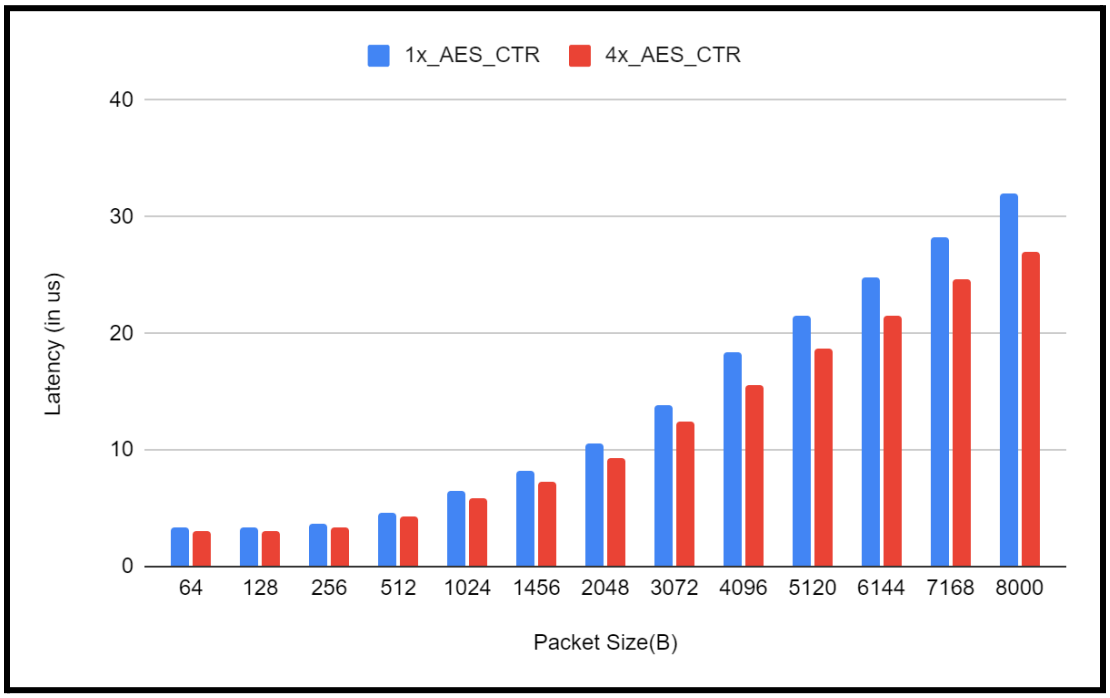
Testbed Setup:

- Zynq UltraScale+ MPSoC ZCU106
  - Based on ZU7EV Ultrascale+
  - Quad-core ARM cortex A53 processing system (PS)
  - 504K system logical cell
  - 38Mb Distributed PL Memory
- **~ 1K lines of hardware code**
  - HLS source for AES-CTR
  - HLS verification testbench
  - Driver code for PS-PL communication on SDK



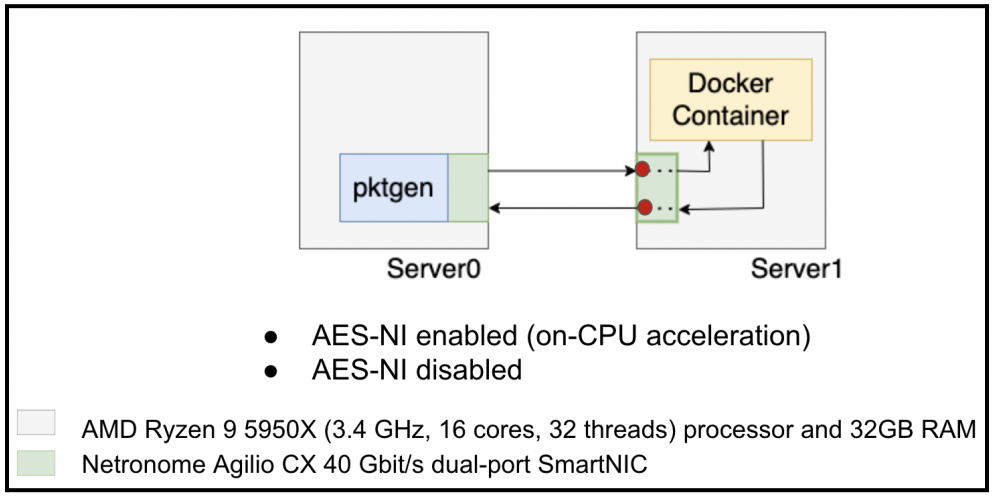
The following readings were found after testing the AES CTR HLS code on FPGA:

Config: AES_CTR		BRAM -32 KB
	latency(us)	latency(us)
Packet Size(B)	1x_AES_CTR	4x_AES_CTR
64	3.33	3.076
128	3.35	3.076
256	3.613	3.339
512	4.659	4.295
1024	6.532	5.817
1456	8.186	7.259
2048	10.572	9.249
3072	13.79	12.431
4096	18.334	15.626
5120	21.54	18.641
6144	24.754	21.593
7168	28.23	24.722
8000	31.972	27.018


1x_AES_CTR
4x_AES_CTR
40
30
20
10
0
Latency (in us)
64
128
256
512
1024
1456
2048
3072
4096
5120
6144
7168
8000
Packet Size(B)

Motivation Experiment Setup:

I



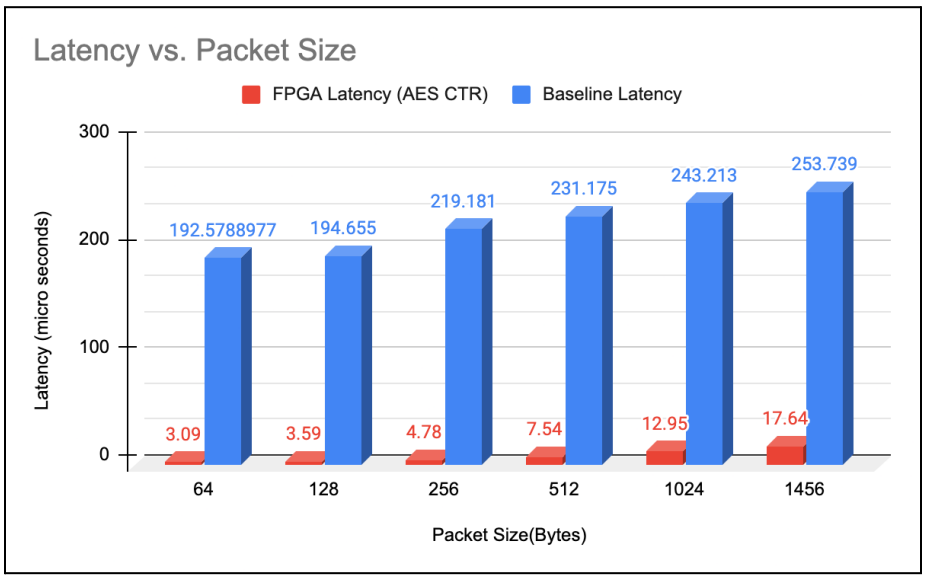
I then implemented a Baseline setup to calculate end-to-end latencies on a docker container. With these readings, we would be able to analyze OS and container overheads, which would have increased overall latencies as described in the abstract of our Project I have then written the AES CTR code with OpenSSL EVP API so that we can compare our IP with the standard AES CTR implementation.

Later, I containerized the OpenSSL C code and used it to run our Experiment with the following readings:

Avg Latency(in micro sec)	Min Latency(in micro sec)
192.5788977	118.609
194.655	123.723
219.181	168.913
231.175	171.018
243.213	194.448
253.739	193.024

Analysis:

We then calculated overheads on the commodity server due to Host OS and Container compared to the FPGA results, which are shown in the following plot, which compare the two results:



The overhead latencies due to the OS and the container came out to be 93 to 98 percent of the total latency generated during the processing while the actual processing latency ranged from 3 microsecond to 18 microsecond, which validates out motivation.

Future Work

My Next Steps in the project are the following:

- ☐ Implementation of HLS Code for AES CTR 8X version.
- ☐ Implementation of Integrity algorithms NEA3, NIA1, NIA2, and NIA3.
- ☐ Completion and Implementation of Cryptographic Library.
- ☐ Implement a framework for Automatically Reconfiguring our FPGA with changes in workload and efficient power utilization.
- ☐ Developing APIs for our Cryptographic Library for Developers.

Bibliography

[1] Kundel, Ralf & Meuser, Tobias & Koppe, Timo & Hark, Rhaban & Steinmetz, Ralf. (2022). User Plane Hardware Acceleration in Access Networks: Experiences in Offloading Network Functions in Real 5G Deployments. 10.24251/HICSS.2022.894.

[2] 3GPP TS 33.210, “Network Domain Security (NDS), IP network layer security”.

[3] https://www.etsi.org/deliver/etsi_ts/133500_133599/133501/15.04.00_60/ts _133501v150400p.pdf

[4] https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf

[5] https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38a.pdf

[6] Run Zhang, WenAn Zhou, Huamiao Hu, "Towards 5G Security Analysis against Null Security Algorithms Used in Normal Communication", *Security and Communication Networks*, vol. 2021, Article ID 4498324, 15 pages, 2021. https://doi.org/10.1155/2021/4498324

[7] https://uk5g.org/media/uploads/resource_files/5G_Architecture_and_Security _technical_report_-_04Dec18.pdf

[8] https://www.eecis.udel.edu/~salehi/files/asee13_lte.pdf

[9] NIST Standards for AES Modes of Operation

[10] RFC 4493 for AES CMAC Implementation