Fall 12-2021

PRIVACY-AWARE AND HARDWARE-BASED ACCLERATION AUTHENTICATION SCHEME FOR INTERNET OF DRONES

Tom Henson

PRIVACY-AWARE AND HARDWARE-BASED ACCLERATION AUTHENTICATION SCHEME FOR INTERNET OF DRONES

by

Tom E Henson

A Thesis
Submitted to the Graduate School,
the College of Arts and Sciences
and the School of Computing Sciences and Computer Engineering
at The University of Southern Mississippi
in Partial Fulfillment of the Requirements
for the Degree of Master of Science

Approved by:

Dr. Amer Dawoud, Committee Chair
Dr. Ahmed Sherif
Dr. Ramakalavathi Marapareddy

December 2021

ABSTRACT

Drones are becoming increasingly present into today's society through many different means such as outdoor sports, surveillance, delivery of goods etc. With such a rapid increase, a means of control and monitoring is needed as the drones become more interconnected and readily available. Thus, the idea of Internet of drones (IoD) is formed, an infrastructure in place to do those types of things. However, without an authentication system in place anyone could gain access or control to real time data to multiple drones within an area. This is a problem that I choose to tackle using a Field Programmable Gate Array (FPGA) that accelerates the k-Nearest Neighbor (kNN) encryption algorithm making it a hardware component. This will allow me to synthesis and implement the three parts of my privacy-aware and hardware-based authentication scheme for internet of drones. I use Vivado and Vivado HLS to obtain results for my authentication scheme. My scheme was able to perform large computational expensive tasks faster than other proposed IoD schemes.

ACKNOWLEDGMENTS

I would like to acknowledge Dr. Amer Dawoud and Dr. Ahmed Sherif for their support and guidance throughout this entire project. Their insight on FPGA systems and the kNN algorithm helped me a great deal. Stephen Adams for his support and direction on many problems that I have had occur. I would like to add a special thanks to John Anttonen for pointing me in the direction of military projects that in some way correlate to my project. I would also like to add another special thanks to David Hellmer for his help with C++/C code problems that I had come across.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

LIST OF SCHEMES

LIST OF ABBREVIATIONS

FPGA	Field Programmable Gate Array
kNN	k-Nearest Neighbors
IoD	Internet of Drones
HLS	High Level Synthesis
C++	General purpose programming language
C	General purpose programming language
MatLAB	Programming and numeric computing software
Vivado	FPGA design and synthesis software
RA	Registration Authority

CHAPTER I – INTRODUCTION

Drones are widely becoming accessible and more active in everyday life. (Gharibi) They are being used in transportation of goods, sporting events such as drone races, government surveillance, wildfire control, and much more. (Tiberiu, Jung) The potential for drones is limitless in their use. As we progress as a society, so does our technology and we are able to interconnect many things and drones are no exception. Companies like amazon, UPS, US Department of Transportation and Verge Aero are pushing for increasing drone presence to implement fleets of drones to better their business and interests. (Tiberiu, Jung, Wisel)

This is one of the reasons why a secure IoD environment is needed in areas with heavy drone activity that is near or in civilian populations. In addition to the increased use of drone activity, so do the risks of hazards and potential threats. Adversaries can use their skills to cause many harmful and potentially deadly attacks to the system in unprotective networks. Some examples of this are drones found spying in private citizens property, drug smuggling along the US-Mexico border, and even in bombing attacks via attached explosives to the drone. (BBC News) Companies like Amazon risk the locations of their drones being made aware, and adversaries can then steal goods that are being delivered.

I propose one solution that could potentially solve some of these problems. My scheme is implemented over an IoD network that requires drones to register before use within a system. The privacy-aware and hardware-based authentication scheme uses the kNN encryption algorithm to encrypt the drone's identification information using a secret key. Each drone gets a unique key that is created by a trusted entity within the system.

There are two kNN encryption processes, one done by the drones during an authentication request and one by the trusted entity after drone registration. The encrypted identification information that is created by the trusted entity is stored on a server to be used later for dot product verification. When an authentication request is made, the drones send their encrypted identification information to the server and then it will be verified before any information can be accepted/rejected by the server. This process will make it difficult for adversaries to attempt to infiltrate the system.

I implement this authentication scheme on a FPGA as a hardware component. The FPGA can perform parallel processing, thus accelerating the kNN algorithm. Since this is an added component to the drone, it can be turned off when not in use to save power. Power is one of the major resources of a drone, thus necessary to save as much power consumption as possible. I chose the FPGA because of its ability to be reconfigured and is flexibility during use. When the device is not needed, it can easily be powered down. The other reason for implementing the authentication scheme on this device is that the FPGA will take over the computational heavy tasks such as the encryption process of the kNN algorithm away from the drones CPU. This allows the drones CPU to focus on other important tasks at hand.

I was able to accelerate the kNN algorithm and perform large computational tasks with exceedingly small computational cost. The dot product search achieved small computation costs while having to search through 200 other drones' encrypted identification information uploaded from the trusted entity. My scheme is very flexible and allows for different encryption sizes as well as affordable pricing availability. Some of the components can range from fifty dollars to several thousands of dollars.

The rest of this thesis is structured as followed; the background of different topics of my work, a literature review of FPGA hardware acceleration and other IoD authentication schemes, system model and proposed scheme that goes further in detail of the kNN authentication scheme and model, and methodology and procedure on how my results were obtained with the different software's that was used and the discussion and conclusion of my results.

CHAPTER I - BACKGROUND

Field Programmable Gate Array

FPGA came to be through the creation of programmable logic devices in the 80s. What makes the FPGA unique compared to other devices is that its hard-wired programming can be reconfigured while other devices have a fixed hardware during the manufacture process and cannot be changed. (HardwareBee) This is extremely useful when an FPGA has been configured for a certain task and there is something wrong with the programming. The FPGA can simply be reconfigured once the problem has been found and fixed. Today, FPGAs are utilized worldwide for many different aspects such as the acceleration of different computational tasks and for development into artificial intelligence. (Touger)

Hardware Acceleration

Hardware acceleration is a fast-growing process that is being used in many areas such as artificial intelligence, video game design, and in my case acceleration of the kNN algorithm. The definition of hardware acceleration is the process of offloading computational heavy tasks onto a special device to decrease the computational cost of the tasks that would normally take a CPU longer to compute. The most common devices used in hardware acceleration are graphic processing units (GPUs), Field Programmable Gate Arrays (FPGAs), and Application-Specific Integrated Circuits (ASICs). (OmniSci) The FPGA hardware acceleration is achieved from its interconnecting logic blocks that can perform real time processing in parallel. The main difference in a CPU and FPGA is the amount of data that can be parallel processed within each device.

Parallelism

Parallelism is the process of running several tasks at the same time to achieve faster results. This process is used to speed up the execution of programs by having multiple parts of a task split up and assigned to different processes to implement in parallel. Parallel can be performed by using Multi-Core CPUs, GPUs, and FPGAs.

k-Nearest Neighbor algorithm

The kNN algorithm was created by Evelyn Fix and Joseph Hodges in 1951 and then later expanded upon by Thomas Cover. It is mostly being used in machine learning as a classification and regression model, (Peterson). It is also known as the supervised learning algorithm and works by finding similarities between two data points (Analytics Vidhya).

Internet of Drones

IoD is a layered network control architecture created to maintain control and access drones over a network within an area or areas, (Gharibi). This is used to monitor registered drones in an area or areas to reduce risks such as unauthorized use, collision hazards, etc. FCC has many requirements for recreational drones and any above 400ft is considered uncontrolled airspace. This reduces the risk of private citizen drones flying into commercial airliners etc., (Secondary nav.). There are many new laws being implemented to control risks of increasing presence of drones, like the law stated above.

CHAPTER II – LITERATURE REVIEW

FPGA Hardware Acceleration

Hardware acceleration is a growing topic in today's technological world. This is the process in which high computationally expensive tasks are offloaded from the CPU onto a hardware component to accelerate the task to improve performance and efficiency. Hardware acceleration is common in video games where CPU offloads the video games graphics onto the GPU to do most of the work. This component is built for this type of work to allow the CPU to focus on different computational tasks. Other examples include bit mining, artificial intelligence, and DNA mapping. These are very heavy tasks that the hardware acceleration can perform in less time than it would take a single CPU. The FPGA is naturally built to accelerate its tasks by the way it is designed. This is a specialized device that can be reconfigured to meet different needs and is capable performing parallel processing.

In Dave et al., they tackled a problem to optimize the matrix-matrix-multiplication using hardware acceleration through the FPGA and an instruction set architecture called PowerPC. They connect directly to the system memory bus allowing for bandwidth up to 800 MB/s. Their software pipelined system would compute the matrix multiplication task faster with smaller size matrices but would be less efficient compared to the hardware implementation on much large matrix sizes. Finally, they achieved some level of optimization but stated they could further improve on this. This paper is one of the works that I considered when implementing my scheme. The kNN encryption algorithm requires several matric multiplication functions to occur in the creation of the encrypted identification indices.

Owaida et al. proposed a solution to lowering the latency of data processing pipelines using FPGA based hardware acceleration, investigate data processing and machine learning in search engine pipelines, and propose an alternative solution through FPGA implementation of gradient boosted decision tree ensembles. This will allow for an efficient way to process much larger results while combining distinct parts of the process to achieve lower latencies. They test their scheme by using a large data analytic software called H20. The data for testing is performing multiple queries for the flight routes and finding quick results for multiple data points. Their design implements two different techniques to speed up the process of the gradient boosted decision tree and the XGboost tree. Both trees are parallelized including the data sets that hold much of the routing information. Next, it removes overhead from high-rate memory that normally a CPU would have, to the memory structures of the FPGA that is highly customizable. Their work shows that the FPGA far outclasses the CPU used in their testing by several factors.

Imagine processing is a heavy computational task that is applied to many fields such as medical, military, and satellite imaging. Stratakos et al. take a dive into the use of FPGAs in medical imagine processing and optimizing the parts of the process that are heavily computational. They propose a hardware and software implementation of the imagine processing where the software application will handle the less computational expensive tasks, and the hardware implementation will handle the heavier loads. The use of direct memory access is used to keep a low latency while keeping a high throughput between communications of the hardware design and software design. Like Stratakos, who had issues where large images would use up all the BLOCKRAMs resources and

needed to be downsized, I had issues where I had to downsize the ID size to one that did not fill up the FPGA resources. Finally, they achieved optimizing the imagine processing using FPGA devices while proving higher performances when compared to standard software implementations during image processing.

Gielata et al. explore using FPGA hardware acceleration for AES encryption and Rijndael algorithm. They designed a pipeline architecture that would encrypt and decrypt the different modules. In their experimentation the coding modules were synthesized and implemented separately from the decoding modules in 10 rounds parallel. They were able to achieve a higher throughput than regular software implementation of a similar design. This allowed for a larger amount of data to be processed while executing the AES encryption much quicker. It is worth noting, the coding module performed faster with more throughput than the decoding process of the encryption algorithm.

In the work proposed by Ernst et al, they use the FPGA device as a hardware component that is connected to PCI. The FPGA is used to process the EC point multiplication function of the elliptic curve public key algorithm. This is the most computational extensive part of the encryption. Instead of first designing their algorithm in C/C++ they take a different approach that generates synthesizable hardware descriptions. This is like Vivado Design suite when creating a block design. They use varying key sizes as well as different FPGA hardware during testing. Their work was greatly increased using the FPGA based Crypto Processor that can generate synthesizable VHDL code that is implemented on the FPGA devices to accelerate the elliptical curve public key encryption algorithm.

Lastly, hardware acceleration is gaining influence in the computational world as bigger and harder to solve problems grow. Image resolutions are becoming quite large requiring a considerable number of resources and time to process. Algorithms and equations that once took CPUs a great amount of time to process can now be implemented much quicker. Super computers are being built across the world to solve complex problems through parallel computing. GPUs are increasing with the number of resources that can be placed upon on card. This increases its ability to perform much better through parallelism by using one of its greatest features, CUDA cores, to perform difficult tasks. FPGAs are becoming increasingly present as the ability to put more components together within a small area as well as everything else. Unlike all these other devices, the FPGA is highly customizable to fit different needs as a physical hardware component while having high throughputs and executing tasks much quicker than standard serial means.

Encryption schemes for IoD

Internet of drones is a network architecture that is designed to control unmanned aerial vehicles in a controlled airspace. It is also used to control the information that is being communicated throughout the network. In this part of the literature review I explore the different proposed authentication schemes that are attuned for IoD. A comparison of their findings and mine are found in the comparison results section in chapter VI of this report. First, Wazid et al. propose a lightweight remote user authentication and key agreement scheme for internet of drones.

Wazid proposes a scheme for internet of drones that is user-based authentication scheme that uses key agreements in the scheme. They use a one-way hash function and

bitwise XOR operation. On the user side of the authentication scheme, they use fuzzy extractor to verify the users' biometrics. They boast of being resistant to attacks such as privileged-insider and password guessing, impersonation attacks, and a few others. Their scheme is composed of seven phases. The first step is the server will start the registration of all the drones, next users will register with the server. Then users will login with their credentials. The server will verify the credentials and deny or approve the login. There are other phases, but they do not relate to the direct process. They are for password and biometric updates and the addition of other drones to the system. The computational cost of their scheme on the server side is 2.56 ms while my scheme is in microseconds. Next, Srinivas et al. propose a lightweight authentication scheme for IoD that uses Temporal Credential-Based Anonymous and applies real-or-random models.

Srinivas scheme works by only allowing register users to use the services of the drones that are already registered with the server. Just like Wazid's scheme all drones are preregistered with the server. In this scheme all the remote drones have unique keys that are used in the authentication process. Srinivas scheme applies a three-factor technique that includes smart cards, user passwords and biometrics. They use a one-way hash function and apply fuzzy extractor during the biometric verification process of the scheme.

As previously stated, this authentication scheme has similar steps to Wazid, pre-deployment, user registration, login and authentication, and stages to update user information, add new drones, and replace items that were stollen or lost. Srinivas boast to protect against attacks such as stolen devices, impersonation attacks, man-in-the-middle attacks and many more. The approximation of the server-side computational time

for their scheme is 2.88ms seconds which is slightly slower than the earlier scheme reviewed above. The next paper proposes a lightweight authenticated key exchange protocol for IoD.

Tanveer et al. propose a lightweight authentication scheme that uses key exchange protocol that they label as LAKE. This scheme is composed of six phases that are like previously described schemes: registration, user authentication and key exchange, update user information and biometrics, and the ability to add more drones to the scheme. Their scheme uses a well-known authentication process as AEGIS encryption scheme. They also use a one-way hash function and bitwise XOR operations in their scheme. Tanveer boast that their authentication schemes protect from attacks such as password and biometric update attacks, offline password-guessing, identity-guessing attacks, and others. Note that Tanveer uses a larger hash function than both Wazid and Srinivas but require less computational overhead in their server-side encryption. The server-side computational cost that Tanveer claims is 0.174 ms. Finally, the last paper proposes a lightweight authentication and key agreement scheme for Iod.

Zhang et al. propose a similar one-way hash function and bitwise XOR operations, that happen when the drones and users mutually authenticate one another. They boast their scheme can potentially withstand various attacks such as impersonation, server spoofing, drone capture etc. Their scheme is composed of four phases: setup, user registration, drone registration and lastly authentication.

As a final note, a strong authentication scheme is necessary in the IoD environment. This reduces the risk of adversaries trying to infiltrate the system to do harm within and without the architecture. Thus, having various parts of the scheme

passed on as a hardware component specialized in parallel processing will allow for larger throughput with lower latency times. This is recognized in the results section of this report where I make a comparison of similar works to my own. I also compare the server side of my work as if it were compiled on the Kintex 116 FPGA, Nexys A7-100T FPGA, MatLAB and Visual Studios 2019 Enterprise.

CHAPTER III – SYSTEM MODEL

Network Model

The network model is composed of three main entities that work together to achieve one goal, a privacy-aware and hardware-based authentication scheme for IoD.

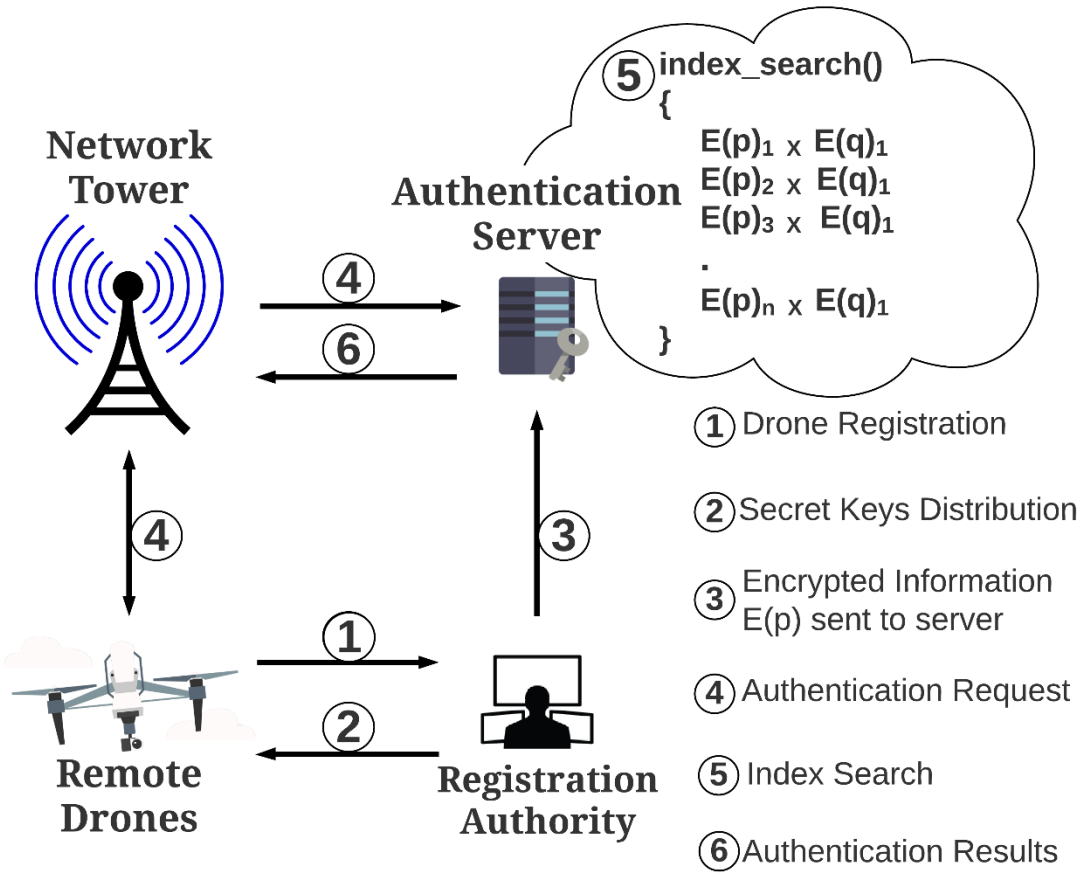


Figure 3.1 Network Model of Authentication Scheme

The three main entities as shown in the above figure are the remote drones, registration authority (RA) and the Authentication server. The network tower is purely communication purposes within this scheme. The first entity is the remote drone that must register with the registration authority before it can perform any kind of tasks within

the IoD area. Once this step is complete the drone can make a request to the authentication server when information needs to be passed along or requested. Before a request can be sent, the remote drone must perform the kNN algorithm on its identification information to create an encrypted identification index E(q) that can be sent to the authentication server for verification. The second entity is the registration authority whose sole purpose is to take in drone identification information, encrypt such information using kNN algorithm, upload encrypted identification index $E(p_i)$ for all drones registered to the authentication server, and create and distribute keys to all drones. The last entity is the authentication server, which functions as verification check and storage for all drone encrypted identification indices $E(p_i)$. The authentication server will receive the encrypted identification information index E(q) from the drone and perform the dot product search between it and all drone stored encrypted identification indices $E(p_i)$ received from the RA. If a match is found the drone can then send or request information to the authentication server, however, if the server does not find a match the drone will no longer be able to send any information over the network.

Threat Model

The types of threats my scheme will face are attackers that are "honest but curious." This will come from inside advisories within the authentication scheme such as the authentication server or other drones. "Honest but curious" means that the attackers will follow the proper protocols of the scheme and will not try to change it or deviate from the process. However, they will try to gain insight and learn all information possible throughout the scheme. This includes information such as the keys used in the kNN algorithm that are created and distributed by the RA and the drone's identification

information. The purpose of this authentication scheme is to keep adversaries that are described above from gaining access to such information so that the system and its information remains private.

Design Goals

In this section the three design goals are defined here as well as how they apply to this authentication scheme.

Scalability and Efficiency

The scheme must be able to perform the dot product search over a large amount of data in a short bit of time. In this case, the index search is performed using two hundred different drone encrypted identifications information $E(p_i)$ from the RA. The experiment used varying sizes of encrypted data between 1.024 kbits and 101.032 kbits depending on which FPGA device that is implementing the search.

Confidentiality Identification Information

Important identification information that is passed between the RA to authentication server or drone to authentication server should be confidential and private from adversaries trying to gain insight on its contents. This information is hidden inside the encrypted indices created via all drones and RA making it difficult for advisories to decipher.

Unlinkability of Authentication Request

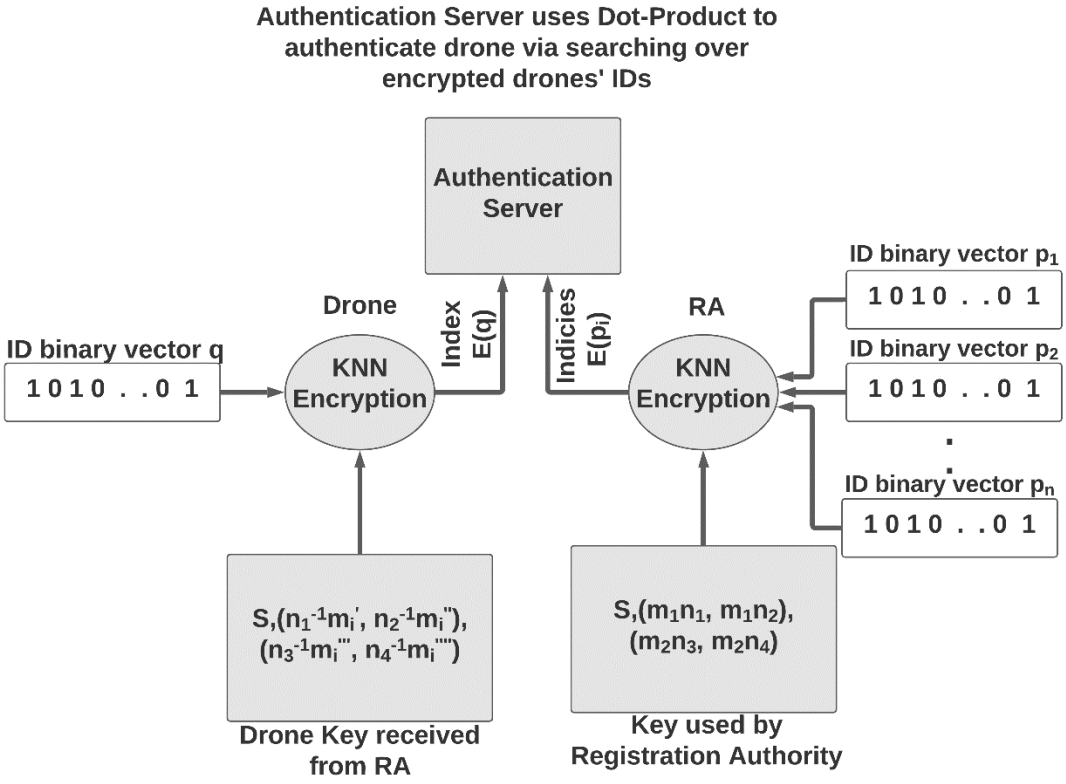
Inside the kNN encryption process for the remote drones, random numbers are generated during the encryption of the identification information. Each time a drone makes a request to the authentication server different random numbers are created making no two-authentication request alike thus unlinkable.

CHAPTER IV – Proposed Scheme

Overview

A brief overview of the proposed scheme is defined below.

1) **Drone Registration**: Drone gives unique identification information to registration authority. This is in the form of a binary vector.
2) **Key Distribution**: Registration authority creates and gives each drone a unique key.
3) **RA Identification Encryption**: Registration authority performs kNN encryption algorithm on drones' identification information creating the index $E(p_i)$.
4) **Upload of Index from RA**: The registration authority uploads all drones' encrypted identification information to the authentication server.
5) **Authentication Request**: Once a drone has received the key from the RA, it can now use the key to encrypt its identification information using kNN algorithm. This creates the E(q) encrypted index that is sent to the authentication server during a request.
6) **Dot Product Index Search**: Once the authentication server receives the drones encrypted index E(q), it takes the drones encrypted index E(q) and performs dot product on all stored indices $E(p_i)$ to find a match.
7) **Authentication Results**: If a match is found, the server will then accept the authentication request allowing the drone to receive or give information to and from the authentication server. If denied, the drone will no longer be able to send information within the system.



Scheme A.1 Proposed Authentication Scheme

Drone Registration

The first step of the authentication scheme is registration of the drone with the RA. The drone must give the RA its unique identification information q. In the implementation of this in Vivado HLS, the q is a binary vector and is tested using varied sizes: 4, 32, 64, 128, 256, and 512. The varied sizes allow for various levels of encryption. The Nexys FPGA can only handle sizes of 256 and below.

Key Distribution

Once the RA receives the drone's identification q it will create the parts of the key that is used in its own implementation of the kNN encryption and the drones. First, a binary vector S is created of size n. The n is determined by the size of q. The first six

matrices are arrays type double composed of variables between 0.01 and 1 labeled (m_1, m_2, n_1, n_2, n_3, n_4). These random numbers are created by using a seeded 32-bit Mersenne Twister, mt19937. The seeded numbers in the experiment can be replaced with variables such as the internal temperatures of the board at that time during process. These matrices and the binary vector are what the registration authority will use to create its key that will be used to encrypt the drones' identification information before it is uploaded to the authentication server. Thus, the RA's key is composed of [S, m_1n_1, m_1n_2, m_2n_3, m_2n_4].

Next, the secret key for the drone is created using the inverse of the matrices created earlier labeled, (m_1^{-1}, m_2^{-1}, n_1^{-1}, n_2^{-1}, n_3^{-1}, n_4^{-1}). Also, for each of the drones' key, the RA will generate four random matrices ($m_i', m_i'', m_i''', m_i''''$), so that $m_1^{-1} = m_i' + m_i''$ and $m_2^{-1} = m_i''' + m_i''''$. Finally, the drones secret key is composed of [S, $n_1^{-1}m_i'$, $n_2^{-1}m_i''$, $n_3^{-1}m_i'''$, $n_4^{-1}m_i''''$].

RA Identification Information Encryption and Upload

The next step of the proposed scheme is the RA's application of the kNN encryption to the drones 'identification information. After encryption, the RA will upload the encrypted indices to the server for each drone registered. First, the RA receives the unique id of the drone that is labeled p_i and sent to the split function to create two row arrays $p_i'[k], p_i''[k]$. The S vector is also sent into the split function and performs the following tasks:

```
for k < size of S, k++
        if (S[k] == 1)
                p_i'[k] = p_i[k]
                p_i''[k] = p_i[k]
        else if (S[k] == 0)
```

$p_i'[k]$ = random number (0.01 and 1)

$p_i''[k] = p_i[k] - p_i'[k]$

After the p_i has been split the RA will perform matrix multiplication four times to create four indices of the same size as p_i which is shown in the following equations below.

$$I_1 = p_i' \; x \; (m_1 \; x \; n_1)$$

$$I_2 = p_i' \; x \; (m_1 \; x \; n_2)$$

$$I_3 = p_i'' \; x \; (m_2 \; x \; n_3)$$

$$I_4 = p_i'' \; x \; (m_2 \; x \; n_4)$$

Then all four indices are combined into one index that is now size n x 4. This index is labeled $E(p_i) = [I_1, I_2, I_3, I_4]$ and is uploaded to the authentication server, where it will be stored with all other drones' encrypted indices.

Authentication Request via Drone

Once a drone is registered it can now encrypt its identification information q using the key given by the RA. The drone only encrypts its identification information when making an authentication request. If a request is needed the following will occur. The drone's identification information q is sent into a similar split function as used in the RA to create two row vectors q' and q''. The following will occur inside the split function which is shown below in the following pseudo code.

```
for k < size of S, k++
    if (S[k] == 0) //different from RA
        q'[k]= q[k]
        q''[k] = q[k]
    else if (S[k] == 1) //different from RA
```

$q'[k]$ = random number (0.01 and 1)

$q''[k] = q[k] - q'[k]$

Next, the two row vectors are transposed into column vectors q'^T and q''^T. Then, the four column indices of size n are created using matric multiplication, as shown in the equations below.

$$T_1 = (n_1{}^{-1} \; x \; m_i') \; x \;\; q'^T$$

$$T_2 = (n_2{}^{-1} \; x \; m_i'') \; x \; q'^T$$

$$T_3 = (n_3{}^{-1} \; x \; m_i''') \; x \; q''^T$$

$$T_4 = (n_4{}^{-1} \; x \; m_i'''') \; x \; q''^T$$

Finally, the four indices are combined in to one index E(q) = [T_1, T_2, T_3, T_4]. Then the encrypted index E(q) is sent to the authentication server for verification.

Authentication Process and results

In this step the authentication server will receive an authentication request from the remote drone over the network. The authentication server has previously stored all encrypted identification information from the RA. The authentication server will perform dot product multiplication on all saved indices E(p_i), with that of the drone's encrypted authentication request E(q). The server does this until a match is found, and then sends the results through the network. The server will either accept or reject the request depending on if there was a match in the dot product of the two indices. If accepted the server will allow the remote drone to send its information to the server. If it is rejected, the server will send a message saying this drone is not authenticated and does not accept any of its data.

CHAPTER V – METHODOLOGY

Overview of Experiments

Vivado HLS takes two C/C++ files for the source and testbench, these are the main code and top-level function code that implements the kNN encryption and the Dot-Product Index Search. Three separate instances of Vivado HLS are implemented in my design, the Drone Encryption, the Registration Authority Encryption, and the Dot-Product Index Search. The top function is the Matrix multiplication function for the kNN encryption and the dot-product function for the search portion of the scheme. This means that those functions are the top-level functions of the FPGA device. The source is where the other functions and main function of the code is located.

Software

This is the software I used to implement my experiments and performed extensive testing.

MatLAB

MatLAB is a platform for numeric computing and programming that is used to analyze data, develop several types of algorithms, and create models within the software. The kNN algorithm used in my scheme was first implemented in MatLAB and developed by Ahmed Sheriff. I used this software to crosscheck my work to verify that it produced the same results that I did. I was tasked with taking this code an converting it into C/C++ and then implementing it in my privacy-aware and hardware-based accelerated authentication scheme.

Visual Studios

Visual studio is an integrated development environment (IDE) used in software development. I used this as an intermediate before implementing my code in Vivado HLS. This is the first IDE in which I was able to verify my C/C++ results to that of the MatLAB implementation of the kNN algorithm. I was able to perform my code in this IDE before moving on to the next step.

Vivado HLS

Vivado HLS is a high-level synthesis design tool that allows C and C++ code to be converted into Verilog and VHDL and then implemented onto an FPGA as hardwired code. This software will automatically accelerate parts of the code to further lower the computational cost through parallelism. Vivado HLS also allows the user to simulate results through synthesis, co-synthesis, and implementation. This is where timing, latency, utilization can be outputted for many different FPGA devices. Further, files can be created for Vivado Design suite to acquire even more data on the implementation of different designs.

Vivado Design Suite

This is a design suite that was created by Xilinx for the synthesis, implementation, and analysis of HDL designs. This software allows many functions and analysis such as power usage, utilizations, timing, and design implementations and schematics. I used this software to produce theoretical results of how much power my authentication scheme consumes, different timings results, as well as the different resources used in each instance of the scheme. Instances such as increasing the ID size also increase the number of resources used which is reflected in my results.

Experiment Methodology

I tested the kNN algorithm encryption and search function simulating two different FPGA devices, the Kintex 116 UltraScale+ and the Nexys A7-100T. These two devices are on opposite ends of each other, meaning that the cost and available resources of the Kintex far outweighs that of the Nexys. This is shown in the results sections of each device utilization under the row of available resources. The cost of each board is $2,995 for the Kintex and $229.99 for the Nexys. A brief description of the resources used in my experiments is shown below:

- BRAM_18K: This is a dual-port random access memory module that is on the FPGA board. It is used to store large sets of data and each module can store 18 kilobits of data. The dual-port is important because this allows for parallel same-clock-cycle access to these various locations in memory. (Xilinx documentation)
- DSP48E: This is an arithmetic logic unit that is embedded into the FPGA board. It is composed of an Add/Subtract unit connected to a multiplier that is connected to a final add/subtract/accumulate engine. Most importantly this allows a single unit to perform the following function A += X*Y, which is a function used allow in this scheme.
- FF: This a flip flop that includes data input, clock input, clock enable, reset and data output.
- LUT: Look up tables that are connected to the FF. This is a table that generates the output based on the inputs received.

The first steps in creating this experiment are too open Vivado HLS and choose which FPGA device will be used during experimentation. Then the C/C++ files can be imported or created into the design. Then a top function is chosen from the source file. Lastly, a solution folder is created, this is where all csim, implementation and synthesis information are stored. Csim is used to test the results of the experiment and that the correct values are shown, C Synthesis starts the source code, C/RTL Cosimulation verifies the RTL output and finally Export RTL packages the RTL into an IP output that can be opened in Vivado Design Suite.

Synthesis and Export RTL

Once the correct files and parameters have been setup in Vivado HLS the testing phase can begin. This is where I can acquire information such as Latency, Number of Cycles, Utilization, Power consumption and much more. The following identification information sizes were tested in this authentication scheme.

- ID array of 4 elements type double
    - Creates an encrypted index of 16 elements type double
        - Size in bits: 4,096 bits
- ID array of 32 elements type double
    - Creates an encrypted index of 128 elements type double
        - Size in bits: 8,192 bits
- ID array of 64 elements type double
    - Creates an encrypted index of 256 elements type double
        - Size in bits: 16,384 bits
- ID array of 128 elements type double

    - Creates an encrypted index of 512 elements type double
        - Size in bits: 32,768 bits
- ID array of 256 elements type double
    - Creates an encrypted index of 1024 elements type double
        - Size in bits: 65,536 bits
- ID array of 512 elements type double
    - Creates an encrypted index of 2048 elements type double
        - Size in bits: 131,072 bits

Using any larger ID sizes risk using up all the memory in the lower-end FPGA devices, the Nexys can only achieve index encryption sizes slightly above 256 before it can no longer store anything in block ram memory. In the results section, most graphs have encrypted index size in kilobits, this means the size of the experiment I was performing at that time while either creating an encrypted index of that size or searching through encrypted indices of that size. Vivado HLS allows the user to test multiple solutions in the same project, in this case, each size change of the identification information is a new solution with its own simulation and implementation reports. This also applies to when testing is done on a different FPGA device. The following occurs when running this experiment.

1) CSIM: if you have printed statements in the code to verify results it will print them in a console.
2) C Synthesis: This gets the source code started in Vivado HLS and will produce reports for utilization, latency, and more. This is shown in the results section of the report.

3) Next, Cosimulation is ran, this verifies the RTL output of the code.
4) Lastly, the Export RTL is in setup only and for VHDL. This will create an IP output file labeled project. This file can be opened in Vivado Design suite and is where the power consumption can be obtained. This will also allow for design schematics and show visual representations of the resources used on the board.

The latency that is reported is the total amount time that it takes to perform the kNN encryption of the index or the dot product search. It is measured in unites of time and is calculated by the number of clock cycles it takes for an operation to perform. Where there is no data dependency, I apply loop unrolling using the #pragma HLS UNROLL. This will allow for parallelism in functions that are not dependent on other variables. There is a noticeable difference in speed if not using this #pragma in the experiments.

Implementation System

The system that I implemented my work on is a i9-9900k @ 3.60 GHz with 8 cores and 16 threads total. It has 32 GB of 3200 MHz of ram and a NVIDIA GTX 1080 Graphics card that supports 2560 CUDA cores. Lastly, the main implementation software; Vivado HLS, Vivado Design Suite, MatLAB and Visual Studios is installed on a M.2 Samsung 970 Pro 1TB with read/write speeds up to 3500/2700 MB/s.

Results of Kintex 116

Table 5.1 Kintex 116 Encryption of Index Utilization 1.024kBits

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	0	0	0	0	0
Expression	0	0	0	133	0
FIFO	0	0	0	0	0
Instance	0	25	1043	1188	0
Memory	4	0	0	0	0
Multiplexer	0	0	0	565	0
Register	0	0	680	0	0
Total	4	25	1723	1886	0
Available	960	1824	433920	216960	64
Utilization (%)	0.42	1.37	0.40	0.87	0

Table 5.2 Kintex 116 Search of Index Utilization 1.024kBits

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	0	0	0	0	0
Expression	0	0	0	355	0
FIFO	0	0	0	0	0
Instance	0	14	744	985	0
Memory	4	0	0	0	0
Multiplexer	0	0	0	77	0
Register	0	0	217	0	0
Total	4	14	961	1417	0
Available	960	1824	433920	216960	64
Utilization (%)	0.417	0.768	0.221	0.653	0.000

Tables 5.1 and 5.2 represent the kNN encryption process and index dot-product search results of an encrypted index size, 16 elements of type double using the Kintex 116. As shown in both tables, the devices resources are barely used.

Table 5.3 Kintex 116 Encryption of Index Utilization 8.194 kBits

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	0	0	0	0	0
Expression	0	0	0	1049	0
FIFO	0	0	0	0	0
Instance	0	25	1043	1188	0
Memory	4	0	0	0	0
Multiplexer	0	0	0	2757	0
Register	0	0	3864	0	0
Total	4	25	4907	4994	0
Available	960	1824	433920	216960	64
Utilization (%)	0.42	1.37	1.13	2.30	0

Table 5.4 Kintex 116 Search of Index Utilization 8.194 kBits

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	0	0	0	0	0
Expression	0	0	0	357	0
FIFO	0	0	0	0	0
Instance	0	14	744	985	0
Memory	4	0	0	0	0
Multiplexer	0	0	0	77	0
Register	0	0	223	0	0
Total	4	14	967	1419	0
Available	960	1824	433920	216960	64
Utilization (%)	0.417	0.768	0.223	0.654	0.000

Tables 5.3 and 5.4 represent the kNN encryption process and index dot-product search results of an encrypted index size, 128 elements of type double using the Kintex 116. As shown in both tables, the devices resources have slightly increased.

Table 5.5 Kintex 116 Encryption of Index Utilization 16.384 kbits

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	0	0	0	0	0
Expression	0	0	0	2327	0
FIFO	0	0	0	0	0
Instance	0	25	1043	1188	0
Memory	15	0	0	0	0
Multiplexer	0	0	0	4209	0
Register	0	0	6850	0	0
Total	15	25	7893	7724	0
Available	960	1824	433920	216960	64
Utilization (%)	1.56	1.37	1.82	3.56	0

Table 5.6 Kintex 116 Search of Index Utilization 16.384 kbits

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	0	0	0	0	0
Expression	0	0	0	358	0
FIFO	0	0	0	0	0
Instance	0	14	744	985	0
Memory	4	0	0	0	0
Multiplexer	0	0	0	77	0
Register	0	0	225	0	0
Total	4	14	969	1420	0
Available	960	1824	433920	216960	64
Utilization (%)	0.417	0.768	0.223	0.654	0.000

Tables 5.5 and 5.6 represent the kNN encryption process and index dot-product search results of an encrypted index size, 256 elements of type double using the Kintex 116. As shown in the tables, the devices resources are still only slightly increasing for the kNN encryption while the dot-product index search remains nearly the same.

Table 5.7 Kintex 116 Encryption of Index Utilization 32.768 kbits

DSP	0	0	0	0	0
Expression	0	0	0	5213	0
FIFO	0	0	0	0	0
Instance	0	25	1043	1188	0
Memory	57	0	0	0	0
Multiplexer	0	0	0	6030	0
Register	0	0	13514	0	0
Total	57	25	14557	12431	0
Available	960	1824	433920	216960	64
Utilization (%)	5.94	1.37	3.35	5.73	0

Table 5.8 Kintex 116 Search of Index Utilization 32.768 kbits

Name	**BRAM_18K**	**DSP48E**	**FF**	**LUT**	**URAM**
DSP	0	0	0	0	0
Expression	0	0	0	359	0
FIFO	0	0	0	0	0
Instance	0	14	744	985	0
Memory	4	0	0	0	0
Multiplexer	0	0	0	77	0
Register	0	0	227	0	0
Total	4	14	971	1421	0
Available	960	1824	433920	216960	64
Utilization (%)	0.417	0.768	0.224	0.655	0.000

Tables 5.7 and 5.8 represent the kNN encryption process and index dot-product search results of an encrypted index size, 512 elements of type double using the Kintex 116. As shown in both tables, the devices resources are steadily increasing for the kNN encryption process while the dot-product index search barely increases.

Table 5.9 Kintex 116 Encryption of Index Utilization 65.536 kbits

Name	**BRAM_18K**	**DSP48E**	**FF**	**LUT**	**URAM**
DSP	0	0	0	0	0
Expression	0	0	0	11615	0
FIFO	0	0	0	0	0
Instance	0	25	980	1130	0
Memory	228	0	0	0	0
Multiplexer	0	0	0	9774	0
Register	0	0	27037	0	0
Total	228	25	28017	22519	0
Available	960	1824	433920	216960	64
Utilization (%)	23.75	1.37	6.46	10.38	0

Table 5.10 Kintex 116 Search of Index Utilization 65.536 kbits

DSP	0	0	0	0	0
Expression	0	0	0	360	0
FIFO	0	0	0	0	0
Instance	0	14	744	985	0
Memory	4	0	0	0	0
Multiplexer	0	0	0	77	0
Register	0	0	229	0	0
Total	4	14	973	1422	0
Available	960	1824	433920	216960	64
Utilization (%)	0.417	0.768	0.224	0.655	0.000

Tables 5.9 and 5.10 represent the kNN encryption process and index dot-product search results of an encrypted index size, 1024 elements of type double using the Kintex 116. As shown in both tables, the kNN encryption process utilization is increasing while the dot-product index search resources are only slightly increasing.

Table 5.11 Kintex 116 Encryption of Index Utilization 131.072 kbits

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	0	0	0	0	0
Expression	0	0	0	25697	0
FIFO	0	0	0	0	0
Instance	0	25	980	1130	0
Memory	912	0	0	0	0
Multiplexer	0	0	0	16758	0
Register	0	0	53650	0	0
Total	912	25	54630	43585	0
Available	960	1824	433920	216960	64
Utilization (%)	95.00	1.37	12.59	20.09	0

Table 5.12 Kintex 116 Search of Index Utilization 131.072 kbits

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	0	0	0	0	0
Expression	0	0	0	361	0
FIFO	0	0	0	0	0
Instance	0	14	744	985	0
Memory	4	0	0	0	0
Multiplexer	0	0	0	77	0
Register	0	0	231	0	0
Total	4	14	975	1423	0
Available	960	1824	433920	216960	64
Utilization (%)	0.417	0.768	0.225	0.656	0.000

Tables 5.11 and 5.12 represent the kNN encryption process and index dot-product search results of an encrypted index size, 2048 elements of type double using the Kintex 116. As shown in both tables, the kNN encryption process utilization is reaching its max resources for block ram while the dot-product index search resources are only slightly increasing.

Table 5.13 Kintex 116 Encryption of Index Latency and Power Consumption

size (kbits)	Latency (s)	Power Consumption(W)
1.024	0.00000547	0.053
8.167	0.001764	0.07
16.384	0.013604	0.084
32.768	0.107	0.13
65.536	0.754	0.223
131.072	6.21	0.42

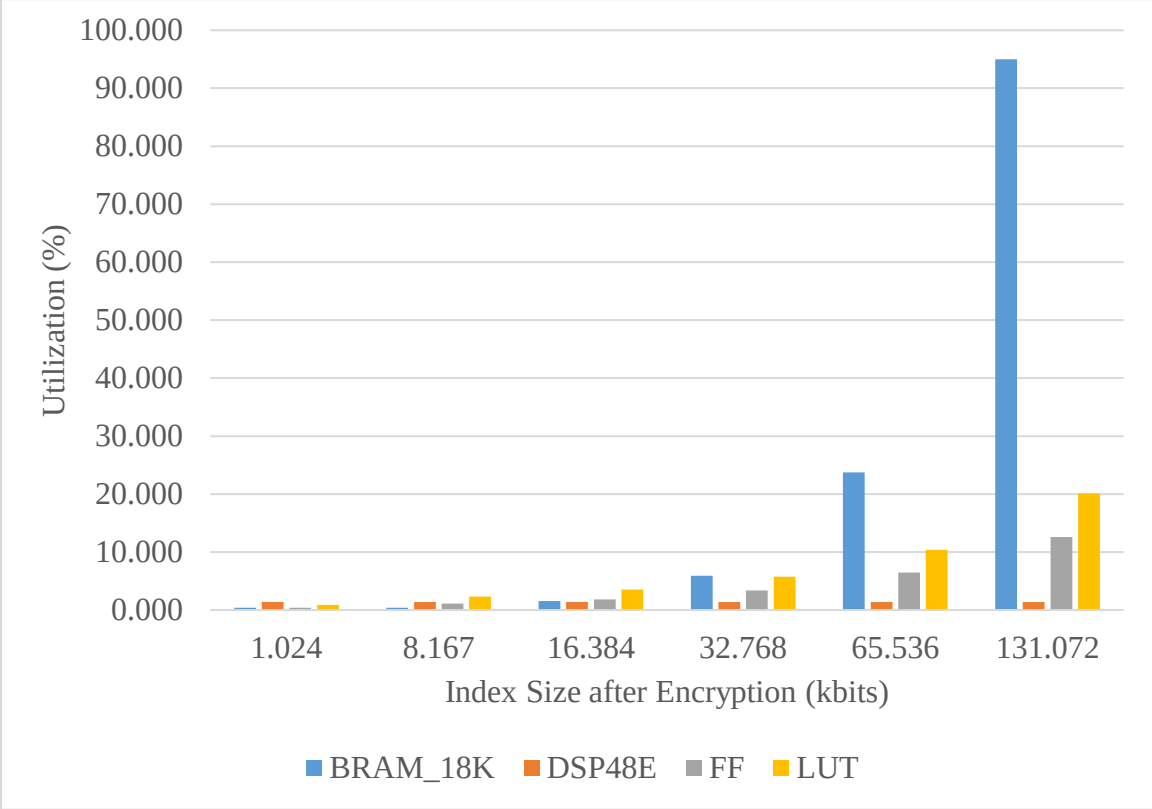


Figure 5.1 Kintex 116 Utilization Comparison of Encryption of Index

Table 5.13 is composed of the total latency calculations and power consumptions for all encrypted index sizes that were tested. Figure 5.1 represents the total utilization resources that were consumed for each encrypted index size during the kNN encryption process.

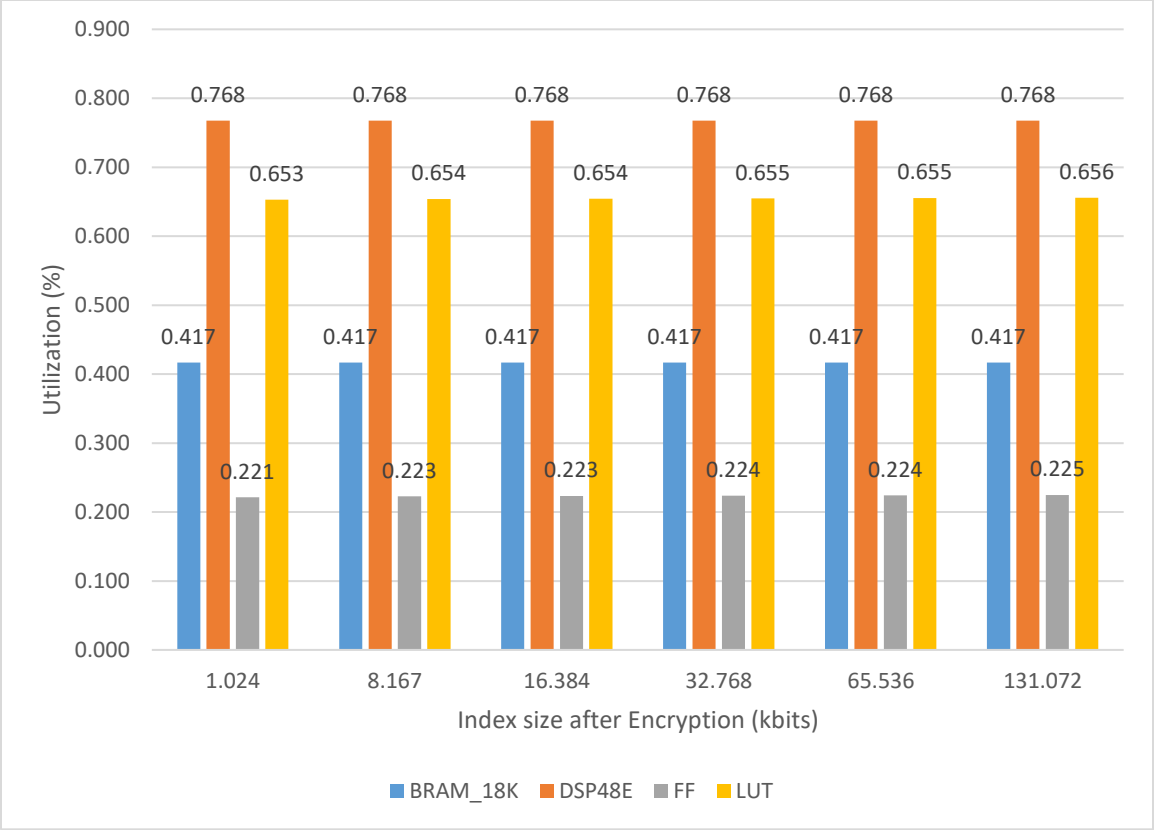


Figure 5.2 Kintex 116 Utilization Comparison for Encrypted Index Search

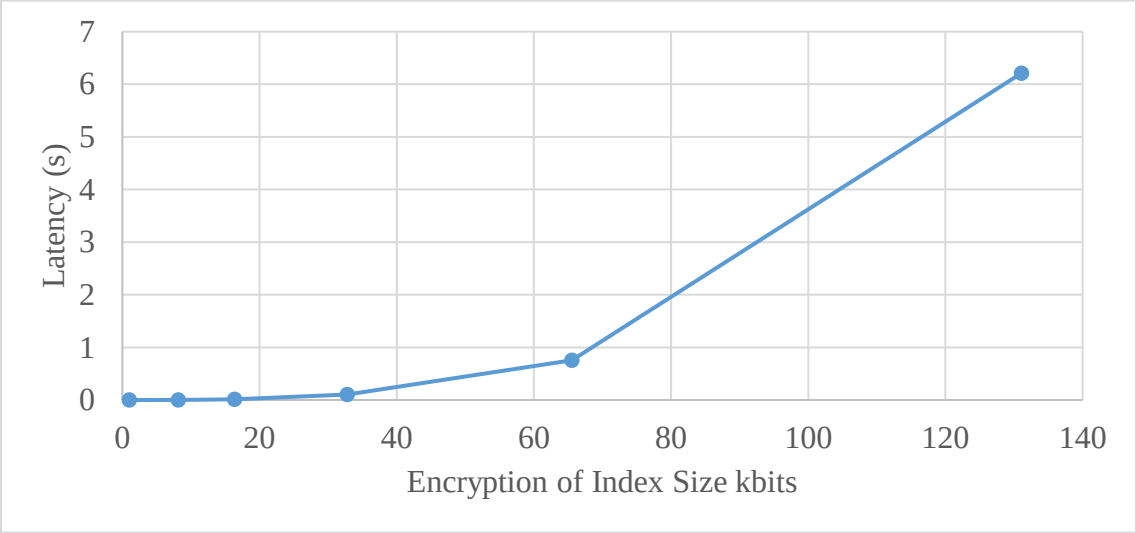


Figure 5.3 Kintex 116 Encryption of Index Latency graph

Figures 5.2 and 5.3 represent the total utilization of the Dot-Product index search and the representation of the latency as the encrypted index sizes are increased.

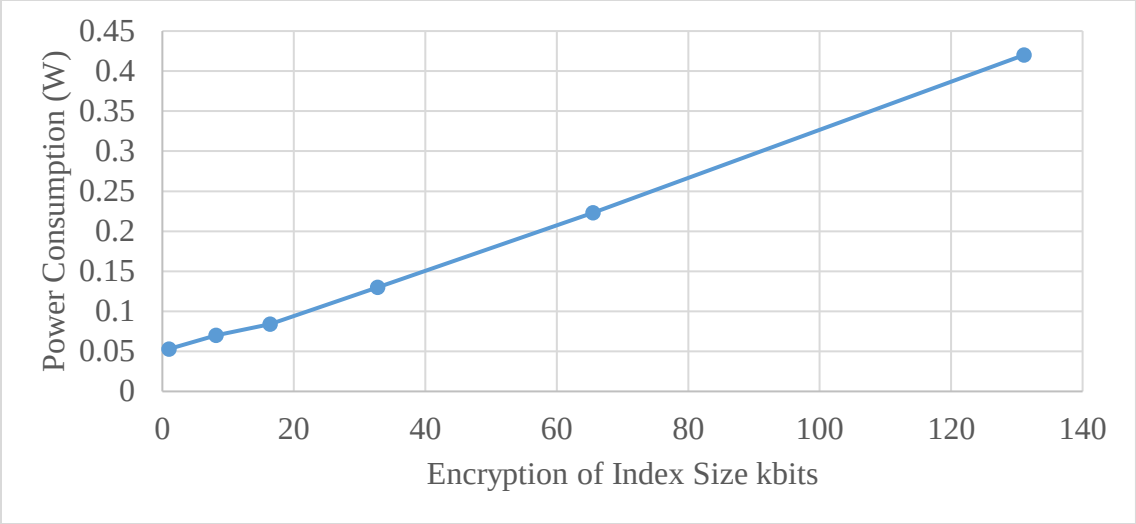


Figure 5.4 Kintex 116 Encryption of Index Power Consumption

Table 5.14 Kintex 116 Dot-Product Index Search Latency and Power Consumption

size (kbits)	**Latency (s)**	**Power Consumption (W)**
1.024	1.78E-6	0.028
8.167	14.10E-6	0.028
16.384	28.18E-6	0.028
32.768	56.34E-6	0.028
65.536	113.00E-6	0.028
131.072	225.00E-6	0.028

Figures 5.4 represent the power consumption of the kNN encryption process as the encrypted index size is increased. Table 5.14 represents the total latency's calculated and power consumptions calculations during the dot-product index search as the encrypted index size is increased. Finally, Figure 5.5 represents the Kintex 116 how the latency is increased as the encrypted index sizes are increased.

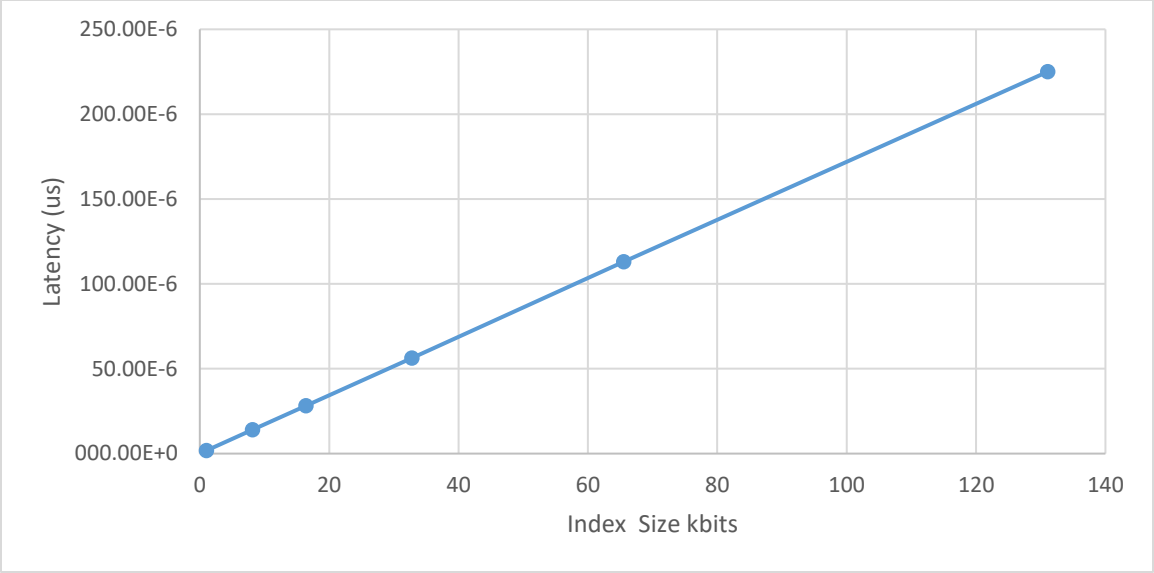


Figure 5.5 Kintex 116 Search of Index Latency

Results of Nexys A7-100T

Table 5.15 Nexys A7-100T Encryption of Index Utilization 1.024 kbits

Name	BRAM_18K	DSP48E	FF	LUT
DSP	0	0	0	0
Expression	0	0	0	137
FIFO	0	0	0	0
Instance	0	25	1143	1225
Memory	4	0	0	0
Multiplexer	0	0	0	646
Register	0	0	1078	0
Total	4	25	2221	2008
Available	270	240	126800	63400
Utilization (%)	1.48	10.42	1.75	3.17

Table 5.16 Nexys A7-100T Search of Index Utilization 1.024 kbits

Name	BRAM_18K	DSP48E	FF	LUT
DSP	0	0	0	0
Expression	0	0	0	356
FIFO	0	0	0	0
Instance	0	14	826	1021
Memory	4	0	0	0
Multiplexer	0	0	0	107
Register	0	0	454	0
Total	4	14	1280	1484
Available	270	240	126800	63400
Utilization (%)	1.481	5.833	1.009	2.341

Tables 5.15 and 5.16 represent the total utilization of resources for the Nexys A7-100T during the kNN encryption process and the Dot-Product index search. The encrypted index size is 16 elements of type double.

Table 5.17 Nexys A7-100T Encryption of Index Utilization 8.192 kbits

Name	**BRAM_18K**	**DSP48E**	**FF**	**LUT**
DSP	0	0	0	0
Expression	0	0	0	1049
FIFO	0	0	0	0
Instance	0	25	1143	1225
Memory	4	0	0	0
Multiplexer	0	0	0	3098
Register	0	0	4254	0
Total	4	25	5397	5372
Available	270	240	126800	63400
Utilization (%)	1.48	10.42	4.26	8.47

Table 5.18 Nexys A7-100T Search of Index Utilization 8.192 kbits

Name	**BRAM_18K**	**DSP48E**	**FF**	**LUT**
DSP	0	0	0	0
Expression	0	0	0	361
FIFO	0	0	0	0
Instance	0	14	826	1021
Memory	4	0	0	0
Multiplexer	0	0	0	107
Register	0	0	460	0
Total	4	14	1286	1489
Available	270	240	126800	63400
Utilization (%)	1.481	5.833	1.014	2.349

Tables 5.17 and 5.18 represent the total utilization of resources for the Nexys A7-100T during the kNN encryption process and the Dot-Product index search. The encrypted index size is 128 elements of type double. As shown above, the Dot-Product index search slightly increases while the kNN encryption process utilization is steadily increasing.

Table 5.19 Nexys A7-100T Encryption of Index Utilization 16.384 kbits

Name	**BRAM_18K**	**DSP48E**	**FF**	**LUT**
DSP	0	0	0	0
Expression	0	0	0	2327
FIFO	0	0	0	0
Instance	0	25	1143	1225
Memory	16	0	0	0
Multiplexer	0	0	0	4278
Register	0	0	7560	0
Total	16	25	8703	7830
Available	270	240	126800	63400
Utilization (%)	5.93	10.42	6.86	12.35

Table 5.20 Nexys A7-100T Search of Index Utilization 16.384 kbits

Name	**BRAM_18K**	**DSP48E**	**FF**	**LUT**
DSP	0	0	0	0
Expression	0	0	0	362
FIFO	0	0	0	0
Instance	0	14	826	1021
Memory	4	0	0	0
Multiplexer	0	0	0	107
Register	0	0	462	0
Total	4	14	1288	1490
Available	270	240	126800	63400
Utilization (%)	1.481	5.833	1.016	2.350

Tables 5.19 and 5.20 represent the total utilization of resources for the Nexys A7-100T during the kNN encryption process and the Dot-Product index search. The encrypted index size is 256 elements of type double. As shown above, the Dot-Product index search slightly increases while the kNN encryption process utilization is steadily increasing.

Table 5.21 Nexys A7-100T Encryption of Index Utilization 32.768 kbits

Name	BRAM_18K	DSP48E	FF	LUT
DSP	0	0	0	0
Expression	0	0	0	5213
FIFO	0	0	0	0
Instance	0	25	1143	1225
Memory	64	0	0	0
Multiplexer	0	0	0	6103
Register	0	0	14430	0
Total	64	25	15573	12541
Available	270	240	126800	63400
Utilization (%)	23.70	10.42	12.28	19.78

Table 5.22 Nexys A7-100T Search of Index Utilization 32.768 kbits

Name	BRAM_18K	DSP48E	FF	LUT
DSP	0	0	0	0
Expression	0	0	0	363
FIFO	0	0	0	0
Instance	0	14	826	1021
Memory	4	0	0	0
Multiplexer	0	0	0	107
Register	0	0	464	0
Total	4	14	1290	1491
Available	270	240	126800	63400
Utilization (%)	1.481	5.833	1.017	2.352

Tables 5.21 and 5.22 represent the total utilization of resources for the Nexys A7-100T during the kNN encryption process and the Dot-Product index search. The encrypted index size is 512 elements of type double. As shown above, the Dot-Product index search slightly increases while the kNN encryption process utilization is steadily increasing.

Table 5.23 Nexys A7-100T Encryption of Index Utilization 65.536 kbits

Name	BRAM_18K	DSP48E	FF	LUT
DSP	0	0	0	0
Expression	0	0	0	11615
FIFO	0	0	0	0
Instance	0	25	1143	1225
Memory	256	0	0	0
Multiplexer	0	0	0	9798
Register	0	0	28056	0
Total	256	25	29199	22638
Available	270	240	126800	63400
Utilization (%)	94.81	10.42	23.03	35.71

Table 5.24 Nexys A7-100T Search of Index Utilization 65.536 kbits

Name	BRAM_18K	DSP48E	FF	LUT
DSP	0	0	0	0
Expression	0	0	0	364
FIFO	0	0	0	0
Instance	0	14	826	1021
Memory	4	0	0	0
Multiplexer	0	0	0	107
Register	0	0	466	0
Total	4	14	1292	1492
Available	270	240	126800	63400
Utilization (%)	1.481	5.833	1.019	2.353

Tables 5.23 and 5.24 represent the total utilization of resources for the Nexys A7-100T during the kNN encryption process and the Dot-Product index search. The encrypted index size is 1024 elements of type double. As shown above, the Dot-Product index search slightly increases while the kNN encryption process utilization reaching its max resources available in the block ram. The Nexys is only able to encrypt indices sizes 256 elements of type double and below.

Table 5.25 Nexys A7-100T Encryption of Index Latency and Power Consumption

size (kbits)	Latency (s)	Power Consumption(W)
1.024	0.00000687	0.077
8.167	0.002141	0.103
16.384	0.016621	0.15
32.768	0.133	0.337
65.536	1.097	0.337

Table 5.26 Nexys A7-100T Search of Index Latency and Power Consumption

size (kbits)	Latency (s)	Power Consumption (W)
1.024	2.27E-6	0.051
8.167	17.95E-6	0.053
16.384	35.87E-6	0.054
32.768	71.71E-6	0.054
65.536	143.00E-6	0.054

Tables 5.25 and 5.26 represent the total latency's calculated and power consumed during each instance of the kNN encryption process and Dot-Product index search from encrypted index sizes of 16 elements to 1024 elements.

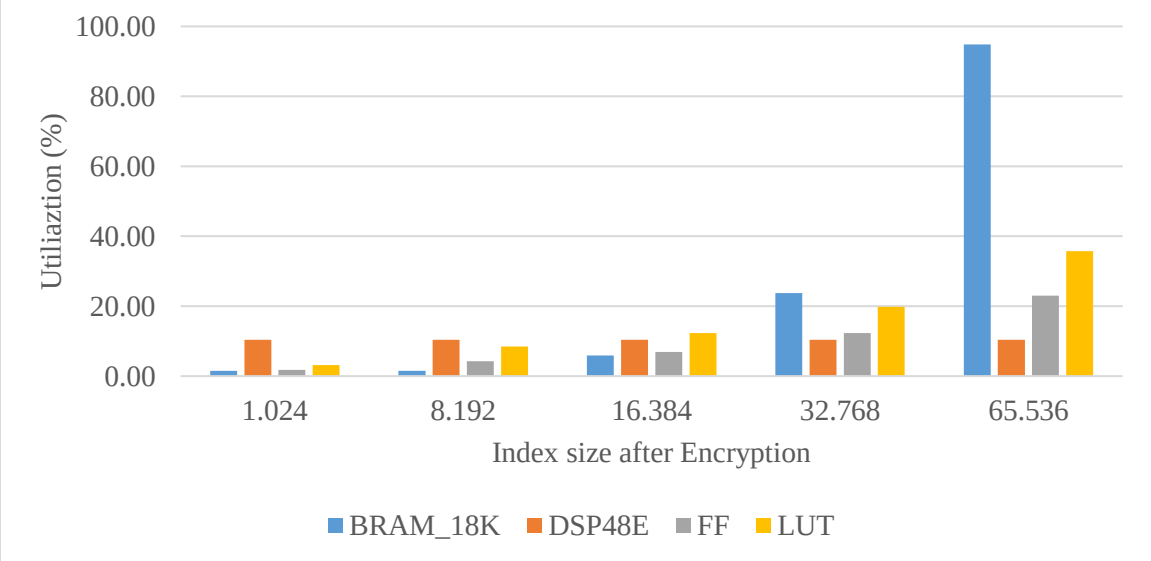


Figure 5.6 Nexys A7-100T Utilization Comparison of Encryption of Index

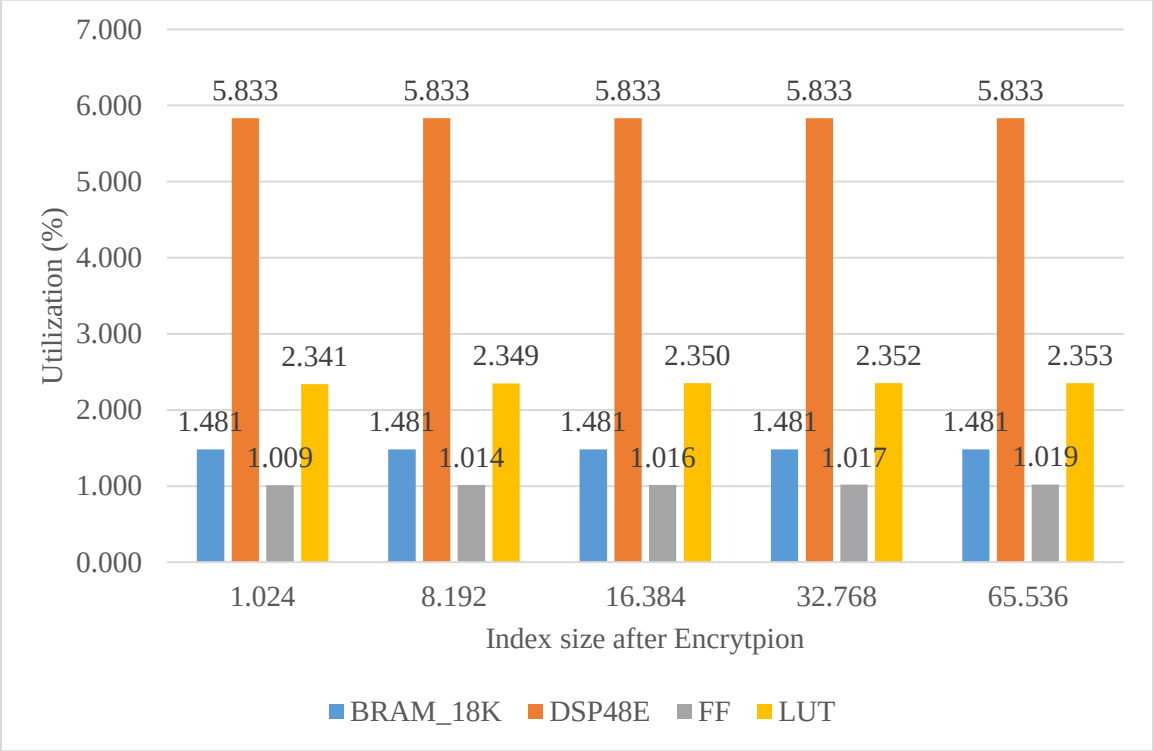


Figure 5.7 Nexys A7-100T Utilization Comparison of Index Search

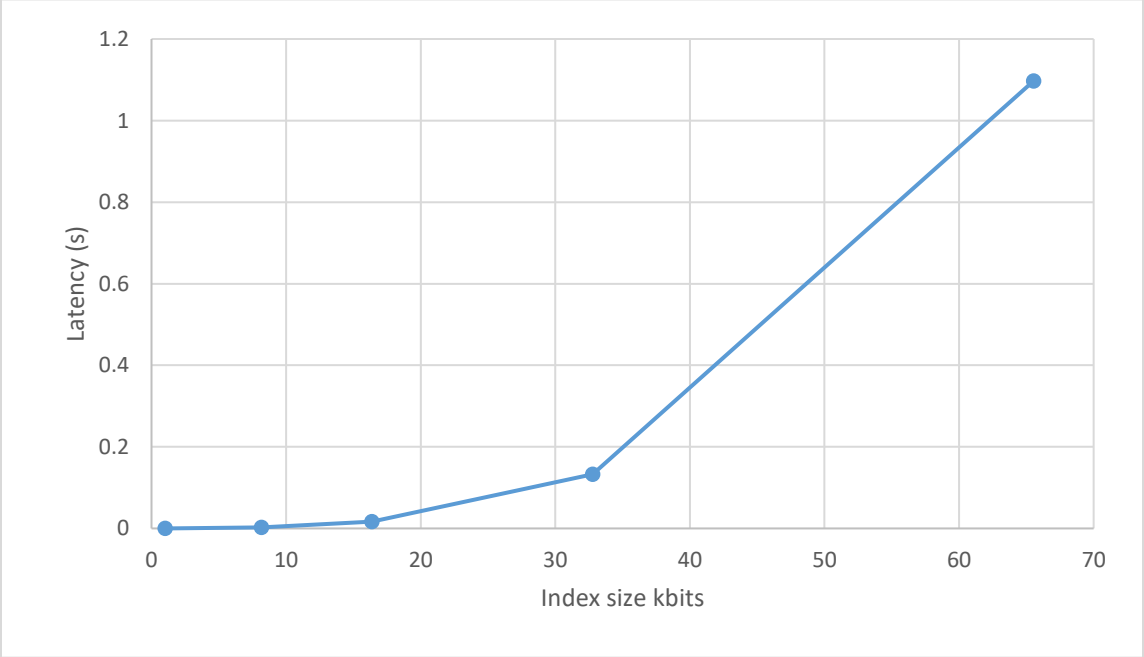


Figure 5.8 Nexys A7-100T Latency Comparison of Index Encryption

In figures 5.7 and 5.8, the first represents the resource utilization during the index search while the other represents the latency as the encrypted index size is increased.

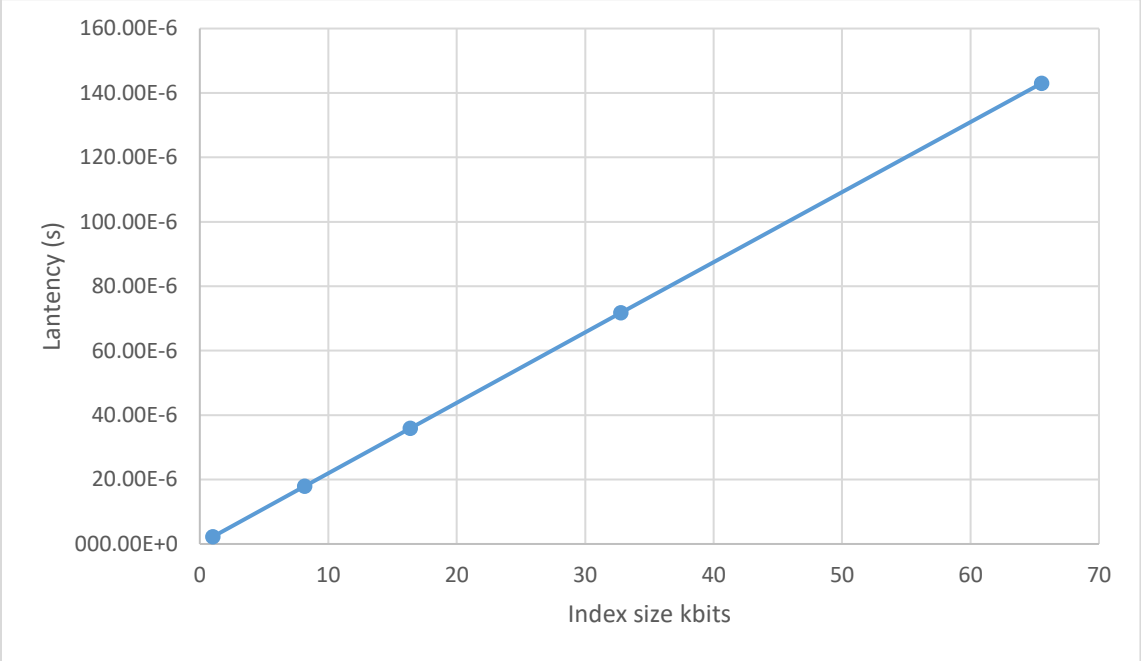


Figure 5.9 Nexys A7-100T Latency Comparison of Index Search

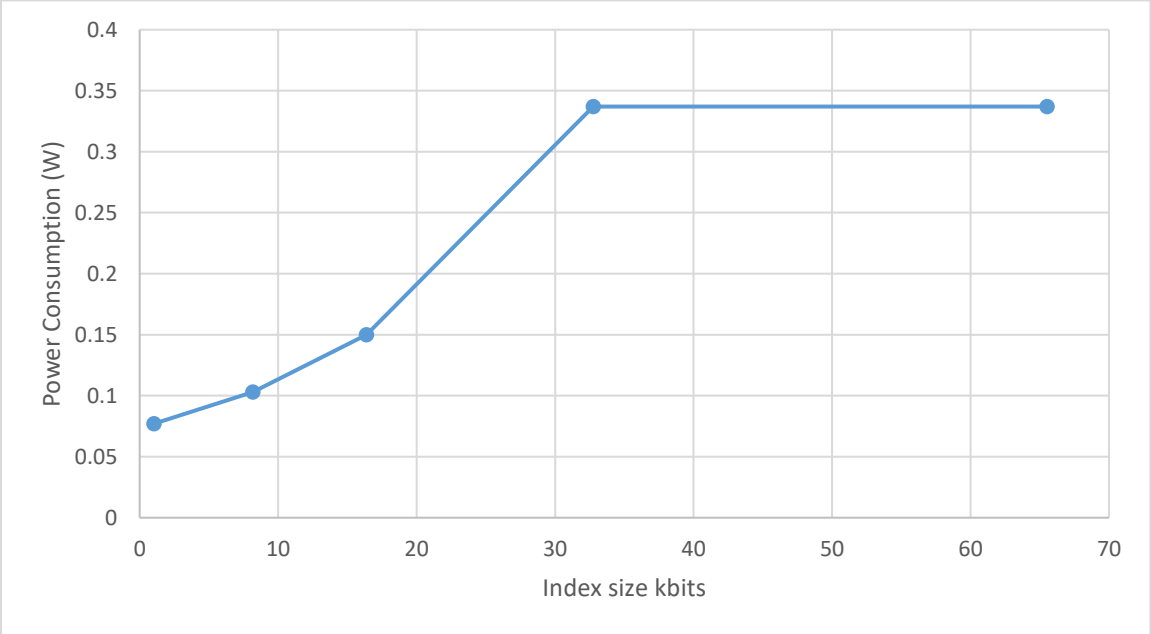


Figure 5.10 Nexys A7-100T Power Consumption Comparison of Index Encryption

Figures 5.9 and 5.10 represent the latency in the index search and power consumption in the index encryption process as the encrypted index size is increased from 16 elements to 1024 elements of type double.

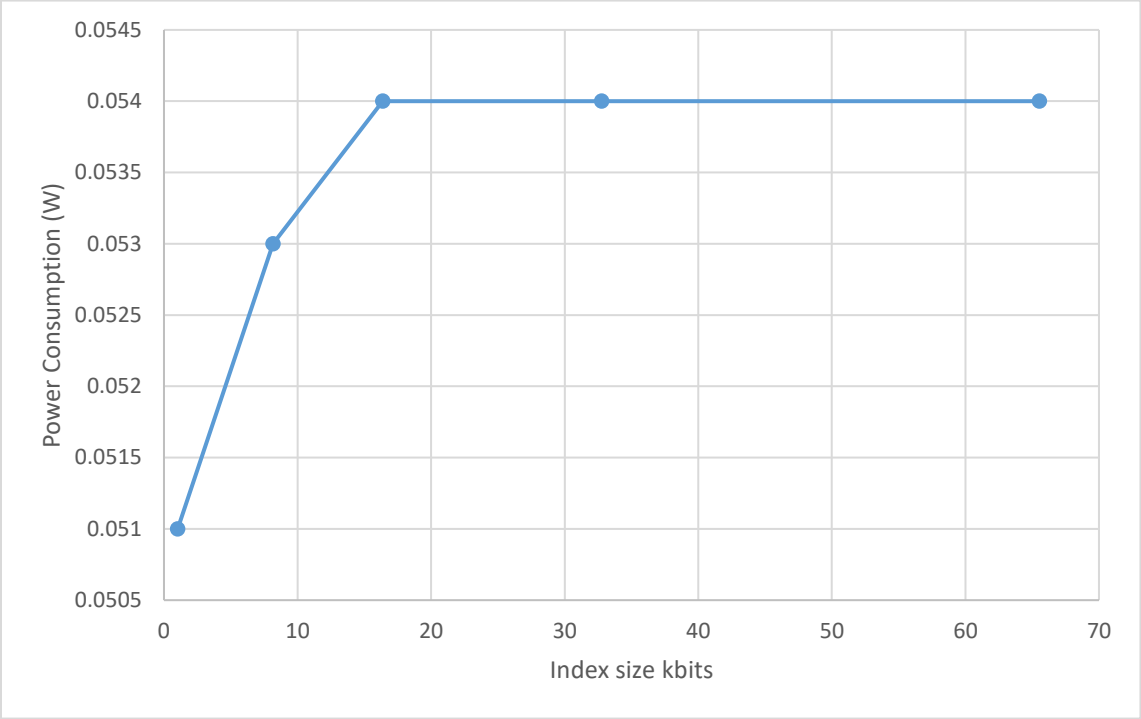


Figure 5.11 Nexys A7-100T Power Consumption Comparison of Index Search

Figure 5.11 represents the power consumption during the index search as the encrypted index size is increased.

Table 5.27 Execution time of index search comparisons of different IDEs (in seconds)

Size	Matlab	Visual Studios	Kintex	Nexys
1.024	0.000807	0.0020	0.00000178	0.00000227
8.167	0.032	0.0040	0.0000141	0.00001795
16.384	0.126	0.0070	0.00002818	0.00003587
32.768	0.591	0.0140	0.00005634	0.00007171
65.536	2.3	0.0250	0.000113	0.000143
131.072	9	0.0500	0.000225	N/A

Table 5.28 Execution time of index encryption IDE comparison (in seconds)

size	MatLAB	Visual Studios	Kintex	Nexys
1.024	0.000064	0.0002	0.00000547	0.00000687
8.167	0.000142	0.003	0.001764	0.002141
16.384	0.000424	0.008	0.013604	0.016621
32.768	0.00101	0.048	0.107	0.133
65.536	0.003753	0.377	0.754	1.097
131.072	0.0134	2.74	6.21	N/A

Tables 5.27 and 5.28 represent the latency's of both the kNN encryption process and the Dot-Product search implemented on MatLAB, Visual Studios, Kintex 116 FPGA, and the Nexys A7-100T FPGA.

CHAPTER VI – DISCUSSION AND CONCLUSION

Comparison of other works

My scheme is much faster than other similar authentication schemes for IoD. This is achieved by having a hardware component that offsets heavy computational tasks onto a specialized device, the FPGA. Where you need a GPUs or multiple cores to parallel certain functions the FPGA is naturally built for it because of its architecture.

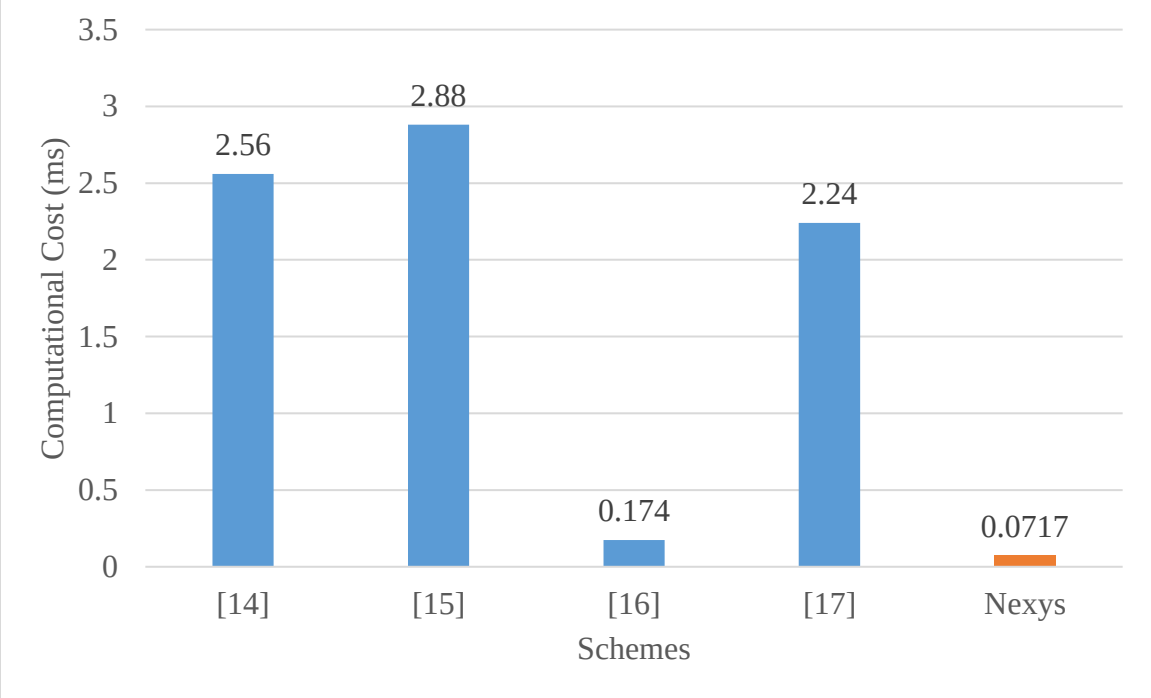


Figure 6.1 Proposed scheme (Nexys) comparison of results of server-side computational cost.

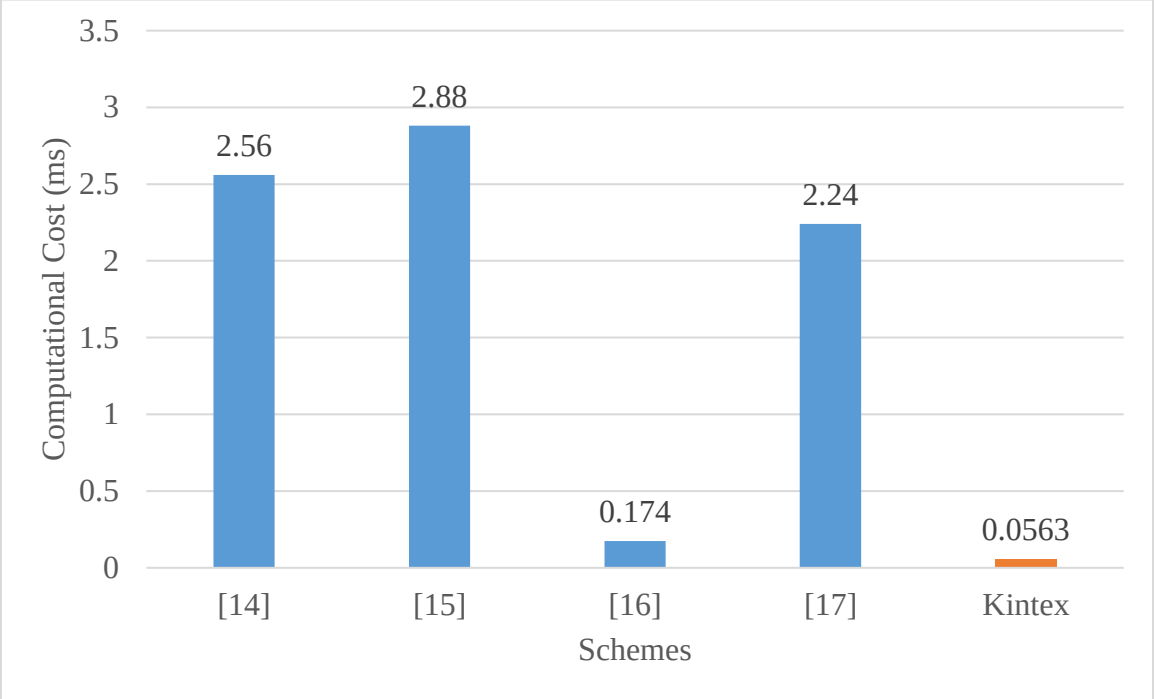


Figure 6.2 Proposed scheme (Kintex) comparison of results of server-side computational cost

I used the results from the Nexys A7-100T in figure 6.1 and Kintex 116 in figure 6.2 in the comparison of an encrypted identification size of 512 elements of type double. The dot product search was performed against 200 encrypted indices of the same size. The Kintex 116 performed even better, only taking 56us to do a similar task.

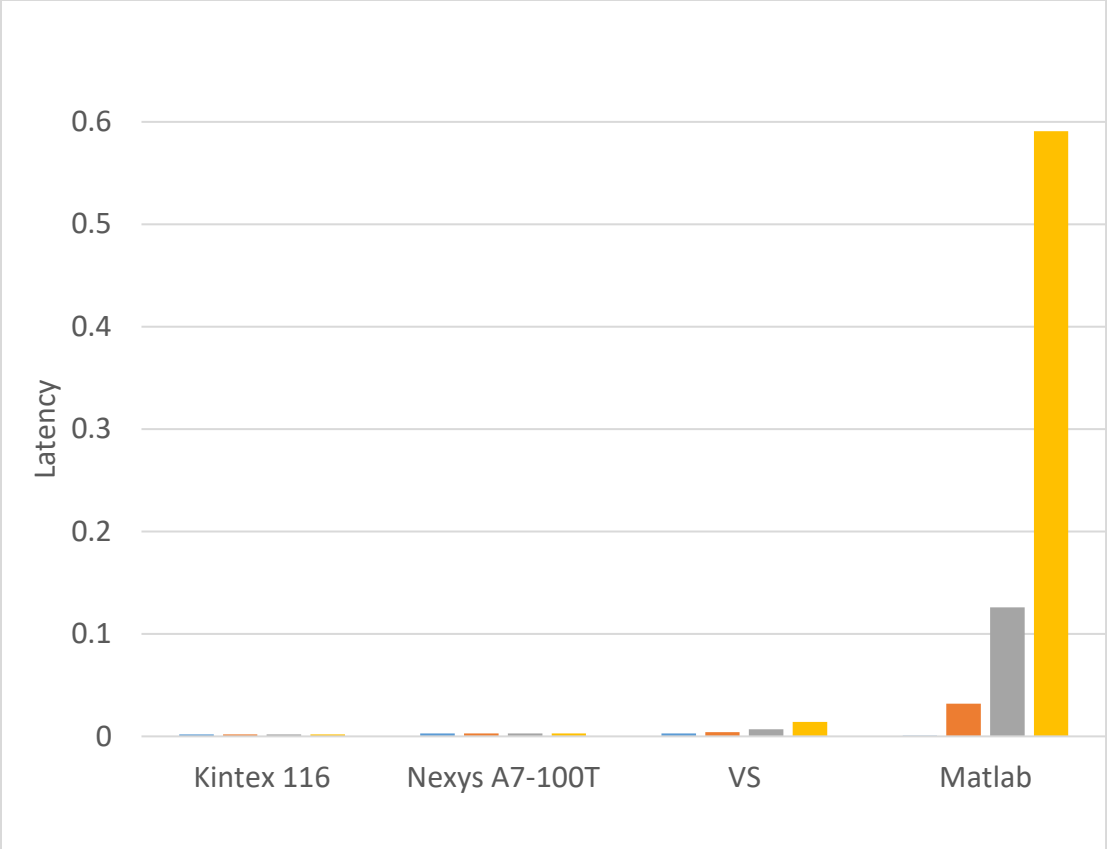


Figure 6.3 Implementation comparisons of dot-product Search

In Figure 6.3 is a compassion of the implementation of the dot product search on MatLAB, visual studios and the two FPGA devices using encrypted index sizes of 16, 128, 256, and 512. The 1024 and 2048 sizes were not included in this chart because the latency would cause the Kintex, Nexys, and Visual studio results to look like they were at zero. Also, 2048 was not used because the Nexys cannot encrypt indies that large. The chart shows that the FPGA devices outperformed Visual Studios and MatLAB and the Kintex 116 performed the best.

Drawbacks

Implementing the kNN algorithm on an FPGA device had some challenges and limitations from the software and hardware itself. The Vivado HLS software had some limitations such as no pointers to pointers, pointers to functions, no dynamic memory where everything had to be bounded, and no recursive functions etc. The csim function in Vivado had limitations when implementing large matrices within the design. However, the Vivado HLS could still synthesis and implement the kNN algorithm spite of this limitation.

Conclusion

I was able to simulate the different components of my proposed scheme using Vivado HLS and Vivado Design Suite. My results show that my scheme computational cost of the search performed much quicker than other similar schemes. The authentication scheme is flexible and efficient and can meet the needs of different consumers by choosing different FPGA devices to implement the Hardware Accelerated kNN Authentication Scheme. This is necessary for users that need cheaper means of security and do not need higher levels of encryption that require more FPGA resources and a more expensive device.

REFERENCES

Gharibi, Mirmojtaba, et al. “Internet of Drones.” *IEEE Access*, vol. 4, 2016, pp. 1148–1162., https://doi.org/10.1109/access.2016.2537208.

Tiberiu Paul Banu, et al. “The Use of Drones in Forestry.” *Journal of Environmental Science and Engineering B*, vol. 5, no. 11, 2016, https://doi.org/10.17265/2162-5263/2016.11.007.

Jung, Sunghun, and Kim Hyunsu. “Analysis of Amazon Prime Air UAV Delivery Service.” *Journal of Knowledge Information Technology and Systems*, vol. 12, no. 2, 2017, pp. 253–266., https://doi.org/10.34163/jkits.2017.12.2.005.

Wisel, Carlye. “How Verge Aero's Drones Pulled off an Election Day Win as Big as Biden's.” *TechCrunch*, TechCrunch, 24 Nov. 2020.

“Mexico Cartel Used Explosive Drones to Attack Police.” *BBC News*, BBC, 21 Apr. 2021.

“Field Programmable Gate Array (FPGA) History.” *HardwareBee*, 31 Dec. 2020.

Touger, Evan. “What Is an FPGA and Why Is It a Big Deal?” *Prowess Consulting*, 24 Sept. 2018.

“What Is Hardware Acceleration? Definition and Faqs.” *OmniSci*.

Peterson, Leif. “K-Nearest Neighbor.” *Scholarpedia*, vol. 4, no. 2, 2009, p. 1883., https://doi.org/10.4249/scholarpedia.1883.

“K Nearest Neighbor: Knn Algorithm: KNN in Python & R.” *Analytics Vidhya*, 18 Oct. 2020.

Gharibi, Mirmojtaba, et al. “Internet of Drones.” *IEEE Access*, vol. 4, 2016, pp. 1148–1162., https://doi.org/10.1109/access.2016.2537208.

“Secondary Navigation.” *Recreational Flyers & Modeler Community-Based Organizations*, 2 Sept. 2021, https://www.faa.gov/uas/recreational_fliers/.

“Bram and Other Memories.” *Xilinx*, https://www.xilinx.com/html_docs/xilinx2017_4/sdaccel_doc/jbt1504034294480.html.

Dave, Nirav, et al. “Hardware Acceleration of Matrix Multiplication on a Xilinx FPGA.” *2007 5th IEEE/ACM International Conference on Formal Methods and Models for*

Codesign (MEMOCODE 2007), 2007, https://doi.org/10.1109/memcod.2007.371239.

Owaida, Muhsen, et al. "Lowering the Latency of Data Processing Pipelines through FPGA Based Hardware Acceleration." *Proceedings of the VLDB Endowment*, vol. 13, no. 1, 2019, pp. 71–85., https://doi.org/10.14778/3357377.3357383.

Stratakos, Ioannis, et al. "Hardware Acceleration of Image Registration Algorithm on FPGA-Based Systems on Chip." *Proceedings of the International Conference on Omni-Layer Intelligent Systems*, 2019, https://doi.org/10.1145/3312614.3312636.

Gielata, Artur, et al. "AES Hardware Implementation in FPGA for Algorithm Acceleration Purpose." *2008 International Conference on Signals and Electronic Systems*, 2008, https://doi.org/10.1109/icses.2008.4673377.

Ernst, M., et al. "FPGA Based Hardware Acceleration for Elliptic Curve Public Key Cryptosystems." *Journal of Systems and Software*, vol. 70, no. 3, 2004, pp. 299–313., https://doi.org/10.1016/s0164-1212(03)00075-x.

Wazid, Mohammad, et al. "Design and Analysis of Secure Lightweight Remote User Authentication and Key Agreement Scheme in Internet of Drones Deployment." *IEEE Internet of Things Journal*, vol. 6, no. 2, 2019, pp. 3572–3584., https://doi.org/10.1109/jiot.2018.2888821.

Srinivas, Jangirala, et al. "TCALAS: Temporal Credential-Based Anonymous Lightweight Authentication Scheme for Internet of Drones Environment." *IEEE Transactions on Vehicular Technology*, vol. 68, no. 7, 2019, pp. 6903–6916., https://doi.org/10.1109/tvt.2019.2911672.

Tanveer, Muhammad, et al. "Lake-Iod: Lightweight Authenticated Key Exchange Protocol for the Internet of Drone Environment." *IEEE Access*, vol. 8, 2020, pp. 155645–155659., https://doi.org/10.1109/access.2020.3019367.

Zhang, Yunru, et al. "A Lightweight Authentication and Key Agreement Scheme for Internet of Drones." *Computer Communications*, vol. 154, 2020, pp. 455–464., https://doi.org/10.1016/j.comcom.2020.02.067.