RFID++: Testing existing security of RFID cards for next-gen authentication systems.

A PROJECT REPORT

Submitted by

Saket Upadhyay (18BCY10079)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING (WITH SPECIALISATION IN CYBERSECURITY AND DIGITAL FORENSICS)

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

VIT BHOPAL UNIVERSITY

KOTHRIKALAN, SEHORE

MADHYA PRADESH - 466114

MAY 2022

VIT BHOPAL UNIVERSITY, KOTHRIKALAN, SEHORE
MADHYA PRADESH – 466114

BONAFIDE CERTIFICATE

Certified that this project report titled **"RFID++: Testing existing security of RFID cards for next-gen authentication systems."** is the bonafide work of **"Saket Upadhyay (18BCY10079)"** who carried out the project work under my supervision. Certified further that to the best of my knowledge the work reported at this time does not form part of any other project/research work based on which a degree or award was conferred on an earlier occasion on this or any other candidate.

PROGRAM CHAIR
Dr. R. Rakesh, Assist. Prof., P.C. CSDF
School of Computer Science and Engineering
VIT BHOPAL UNIVERSITY

PROJECT GUIDE
Dr. (Prof.) Shishir K. Shandilya, Div. Head CSDF
School of Computer Science and Engineering
VIT BHOPAL UNIVERSITY

The Project Exhibition I Examination was held on 21 September 2021.

ACKNOWLEDGEMENT

I wish to express my heartfelt gratitude to Dr. Manas Kumar Mishra, Dean of the School of Computer Science and Engineering, Dr. Pushpinder Singh Patheja, Division Head of Cybersecurity and Digital Forensics Division (SCSE), and Dr. R. Rakesh, Program Chair, Cybersecurity and Digital Forensics Division (SCSE) for much of their valuable support and encouragement in carrying out this work.

I would like to thank my internal guides Dr. Shishir Kumar Shandilya and Dr. R. Rakesh, for continually guiding and actively participating in my project, and giving valuable suggestions to complete the project work.

I would like to thank all the technical and teaching staff of the School of Computer Science and Engineering, who extended directly or indirectly all support.

Last, but not least, I am deeply indebted to my parents who have been the greatest support while I worked day and night on the project to make it a success.

LIST OF FIGURES

ABSTRACT

RFID (Radio Frequency Identification) is a crucial application technology in the Internet of Things (IoT) technology. The terminal device needs to be authenticated before accessing the IoT network to avoid security holes. Due to the limited resources of the tag side in passive RFID systems, ultra-lightweight RFID authentication protocols are often used in such systems.

In the RFID systems during wireless communication between the tag and the reader, an adversary can perform malicious attacks and compromise data privacy. Apart from privacy and security issues, the computation capability and memory of the tags are very limited as compared to the backend server and the reader; therefore, an RFID-based authentication protocol suffers from high computational overhead.

This project explores the existing and recently proposed RFID protocols and strategies for authentication systems and attempts to find the research gap in the field. Also, we aim to create a demonstration of the RFID Card Cloning procedure where we will try to clone an existing production RFID ID card.

Some of the important expected contributions of this project are:

1. Creating a Code-Base for executing RFID based authentication module in Raspberry-Pi and Raspberry-Pico using RFID-RC522 chipset.
2. Creating an open-source code base for the Card Clone Demo for academia to use in security lectures and study of the basic low-level working of RFID-based authentication systems.
3. Shed some light on the existing research on RFID authentication.
4. An attempt to summarise and find potential for future work in this field.

Table of Contents

CHAPTER 1: PROJECT DESCRIPTION AND OUTLINE

1.1 Introduction

This project aims to understand the current state and requirements of Radio Frequency-based Identification (RFID) systems and their use in authentication. Authentication is one of the major factors in enforcing Confidentially and Integrity in the CIA triad in cyber security.

Authentication is the process of determining whether someone or something is who or what it says it is. With the help of RFID, we can use a contactless method of exchanging information to establish trust between two participating parties.

1.2 Motivation for the work

The seamless integration and ease of use of RFID systems inherit some fundamental security issues, as the size and power of computation are limited, if not negligible, in RFID chips, and securing the data stored in these chips is an ongoing challenge.

The limited computing power of silicon used in RFID tags and cards prohibits the use of more complex cryptographic functions to encrypt the data. Manufacturers usually end up using standard encoding to store the data which is not enough for security applications nowadays as mobile computing devices are getting more powerful and can intercept and process the data transmitted and manipulate them while having a small footprint, this introduces a big risk for enterprises and end-users in terms of secure implementation of such devices.

1.3 Problem Statement

With an exponential increase in electronics in our day-to-day life, our bias is increasing toward contactless/seamless authentication systems as they are easy to use and maintain. This is introducing more and more RFID-based technology into the consumer market, most of them are not properly regulated and end up as off-brand door locks or security systems on popular e-commerce websites like Amazon and Alibaba.

They create an illusion of security but, are pretty easy to override. This creates a serious problem, and the solution is the need to create a robust and affordable security algorithm/strategy for consumer-level RFID security and authentication.

1.4 The objective of the work

Based on the above observations, the main goal of the project is to understand the current state of authentication methods in RFID-based security systems and understand the need for next-generation RFID-based authentication systems.

While engaging in this small research for this capstone, we will try to get some significant side products that will help the community/students to replicate the algorithms and experiment on their own-

1. Creating a Code-base for executing RFID-based authentication module in Raspberry-Pi and Raspberry-Pico using RFID-RC522 chipset.
2. Shed some light on the existing research on RFID authentication.
3. An attempt to summarise and find potential for future work in this field.

1.5 Organization of the project

In this project, first, we will look into the existing work by doing a breadth-wise literature review, as we don't have enough time for an exhaustive literature review and complete research, we will try to understand the basics by following the research trends from 2015 and later.

After that, we will try to create an RFID scanner and writer so that we can manipulate RFID cards and we will use one blank card and my university's ID card for tests.[1]

All the code and infrastructure that we will use/generate during this work will be available on open-source platforms for others to use and replicate, supporting the idea of reproducible research.

[1] Vellore Institute of Technology Bhopal campus has not activated RFID based technologies in campus as of 8th May, 2022. Hence using the college ID will not do any damage to university asset. Should we damage the IC inside the card, there will not be any consequences as there is no active use of the digital capabilities of the card so far in/off campus. **The author advises against using an active card which is being used in your workspace's operation without prior permission from supervisors and security department**, any such action might result in fine and related punishment for damaging assets and risking security infrastructure of the workplace.

CHAPTER 2: INTRODUCTION TO EXISTING TECHNOLOGIES AND RELATED WORK

There are lots of work already done in the domain of RFID security and authentication strategies and many new standards have been proposed. Some of them solve major problems in the current state of RFID security, but it is not yet completely optimized when compared to other authentication and security methodologies. Due to the limited resources of the tag side in passive RFID systems, ultra-lightweight RFID authentication protocols are often used in such systems.

2.1 RFID Tags, Types, and Security

RFID tags have the advantages of low manufacturing cost and small size, they are widely used in practice. RFID tags are roughly divided into two categories: passive RFID tags and active RFID tags. The active RFID tag is equipped with a battery, and its service life is affected by the battery. It needs to be replaced regularly and cannot work normally at high or low temperatures. The passive RFID tags have no batteries. The working principle is: the reader transmits a certain frequency radio frequency signal through the transmitting antenna; when the tag enters the working area of the transmitting antenna, an induced current is generated; and the energy is obtained through the current for transmitting the signal from the RFID. Compared with active RFID tags, passive tags have the advantages of being maintenance-free, lower cost, longer service life, and can work normally at high temperatures. Therefore, the passive RFID system can be applied in a much wider scope. The wireless communication method in the RFID system makes the RFID system face many security threats [1]–[3], such as channel eavesdropping, tag or reader forgery, tag tracking, replaying information, and information tampering, and desynchronization attacks. In the RFID system, the security of information is ensured by verifying the legality of the identity of the tag and the reader. Passive RFID tags have limited computing and storage capabilities and cannot support conventional cryptographic functions such as symmetric encryption; therefore they usually use lightweight RFID authentication protocols or ultra-lightweight authentication protocols. The lightweight protocols require a random number generator and simple functions such as Cyclic Redundancy Code (CRC) checksum but not a hash function. The ultralightweight protocols only involve simple bitwise operations (e.g., XOR, AND, OR, etc.) on tags.

Early popular RFID ultra-lightweight security protocols include MMAP (Minimalist Mutual Authentication Protocol) [4], LMAP (Lightweight Mutual Authentication Protocol) [5], and EMAP (Efficient Mutual Authentication Protocol) [6]. The common point of these three protocols is that they no longer use traditional algorithms such as hash functions and block ciphers, but use bitwise operations which offer an adequate security level for certain applications and can be implemented even in the most limited low-cost RFID tags. Since then, a large number of ultra-lightweight protocols for RFID have been proposed to improve the security of the protocol, such as SASI [7] which can provide strong authentication and strong integrity of the transmissions, and RAPP [8] which proposed an ultra-lightweight authentication protocol with permutation operation, etc. For these protocols, many scholars have analyzed their security. For example, Li et al. analyzed the security of the protocol [4]–[6] and proposed corresponding attacks [9], [10]. Qurat et al. give the desynchronization attack and full disclosure attack on SASI and RAPP [11]. In 2017, Tewari and Gupta proposed a new ultra-lightweight authentication protocol [12], which is denoted by TGAP (Tewari and Gupta Authentication Protocol) in this paper. TGAP mainly utilizes two bit-operations, XOR and rotate operation. Compared with other protocols, TGAP further reduces the computational cost of tags and thus gains widespread attention. Many researchers have studied and analyzed TGAP. Wang et al. gave a full disclosure attack on the protocol and modified it to prevent this attack [13]. But Jing et al. pointed out that the modified protocol of Wang and TGAP are susceptible to full disclosure, man-in-the-middle, and desynchronization attacks [14]. Madiha et al. devised another desynchronization attack after analyzing TGAP [15], and Huang et al. designed a full disclosure attack on TGAP [16]. Therefore, the TGAP still has some significant security issues.

In [17], Gao et. al. propose a safe and efficient ultra-lightweight RFID mutual authentication protocol. The protocol uses only bitwise operations and is superior to existing ultra-lightweight protocols in terms of computational cost and communication overhead. Moreover, the protocol can effectively prevent typical

attacks, such as replay attacks, desynchronization attacks, man-in-the-middle attacks, etc., so it has good security performance.

[18], present an anonymous and secure ultralightweight RFID protocol for deployment on Internet of Vehicles systems, the designed protocol ensures the user's privacy in IoV systems.

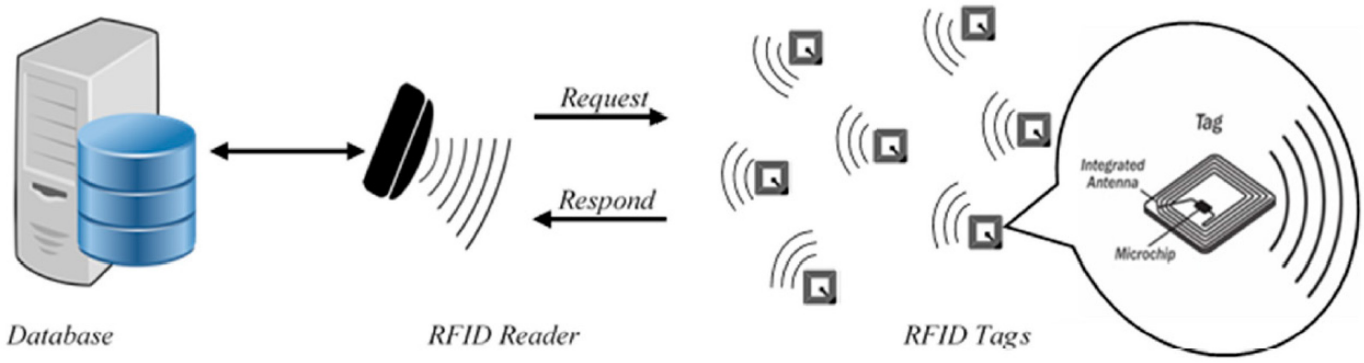

Figure 1: Basic Radio Frequency Identification (RFID) Model.[19]

2.2 Next-Gen Authentication Systems

A common approach to providing privacy-preserving is to use a pseudonym for RFID tags. That is, for each protocol session, each RFID tag uses a different temporary identifier (hence, the name pseudonym) so that malicious parties cannot track RFID tags. However, using different identifiers for every session makes it difficult for the back-end server to identify RFID tags being queried. Some privacy-preserving RFID authentication protocols such as [20] have O(N) worst-case complexity of looking up a tag in the back-end server's database where N is the number of tags in the database.

Many cryptographic protocols differentiate each other by improving this complexity as covered in [21]–[25]. [25] also suggested a possible alternative protocol in which the tag must encrypt its identifier itself using a memory-efficient variant of Rabin's encryption scheme.

2.3 Basic Security Requirements of RFID [19]

Mutual Authentication: the main requirement in a simple scenario of an RFID communication session is the authentication between the reader and tag before exchanging or transmitting any secret or valuable information. Both tag and reader have to prove their legitimacy to each other to start a secure communication.

Confidentiality: all of the transmitted messages have to be secure in which secret information and values that are used to execute communication cannot be obtained by an unauthorized user.

Integrity: the transmitted data has to maintain its accuracy and not be altered or changed during communication.

Availability: the communication should be successfully executed by maintaining a synchronous state between the RFID entities. Communication values have to be updated after every successful session to provide system availability.

Privacy: all of the secret information such as tag identity has to be secured to provide anonymity and avoid tracing the tag or its location.

Forward Security: the transmitted data during communication have to be independent and updated for every session, and cannot be used or related to another authentication session. If a tag or any information is compromised, an adversary can't pass the authentication on or violate the system.

2.4 Existing Security Threats

A secure RFID system must be able to resist different types of attacks. Messages in RFID communication are transmitted clearly, and thus are vulnerable to eavesdropping; hence, secret information is disclosed. Many RFID protocols are proposed to defend against different attacks such as:

1. **Replay Attack:** an adversary tries to capture the tag response and resend it to the reader to start a successful communication with the reader or obtain any secret information.
2. **Man-In-The-Middle:** an adversary intercepts the message between two legitimate entities tag/reader to modify it and send it back.
3. **Impersonate Attack:** an adversary obtains either the reader or tag identity information to create a forged entity. As a result, the adversary acts as a legitimate entity to pass the authentication and proceed with the communication.
4. **Traceability:** an adversary traces the tag to find its location and revokes the tag's privacy. This attack violates the private information of RFID users, which is an instance where privacy is important.
5. **Desynchronization Attack:** communication session between tag and reader starts using the synchronous values stored in both the tag and reader to authenticate each other. A desynchronization attack occurs when an adversary breaks the synchronous state between the tag and server by blocking the update messages, causing the communication values stored in both server and tag to be different.
6. **Denial of Service:** an adversary sends multiple signals simultaneously to the server as responses to make the system unavailable for further communication, which could further lead to a desynchronization attack.
7. **Cloning:** an adversary uses a malicious device to obtain the reader or tag secret information and create a fake entity that can be used to perform a successful communication.
8. **Disclosure:** an adversary identifies the secret information of the tag and the secret keys used in the communication to fully compromise the security of the protocol.

There are many other attacks on RFID systems, a robust RFID system should be able to deter these attempts and/or execute a well-defined fail-safe procedure is compromised.

2.5 Existing Security/Privacy-Preserving Algorithms

RFID protocols can be classified into four categories based on the complexity of the algorithm that is used to compute the tag responses: heavyweight, simple weight, lightweight, and ultra-lightweight. [26]

Heavyweight algorithms use symmetric and public-key cryptography that is beyond the scale of the passive tag's ability to process. Simple-weight algorithms use hash functions that are also not feasible for passive tag resources. Lightweight algorithms use simple one-way hash functions, cyclic redundancy checks, and pseudo-random number generators. [27]

Finally, ultra-lightweight algorithms use bitwise operations, which can be performed at a low cost.

Wang and Sarma [28] proposed two session-based authentication protocols, SB-A and SB-B, for the reader–tag authentication based on symmetric key encryption to ensure privacy and access control using two types of passive tags. The protocols are based on asymmetric cryptography algorithms to provide low-cost authentication such as the Advanced Encryption Standard (AES) and Data Encryption Standard (DES). Protocol SB-A in Figure 2 includes two processes. The first phase involves mutual authentication between server and tag according to the three-pass mutual authentication protocol according to the International Organization of Standardization and the International Electrotechnical Commission—ISO/IEC 9798-2. The second phase is for generating a session key between reader and tag according to the Otway–Rees protocol and updating the pseudo tag identity (PID). Protocol SB-B in Figure 3 uses tags with no memory or ID so that all of the tag's information is stored in the server. A physical tag operation is mapped with the digital virtual tag in the server that can do all of the tag's executions. The protocol time to keep synchronization is controlled by the tag nonce and counter, and not the server, because of the limited power of the tag to keep

synchronization. The protocols proved to be secure against major types of attacks; however, the protocols are considered to be heavyweight, since DES and AES are expensive operations that require a lot of computational overhead.

Server S	Reader R	Tag T
	Step 1: Send RID, OP_R to T →	**Step 2:** Send PID_n and nonce N_T to R ←
	Step 3: Send PID_n, N_T to server ←	
Step 4: Use PID to search the tag K_{TS} **Step 5:** Send E_{KTS} (N_T, N_S, PID_n) to R →	**Step 6:** Send E_{KTS} (N_T, N_S, PID_n) to T →	
	Step 8: Send to server ← - E_{KTS} (N_S, N_T), PID_n - RID, OP_R, N_R	**Step 7:** - Verify N_T to authenticate S - Send E_{KTS} (N_S, N_T), PID_n to R ←
Step 9: - Verify OP_R - Generate K_{RT} - Update PID_n to PID_{n+1} **Step 10:** Send to R - E_{KRS} (N_R, PID_n, RID, OP_R, K_{RT}) → - E_{KTS} (N_T, PID_{n+1}, RID, OP_R, K_{RT}) →	**Step 11:** - Retrieve K_{RT} - Send E_{KTS} (N_T, PID_{n+1}, RID, OP_R, K_{RT}) → - If OP_R is (write), encrypt info with K_{RT} and send it to T →	**Step 12:** - Retrieve K_{RT}, PID_{n+1}, RID, OP_R - Verify OP_R = OP_R in Step1 - Check the on-tag counter - Decode OP_R and execute it - Update PID_n to PID_{n+1} - If OP_R is (read), encrypt info with K_{RT} and send it to reader ←
K_{RS}: server/reader shared key; K_{TS}: server/tag shared key; K_{RT}: reader/tag shared key; N_T: nonce generated by tag; N_R: nonce generated by reader; N_S: nonce generated by server; RID: reader ID; OP_R: operation of reader; PID_n: pseudo-ID of tag in current session; $E_K(M)$: message encrypted by key K.		

Figure 2: Session-Based Authentication Protocol (SB-A) by Wang and Sarma.

Server S	Reader R	Tag T
	Step 1: Send RID, OP_R to T →	
		Step 2: Send PID_n and nonce N_T to R ←
Step 4: Use PID to search the tag K_{TS}	**Step 3:** Send PID_n, N_T to S ←	
Step 5: - Update PID_n to PID_{n+1} - Send E_{KTS}(N_T, N_S, PID_{n+1}) to R →	**Step 6:** Send E_{KTS}(N_S, N_T, PID_{n+1}) to T →	**Step 7:** - Verify N_T to authenticate S - Send E_{KTS} (N_S, N_T, RID, OP_R), PID_n to R ←
	Step 8:	- If OP_R is not (kill), update PID_n to PID_{n+1}
Step 9: - Verify reader authorization for OP_R	- Send E_{KTS} (N_S, N_T, RID, OP_R), PID_n to S ← - Send RID, OP_R, N_R to S ←	
Step 10: - If OP_R = read, send the message - If OP_R = kill: • Send E_{KTS} (N_T, PID_{n+1}, RID) to R → • Kill Vtag	**Step 11:** Send E_{KTS} (N_T, PID_{n+1}, RID) to T →	- Retrieve N_T, PID_{n+1}, RID - Verify RID = RID in step1 - Check on-tag counter with time limit - Perform physical kill operation
K_{RS}: server/reader shared key; K_{TS}: server/tag shared key; K_{RT}: reader/tag shared key; N_T: nonce generated by tag; N_R: nonce generated by reader; N_S: nonce generated by server; RID: reader ID; OP_R: operation of reader; PID_n: pseudo-ID of tag in current session; $E_K(M)$: message encrypted by key K; Vtag: virtual tag in the server.		

Figure 3: Session-Based Authentication Protocol (SB-B) by Wang and Sarma.

For traceability issues in RFID, Ryu et al. [29] proposed an elliptic curve cryptography-based untraceable authentication protocol (ECU) using the Schnorr signature scheme. The elliptic curve cryptography is considered to be public-key cryptography for RFID systems with low constrained tags. It is used to solve the issues of three recent elliptic curve-based untraceable RFID authentication protocols: Strong Privacy-preserving Authentication protocol (SPA) [30], Efficient Mutual Authentication protocol EMA [31], and ECC-based authentication protocol PII [32].

Ryu's protocol generates a digital signature with an appendix on the binary message of arbitrary length, and requires a cryptographic hash function, as shown in Figure 4. The sender's session key is combined with the receiver's public key to provide privacy, in which the message can be verified by only the receiver's private key. Ryu's protocol is secure against replay attacks, impersonate attacks and traceability attacks, and it maintains forward security. It requires two scalar multiplications, two hash functions, a message total size of 544 bits, and two communications between tag and reader. Even though this protocol requires complex computations associated with scalar multiplications and a hash function, it does not authenticate the reader.

It requires a cryptographic hash function, as shown in Figure 4. The sender's session key is combined with the receiver's public key to provide privacy, in which the message can be verified by only the receiver's private key. Ryu's protocol is secure against replay attacks, impersonate attacks and traceability attacks, and it maintains forward security. It requires two scalar multiplications, two hash functions, a message total size of 544 bits, and two communications between tag and reader. Even though this protocol requires complex computations associated with scalar multiplications and a hash function, it does not authenticate the reader.

Server S	Reader R	Tag T
Setup Phase: - Generate elliptic group G of prime order q. - Choose generator P of group G. - Server private/public keys (y, Y = yP) - Store tag verifier X = xP (public key) **Authentication Phase:**	**Step 1:** Send random c to T →	Store x, X, Y (server public key) **Step 2:** - Pick r as session secret - R = rP - v = H (R, c) - schnorr sign Z = rY, s = r + x * v - Encrypted verifier eid = X ⊕ H (R,s) - Send (eid, Z, s) to R ←
	Step 3: To authenticate tag - Compute R′ = y^{-1} Z - Derive X′ = eid ⊕ H (R′, s) - Check X′ = X registered verifier - Compute v′ = H (R′, c) - Authenticate the tag as H(sP − v′ X, c) = v′	
G: Cyclic additive group; P: Generator of group G; q: Order of group G; x_i: Tag's private key; ⊕ XOR; X_i: Tag's public key; y: Server's private; Y: Server's public; H: Hash function.		

Figure 4: Elliptic Curve Cryptography-Based Untraceable Authentication Protocol (ECU) by Ryu.

To reduce the tag's overhead in heavyweight Yao et al. [32] introduced The Reviving-UNder-DoS (RUND) authentication protocol to defend against denial of service (DoS) and preserve user privacy by powering up the tag to do complex computing for symmetric and public-key cryptography. It leverages the power in DoS scans to enable the tag to respond in two ways: either using simple encryption when the tag is activated by low signals from a reader or using public encryption (higher security) when the backscattered signals are high in an insecure environment. The more signals there are in communication, the more power charges the tag. The option of using public key encryption in RUND protocol is to overcome the problem of breaking up the synchronization state between the reader and tag in symmetric key encryption. The protocol is secure because secret information is not sent in clear, so no useful information can be gained if any message is compromised. Moreover, the parameters used in communication are changed and updated in every session, as shown in Figure 5, to prevent replay attacks, maintain forward security, and resist tracking.

Server: S	**Reader R:** PU_R, PR_R, shared K_i	**Tag T:** PU_R, shared K_i, ID
Initialization Phase:	**Step 1:** Precompute and store in S: $f(K_i, c, pad_1) \leftarrow$ Where pad is padding length for f()	- Counter c is set to 0.
Mutual Authentication Phase:	**Step 2:** Send power waves last for T_{pw} with energy E_c. Send PRN r1 in l length to tag →	**Step3:** Compute: **If Ec energy:** - $I_1 = f(K, c, pad_1)$ - $I_2 = r1 \| \| f(K, r1 \| \| I_1, pad_1)$ - $I = I_1 \| \| I_2$ - Update c = c + 1 - Energy consumed E_{sk} **If E_{pk} energy:** - $E(PU_R, K, r1 \| \| r2, ID, c)$ in l length - Energy consumed E_{pk} **Step4:** Send I to reader ←
	Step 5: If response with symmetric: - Check counter c and search database for f(K', c', pad_1) ← - Check r1 for replayed msg. - If matches: tag is authenticated. **If response with public key:** - Check and search database for (ID, K) pair ← - Check r1, r2 for replayed msg. - If matches: tag is authenticated	
Updating Phase:	**Step 6:** Generate r3 and compute $I_3 = r3 \| \| f(K, r3 \| \| I_1, pad_1)$ - Send I_3, r3 to tag → - Update $K = f(K, r3, pad_1)$ - Update precomputed $f(K_i, c, pad_1)$ with updated key. - Preserve old key of tag.	**Step 7:** Check I_3 using r3 by computing I'_3 - If matches: reader is authenticated. - Update $K = f(K, r3, pad_1)$ - C = 0
PU_R: Public key of reader; ID: Tag's ID; K_i: Shared symmetric key; c: Counter for current key lifecycle; PR_R: Private key of reader; pad_i: Padding for f(); E_c: The initial power the tag is charged; T_{PW}: Time for the power waves to last; E_{SK}: Energy consumption for hash function; E_{PK}: Energy consumption for public key.		

Figure 5: The Reviving-UNder-Denial of Service Authentication Protocol (RUND) by Yao.

Even though the overall efficiency of RUND is O(1), it is still not compliant with the Electronic Product Code Class1 Generation2 (EPC C1 G2) standard, which is defined by EPCGlobal Inc. for RFID data communication.

To better improve the performance of RFID protocols and reduce the power that is needed for complex operations in ECC-based protocols, Farash [33] proposed a mutual authentication protocol (IECC) based on the elliptic curve. The protocol enhances Chou's authentication protocol (EMA) [31], which does not fulfill the security requirement of forwarding security, mutual authentication, tag privacy, and security against location tracking, impersonating attacks, and tag cloning attack for an RFID system. The main idea behind the protocol is to use the server's public key to create the authentication message to avoid breaking the system privacy, as depicted in Figure 6. The IECC protocol is secure against major attacks, even though the computation cost is the same as in Chou's protocol that needs to be reduced for practical implementation.

Server S: {X_i, yP, P}	Reader R	Tag T: {X_i, Y, P}
Setup phase: - Generate an elliptic group G of prime order q - Choose generator P of group G - Choose random no. y as private key - Public key Y = yP - Choose random X from G as tag identifier - Store X_i, Y, P in each tag. **Authentication phase:** **Step 1:** - Choose a prime random no. r - Compute C0 = rP - Send C0 to tags →		**Step 2:** - Choose a prime random no. k - K = kP - C1 = kY - C2 = X_i + h(C0, C1, K) - Send C1, C2 to server →
Step 3: - Obtain K′ = y^{-1}C1 - Obtain X_i' = C2 – h(C0, C1, K′) - Find a match for X_i' in DB - If found: C3 = h(X_i', K′) and tag authenticated - Send C3 to tag →		**Step 4:** - Validate C3 = (X_i, K) - Server is authenticated
G: A additive group of prime order q; P: Generator of group G; h: One-way hash function; y: Server's private; Y: Server's public; X_i: Identifier of ith tag which is a random point in G.		

Figure 6: Mutual Authentication Protocol Based on Elliptic Curve Cryptography (IECC) by Farash.

Zhang and Qi [34] also proposed another protocol (EECC) to withstand the security weaknesses of Chou's protocol, EMA [31]. EECC protocol enhances patient medication safety by also using elliptic curve cryptography. In comparison to the EMA protocol, the EECC protocol resulted in better performance and security resistance to impersonate and forward security attacks.

B.Chen [35] proposed a role-based access control (RBAC) protocol for mobile RFID to enable user privacy, role, and access control through the back-end server based on a certification mechanism. RBAC assigns role classes as keys to control the information and the number of times each reader can read a tag. RBAC authorizes readers, assigns role classes to control the reader's authority to request tag information, and updates timestamps using random numbers and different shared keys between the database server and reader and tag ad, as depicted in Figure 7. Traceability and replay attacks are prevented using updated random numbers in every session; access control is provided using shared keys to prevent unauthorized readers to request or read any tag's information, and integrity is ensured using timestamps. However, RBAC uses one encryption mechanism that is excessive for low-cost passive tags.

Server: k_X, k_Y keys	Reader: k_Y keys	Tag: k_X keys
1- Reader Authorization and role class:	**Step 1:** Reader sends Hello to tag →	- Create random no. r1 - $M_1 = E_{k_X}$ (TID, TS, r1)
		Step 2: Sends M_1 to reader ←
	- Create random no. r2. - $M_2 = E_{k_Y}$ (M_1, r2, RID, Command)	
	Step 3: Send M_2 to server ←	
- Request role-class command, read tag command, TID, and RID from RBAC - RBAC sends role-class. - $M_3 = E_{k_Y}$(RID, r1, TS_1, $Cert_R$, role-class) - $M_4 = E_{k_X}$(TID, r2, TS_1, role-class) - Send M_3, M_4 to reader →		
	Step 4: - Retrieve r1, TS_1, $Cert_R$, role-class from M_3. - $M_5 = H(TS_1 \oplus r2)$ - Send M_4, M_5 to tag →	
2- Assign No. of reads and update timestamps:		**Step 5:** Verify M_5 using TS_1 from M_4 and its r1 to authenticate reader - Calculate number of reads $TC_{n-1} = TC_n - 1$ - if TS_1 is verified, it's updated to TS_2 - $M_6 = E_{k_X}$(TS_2, TC_{n-1}) - Send M_6 to reader ←
	Step 6: - Receive M_6 - $M_7 = E_{k_Y}$($Cert_R$, r2, M_6) - Send M_7 to database server ←	
Step 7: Retrieve $Cert_R$, r2 from M_7 - If $Cert_R$ is verified, retrieve TS_2, TC_{n-1} from M_6. - $M_8 = E_{k_Y}$(TS_2, TC_{n-1}, r2) - Send M_8 to reader →	**Step 8:** - Retrieve TS_2, TC_{n-1}, r2 from M_8 - Verify r2	
TID: Tag ID; K_Y: Server/Reader shared key; r: random number; TC_n: number of times a reader request information; K_X: Server/Tag shared key; TS: Timestamp; $Cert_R$: Reader security certificate; RBAC: role-based access control.		

Figure 7: Role-Based Access Control Protocol (RBAC) by B. Chen.

Fernando and Abawajy [36] proposed a mutual authentication protocol for Networked RFID Systems NRS, which is a lightweight mutual authentication scheme for an RFID system using low operations such as exclusive or operation (XOR) and one-way hash functions. However, Alagheband and Aref [27] reported NRS to be vulnerable to major attacks and specifically a full disclosure attack that compromises the whole RFID system. Alagheband and Aref improved the NRS protocol and proposed NRS+ by adding three more hash functions to the authentication message to increase the system security. Chen et al. [37] noted that the NRS+ protocol is exposed to desynchronization and traceability attacks by using one random number for the tag and reader. Thus, Chen proposed NRS++ to improve the security flaws in the previous versions of NRS by generating two different random numbers, r1 and r2, for the tag and reader using a pseudo-random number generator (PRNG) to defend against replay attack. In Figure 8, the authentication message M3 is encrypted using the tag's random number r1 and reader's random number r2 to provide message integrity, so any modified message cannot be verified by the tag. NRS++ uses fewer hash functions, which resulted in less computation overhead and storage space than the other versions, with more security power.

Server S	Reader R	Tag T
	Step 1: - Generate random no. r - Calculate $M_1 = H(EPC \oplus K_1 \| r)$ $M_2 = r \oplus K_1$ - Send to tag $M_1 \| M_2 \rightarrow$	**Step 2:** - Extract r as $r = M_2 \oplus K_1$ - Compute $C_1 = H(EPC \oplus K_1 \| r)$ If $C_1 = M_1$, generate r_1 $N = r_1 \oplus K_1$ $M_3 = H(EPC \oplus K_1 \| r \| r_1)$ Else termination - $\leftarrow$ Send $M_3 \| N$ to reader
- Update secrets in Database $ID_{new} = ID \oplus (r_{2right} \| K_{1left})$ $K_{1new} = H[(K_{1right} \| r_{1left}) \oplus r_2]$	**Step 3:** - Extract $r_1 = N \oplus K_1$ - Compute $C_2 = H(EPC \oplus K_1 \| r \| r_1)$ - Verify $C_2 = M_3$ If equal: Generate random no. r_2 $M_4 = r_2 \oplus K_1$ $M_5 = H(EPC \oplus K_1 \| r_1 \| r_2)$ If not equal: terminate - Send $M_4 \| M_5 \rightarrow$	**Step 4:** - Extract r_2 as $r_2 = M_4 \oplus K_1$ - Compute $C_3 = H(EPC \oplus K_1 \| r_1 \| r_2)$ - Verify $C_3 = M_5$ If equal: Update the secrets. If not equal: terminate
ID, EPC: Tag identifier; H(): one-way hash function; K_1: Server/Tag shared key; r, r_1, r_2: random No; $\oplus$/||: XOR and concatenation operation.		

Figure 8: Mutual Authentication Protocol for Networked RFID Systems (NRS++) by X. Chen.

C. Chen [38] proposed Anti-Counting Security Protocol (ACSP) as another lightweight protocol for RFID systems to defend from a counter-attack, which is defined as the attacker's ability to count the number of objects in a system. Safkhani et al. [39] reported ACSP to be vulnerable to major attacks, including the forward/backward traceability attack. Safkhani further proposed ACSP+ to improve Chen's protocol. Later, X. Chen [37] pointed out that the ACSP protocol is not secure, and proposed ACSP++ to withstand DoS and forward/backward traceability attacks. ACSP++ enhances the session identifier (SID) update, which is used to verify the current session, and tag identification phases that suffer from different attacks in ACSP and ACSP+ versions. In ACSP++ as depicted in Figure 9, a tag identifier (TID) is added to the identification message as (IDENT, R4, R5, TID) instead of (IDENT, R4, R5), and the authentication message (AUTHEN, R4, R5, TID) is replaced with (AUTHEN, R5, TID) to overcome DoS attack and modifying the TID in the identification phase. The update phase of every key is associated with two separate nonce values to avoid forward and backward traceability.

Even though the protocol improved the security weaknesses of all of the ACSP versions, it did not lower the computation overhead or the storage space.

Reader R	Tag T
(SID Update Phase) Step 1: - Generate nonce R1 - Send the following to tag: $\overline{UPDSID}$, R1⊕SID, H($\overline{UPDSID}$, R1, SID) →	**Step 2:** - Extract R1 to verify H($\overline{UPDSID}$, R1, SID) - Generate R_2 - Update SID SID_{new} = H(SID||R2||R1) $SID_{old} = SID_{cur}$
Step 3: - Extract R2 and verify H($\overline{UPDACK}$, R2, R1, SID) - Update SID as SID_{new} = H(SID||R2||R1)	- ← Send to reader confirmation: $\overline{UPDACK}$, R2 ⊕ SID, H($\overline{UPDACK}$, R2, R1, SID)
(Tag Identification Phase) Step1: - Generate R3, R4 - Send the following messages to tag → a) $\overline{SELECT}$, SID1⊕ R3, H($\overline{SELECT}$, R3, SID)) b) $\overline{QUERY}$, SID ⊕ TID ⊕ R4, H($\overline{QUERY}$, R4, SID, TID))	**Step 2:** - Extract R3′ to verify H($\overline{SELECT}$, R3, SID) If not verified: wait until next run. If verified: respond with step3.
	Step 3: - Extract R4′ to verify H($\overline{QUERY}$, R4, SID, TID) - Generate R5 - ← Send ($\overline{IDENT}$, TID ⊕ R5, H($\overline{IDENT}$, R4, R5, TID)
Step 4: - Authenticate tag - Extract R5′ to verify H($\overline{IDENT}$, R4, R5, TID) - If not verified: stop the session and send $\overline{QUERY\ REP}$ → - If verified: update TID as TID_{new} = H(TID||R4||R5) TID_{old} = TID - Send ($\overline{AUTHEN}$, H($\overline{AUTHEN}$, R5, TID) →	**Step 5:** - Calculate and verify H($\overline{AUTHEN}$, R5, TID) - If not verified: stop the session. - If verified: update the tag identifier as TID_{new} = H(TID||R4||R5)
R1, R2, R3, R4, R5: nonce; $\overline{SELECT}$/$\overline{QUERY}$: Select/ query commands; SID_{cur}/ SID_{new}: Current/ New session identifier; $\overline{UPDSID}$/$\overline{UPDACK}$: SID update/ Update knowledge message; TID_{cur}/TID_{new}: Current/ New unique identifier; $\overline{IDENT}$/ $\overline{AUTHEN}$: Identification/ authentication messages.	

Figure 9: Anti-Counting Security Protocol (ACSP++) by X. Chen.

Chien and Huang [26] presented LAP, which is a lightweight authentication protocol to solve the vulnerabilities in the authentication protocol of Li et al. [40] and enhance the computational cost from O(n) to O(1) in identifying tags in RFID systems. The security of the LAP protocol is based on a synchronized PRNG between reader and tag using a secret key, secret ID, and index pseudonym. In Figure 10, the LAP protocol uses the rotate operator on the message and the left/right operator for the divided rotation during the messages that were exchanged to form a secure permutation. Random numbers are used to shift the secret values of the tag to be used safely in communication. Then, the random number is XORed with the shifted secret value to securely retrieve a tag by the server. The server uses the index pseudonym (IDS) to quickly identify the tag in the database instead of computing PIDL ⊕ PIDR for every tag to make the computation O(1). LAP protocol is resistant to replay attacks, DoS, and forward security. It can be employed easily by different standards such as EPC Gen2 and ISO 15693 for practical implementation. However, the protocol was noted as being partially secure against traceability and synchronization attacks, since a tag can be traced between two successful sessions if the tag could not update its IDS.

Server S: flag, X_{old}, X_{new}, IDS_{old}, IDS_{new}, SID	Reader R	Tag T: {SID, IDS, X}
	Step 1: - Generate R_1. - Send Query||R_1 to T →	- Generate R_2 - Compute $g' = g(R_1 \| \| R_2 \| \| X)$ - SID′ = rotate(SID, g′) - R′ = left(SID′ ⊕ g′)
Step 2: - Search IDS_i - If IDS == IDS_{old}: flag = 0, X = X_{old} - If IDS == IDS_{new}: flag = 1, X = X_{new} - $g' = g(R_1 \| \| R_2 \| \| X)$ - SID′ = rotate(SID, g′) - Verify R′ as R′ = left(SID′ ⊕ g′) - Compute R″ = right(SID′ ⊕ g′) - If flag = 1 • $IDS_{old} = IDS_{new}$ • $X_{old} = X_{new}$ - Else • $IDS_{new} = g(IDS \| \| SID')$ • $X_{new} = g(X \| \| g')$ - Send R″ to reader →	- Forward R1||R_2||R′||IDS to S ←	- Send R_2||R′||IDS to R ←
	- Forward R″ to T →	Step 3: - Verify R″ right(SID′ ⊕ g′) - Update: • IDS = g(IDS||SID′) • $X = X_{new} = g(X \| \| g')$
Step 4: - When OK is received, send SID to R →	- Forward ACK to S ←	- Send ACK to R ←
SID: Secure ID; PID: Partial ID; IDS: Index pseudonym; g(): Random No. generator; X: l-bit secret key; R_1, R_2: Random numbers; Rotate(): Rotation function; Left(s): Left half of s; Right(s): Right half of s; ACK: Acknowledgement.		

Figure 10: Lightweight Authentication Protocol (LAP) by Chien.

Burmester and Munilla [41] proposed an authentication protocol called Flyweight that is based on exchanging messages using only PRNG. Their protocol is based on a shared PRNG algorithm between the tags and the back-end server that takes the same seed to produce the same output. The concept of the protocol is to use three consecutive numbers—RN1, RN2, and RN3—generated by the same PRNG in the server, and the tags of five numbers if an active adversary is presented, such as in Figure 11. Furthermore, RFID tags precompute the values to the server challenging the response, so an adversary can be detected based on the response time from the tag. The protocol can provide mutual authentication, integrity, confidentiality, and forward and backward security. In addition, it provides strong synchronization, since the server keeps a record of the current and next response value of the tag. S. Lee et al. [42] proposed a lightweight protocol (MASS) for RFID systems using XOR and a one-way hash function to conform to the scarce resources of RFID tags. The concept of the MASS protocol is to challenge the tag with a fresh random string every session, and the tag responds using the reader's value and its random key to authenticate the reader ad, as depicted in Figure 12. The secret key is shared between entities, and all of the messages are encrypted during transmission. However, Zuo [43] conducted a survivability experiment on the authentication protocol proposed by S. Lee et al. and defined the vulnerability of the protocol to replay, desynchronize, and impersonate attacks. Zuo concluded from his experiment that the system could employ two different values for the keys (old, and new) to recognize the tag and overcome the desynchronization problem.

Server S	Reader R	Tag T
	Step 1:. - Send Query to T →	- RN1 = g_{tag} (state) - Set alarm cnt = 1 - Send RN1 to R ←
	Step 2:	
- Check if RN1 = $RN1^{cur}$ • cnt = 1 • Generate RN2, send RN2 to R → - If RN1 = $RN1^{next}$ • cnt = 0 • Update values in DB • Send updated RN2 to R →	- Forward RN1 to S ←	
	- Forward RN2 to T →	**Step 3:** - If RN2 is correct to authenticate S • Generate RN3, RN4, RN5 • Cnt = 0 - If cnt = 0, send RN3 to R ← - If cnt = 1, send RN4 to R ←
Step 4: - If RN = RN3, and cnt = 0 • Tag is authenticated - If RN = RN4 • Send RN3, store RN5 • Update values • Send RN3 to R	- Forward RN4 to S ←	
	- Forward RN3 to T →	**Step 5:** - If RN3 is correct and cnt = 1 • Send RN5 to R ← - Else terminate
	- Forward RN5 to S ←	
Step 6: - If RN5 is correct • Authenticate T • Update values - Else terminate		
RN: Random numbers output of the same generator function		cnt: l-bit flag

Figure 11: Flyweight Mutual Authentication Protocol by Burmester and Munilla.

Server S	Reader R	Tag T
	Step 1: - Generate l-bit string str - Send str to tag →	
		Step 2: - Generate l-bit string rA - rB = h(rA ⊕ Ki ⊕ str) - rC = h(rB ⊕ Ki ⊕ str) - Send rB, rC to reader ←
	Step 3: - Search database to match key Ki - If found proceed to update key - Retrieve rB from rC - Ki = h(Ki) - r'C = h(rB ⊕ Ki ⊕ str) - Send r'C to tag →	
		Step 4: - Verify r'C = rC - If verified, update key
Ki: Tag/server shared secret key; h(): One-way hash function		

Figure 12: Lightweight Protocol based on Synchronized Secret (MASS) by S. Lee.

2.6 Limitations of Current Approaches

Recent RFID authentication protocols are proposed to develop an efficient and secure RFID system. The survey in [19] is conducted to review and compare different RFID authentication protocols of low-cost passive tags for better utilization in the appropriate application. They demonstrate in this study the security requirements of an RFID system that must be satisfied, so the system could be able to defend against major attacks such as replay, man-in-the-middle, impersonation, desynchronization, DoS, and more. They further identify the category levels of the protocols based on the operation complexity on the tag side and compare the protocols based on the tag computation cost. Since the RFID passive tag has limited resources to compute complex operations, the heavyweight and simple-weight protocols are not feasible for practical implementation. However, lightweight and ultra-lightweight protocols use only simple operations within the tag computation limits and show the lowest tag overhead level. Lightweight and ultra-lightweight protocols are considered the most suitable for the current applications. Another vital aspect when considering the appropriate RFID protocol is the security resistance to the attacks. Examining the security threats in each protocol presented in the review, we found out that Chang et al. [44], Farash [33], Zhang and Qi [34], X. Chen et al. [37], Liu et al., Lin and Song [53], and Ouaskou et al. [56] protocols successfully resist all of the major attacks. Although the other protocols could not resist all of the attacks, they could perform better than the fully secure protocols in terms of computation cost; examples include the protocols presented in Farash [19], Zhang and Qi [34], X. Chen et al. [37], and Liu et al. [45], have high computation overhead on the tag side.

CHAPTER 3: PROJECT PREREQUISITE

3.1 Introduction

MFRC522 supports the MIFARE series of high-speed non-contact communication, a two-way data transmission rate up to 424kbit/s. It supports rapid CRYPTO1 encryption algorithm and terminology validation MIFARE products. The MF RC522 is a highly integrated transmission module for contactless communication at 13.56 MHz. RC522 supports ISO 14443A/MIFARE mode.[2] The project depends on basic hardware and software components to work with RC522 as mentioned below-

3.2 Hardware Requirements

Hardware requirements for this project/prototype are-

1. Microcontroller (RP2040[3] [4])
2. Microprocessor (Raspberry Pi Zero W[5] 1GHz, single-core CPU)
3. A Laptop/Desktop Computer
4. Bread Board
5. RC522 RFID Card Reader Module 13.56MHz[6]
6. Jumper Cables
7. Read/Write RFID Blank Card
8. Read/Write RFID Blank Token
9. A working real-world RFID Card/Token
10. Male to Female Connector Headers
11. Soldering Equipment

3.3 Software Requirements

We don't need any specific proprietary software/stack but we will need some open-source frameworks and programming language support, as follows-

1. Linux Operating System for Microprocessor
2. Python3
3. C/C++ development toolchain
4. MicroPython toolchain for RP2040
5. Opensource RFID libraries.

[2] https://www.ibeyonde.com/configure-rfid-rc522-raspberry-pi.html
[3] https://www.raspberrypi.com/products/rp2040/
[4] https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf
[5] https://www.raspberrypi.com/products/raspberry-pi-zero-w/
[6] https://robu.in/product/mifare-rfid-readerwriter-13-56mhz-rc522-spi-s50-fudan-card-and-keychain/

CHAPTER 4: DESIGN AND METHODOLOGY

4.1 Hardware Configuration (Microprocessor)

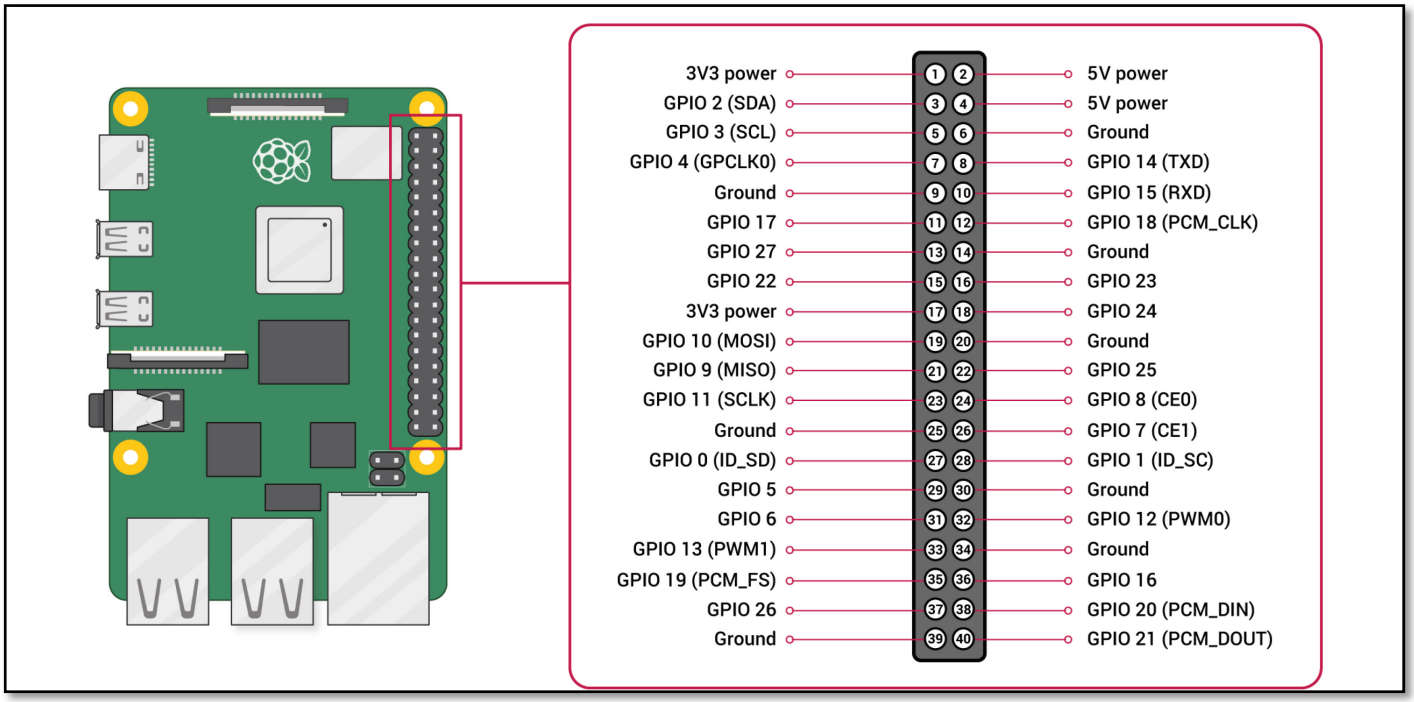

Figure 13: Raspberry Pi General GPIO Layout

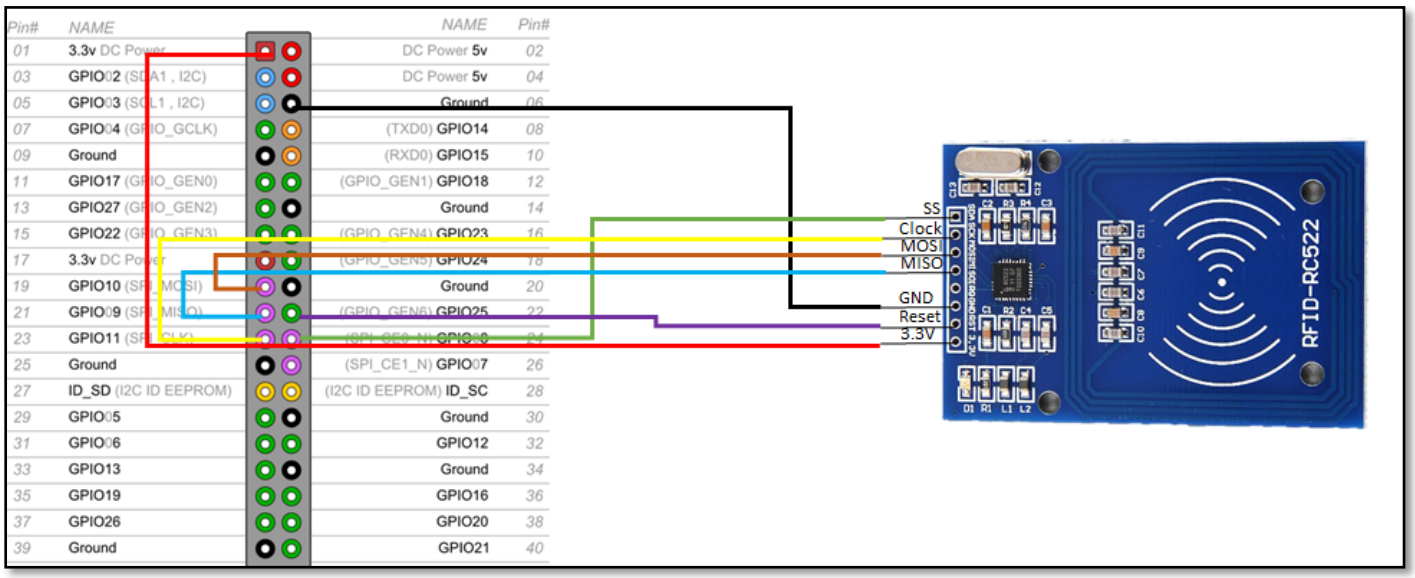

Figure 14: RC522 connection schematic with RP Microprocessor GPIO Pins

This connection schematic shows the basic connection of RC522 with General Purpose I/O pins of Raspberry Pi Zero W microprocessor, this is the basic setup, and we don't need any other major hardware to connect with our Microprocessor. Some switches and LCDs can be added as cosmetics. For results and input, we will use a serial terminal on our computer.

4.1.1 Setup and Initialisation of Hardware[7]

[7] https://www.ibeyonde.com/configure-rfid-rc522-raspberry-pi.html

Enable SPI Interface

On Raspberry Pi, the SPI interface is not enabled by default. We start by turning the SPI interface on by uncommenting `#dtparam=spi=on` in `/boot/config.txt`. On reboot you should see the SPI module loaded as shown below:

```
$lsmod | grep spi

spi_bcm2835 7596 0

spidev 7373 0
```

You should also see the following SPI devices activated:

```
$ls /dev/spi*

/dev/spideev0.0   /dev/spidev0.1
```

4.1.2 Software Setup

4.1.2.1 Loopback test for SPI Interface

Put a wire between MOSI (pin 19) and MISO(pin 21) pins. Then run the following from command line.

```
$wget
https://raw.githubusercontent.com/raspberrypi/linux/rpi3.10.y/Documentation/spi
/spidev_test.c

$gcc -o spidev_test spidev_test.c

$./spidev_test -D /dev/spidev0.0

spi mode: 0

bits per word: 8

max speed: 500000 Hz (500 KHz)

FF FF FF FF FF FF

40 00 00 00 00 95

FF FF FF FF FF FF

FF FF FF FF FF FF

FF FF FF FF FF FF

DE AD BE EF BA AD

F0 0D
```

If you see the above output, you are all set to move further.

4.1.2.2 Install the module that communicates with SPI devices

Install SPI-Py; this lets you interact with the SPI devices. The current state of a repository is not good; that is the reason we will check out a particular commit from it.

```
$git clone https://github.com/lthiery/SPI-Py.git

$cd SPI-Py
```

```
$git checkout 8cce26b9ee6e69eb041e9d5665944b88688fca68

$python setup.py install
```

4.1.2.3 Install the interface for RC522

MFRC522-python is a small class to interface with the NFC reader Module MFRC522 on Raspberry Pi. MFRC522-python is a Python port of the example code for NFC Module MF522-AN.

```
$git clone https://github.com/mxgxw/MFRC522-python.git

$cd MFRC522-python

$pyhton Read.py
```

4.1.2.4 Our Hardware Setup

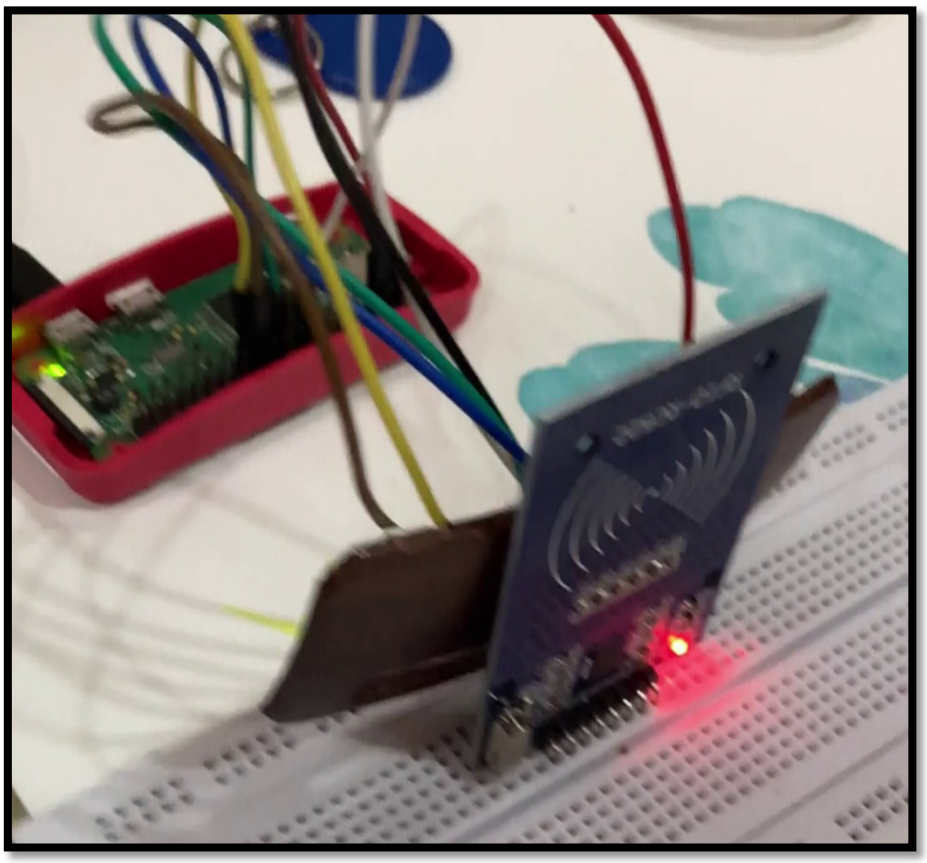

Figure 15: Experiment RC522 setup with Raspberry Pi Microprocessor

4.2 Hardware Configuration (Micro Controller) [46]

RC522 RFID Reader Module	Raspberry Pi Pico
VCC	3.3V
RST	GP0
GND	GND
IRQ	Not connected
MISO	GP4
MOSI	GP3
SCK	GP2
SDA	GP1

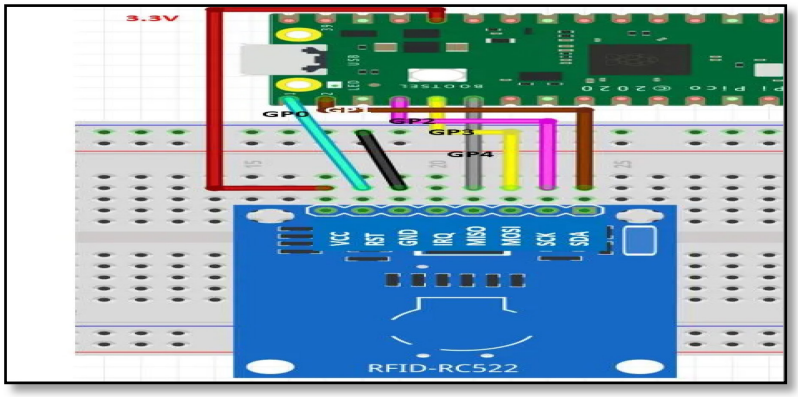

Figure 16: MFRC522 to Pico connection schematic[8]

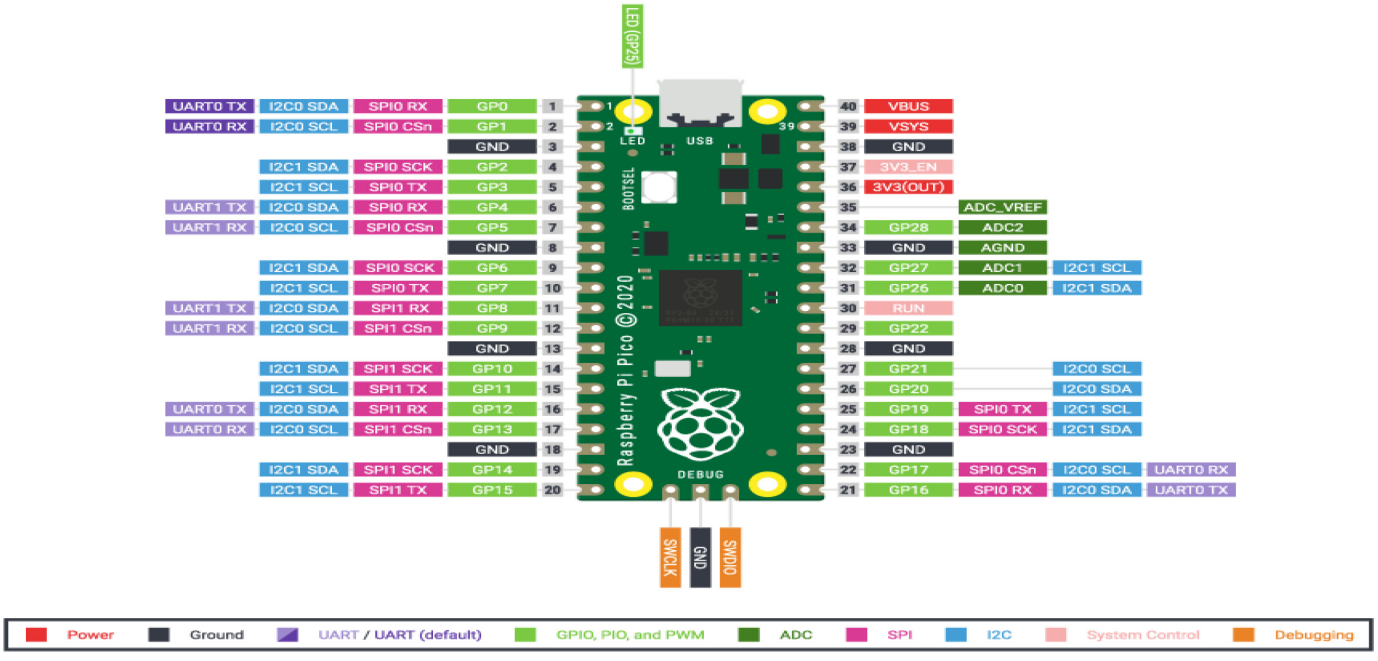

Figure 17: RP2040 Pin Out[9]

[8] https://microcontrollerslab.com/raspberry-pi-pico-rfid-rc522-micropython/
[9] https://microcontrollerslab.com/wp-content/uploads/2021/01/Raspberry-Pi-Pico-pinout-diagram.svg

CHAPTER 5: EXPERIMENT AND OBSERVATIONS

5.1 Code Base (Also Available on GitHub)[47]

1.1.1 Code License (MIT)

The code is licensed under the MIT Opensource license as mentioned in “Appendix I”.

1.1.2 Mfrc522.py (Library)

```
from machine import Pin, SPI
from os import uname

class MFRC522:
    OK = 0
    NOTAGERR = 1
    ERR = 2

    REQIDL = 0x26
    REQALL = 0x52
    AUTHENT1A = 0x60
    AUTHENT1B = 0x61

    def __init__(self, sck, mosi, miso, rst, cs):

        self.sck = Pin(sck, Pin.OUT)
        self.mosi = Pin(mosi, Pin.OUT)
        self.miso = Pin(miso)
        self.rst = Pin(rst, Pin.OUT)
        self.cs = Pin(cs, Pin.OUT)

        self.rst.value(0)
        self.cs.value(1)

        board = uname()[0]

        if board == 'WiPy' or board == 'LoPy' or board == 'FiPy':
            self.spi = SPI(0)
            self.spi.init(SPI.MASTER, baudrate=1000000,
                          pins=(self.sck, self.mosi, self.miso))
        elif board == 'esp8266':
```

```
            self.spi = SPI(baudrate=100000, polarity=0, phase=0,
                           sck=self.sck, mosi=self.mosi, miso=self.miso)
            self.spi.init()
        elif board == 'rp2':
            self.spi = SPI(0)
            self.spi = SPI(0, baudrate=1000000, sck=self.sck,
                           mosi=self.mosi, miso=self.miso)
        else:
            raise RuntimeError("Unsupported platform")

        self.rst.value(1)
        self.init()

    def _wreg(self, reg, val):

        self.cs.value(0)
        self.spi.write(b'%c' % int(0xff & ((reg << 1) & 0x7e)))
        self.spi.write(b'%c' % int(0xff & val))
        self.cs.value(1)

    def _rreg(self, reg):

        self.cs.value(0)
        self.spi.write(b'%c' % int(0xff & (((reg << 1) & 0x7e) | 0x80)))
        val = self.spi.read(1)
        self.cs.value(1)

        return val[0]

    def _sflags(self, reg, mask):
        self._wreg(reg, self._rreg(reg) | mask)

    def _cflags(self, reg, mask):
        self._wreg(reg, self._rreg(reg) & (~mask))

    def _tocard(self, cmd, send):

        recv = []
        bits = irq_en = wait_irq = n = 0
```

```
        stat = self.ERR

        if cmd == 0x0E:
            irq_en = 0x12
            wait_irq = 0x10
        elif cmd == 0x0C:
            irq_en = 0x77
            wait_irq = 0x30

        self._wreg(0x02, irq_en | 0x80)
        self._cflags(0x04, 0x80)
        self._sflags(0x0A, 0x80)
        self._wreg(0x01, 0x00)

        for c in send:
            self._wreg(0x09, c)
        self._wreg(0x01, cmd)

        if cmd == 0x0C:
            self._sflags(0x0D, 0x80)

        i = 2000
        while True:
            n = self._rreg(0x04)
            i -= 1
            if ~((i != 0) and ~(n & 0x01) and ~(n & wait_irq)):
                break

        self._cflags(0x0D, 0x80)

        if i:
            if (self._rreg(0x06) & 0x1B) == 0x00:
                stat = self.OK

                if n & irq_en & 0x01:
                    stat = self.NOTAGERR
                elif cmd == 0x0C:
                    n = self._rreg(0x0A)
                    lbits = self._rreg(0x0C) & 0x07
```

```
                if lbits != 0:
                    bits = (n - 1) * 8 + lbits
                else:
                    bits = n * 8

                if n == 0:
                    n = 1
                elif n > 16:
                    n = 16

                for _ in range(n):
                    recv.append(self._rreg(0x09))
        else:
            stat = self.ERR

    return stat, recv, bits

def _crc(self, data):

    self._cflags(0x05, 0x04)
    self._sflags(0x0A, 0x80)

    for c in data:
        self._wreg(0x09, c)

    self._wreg(0x01, 0x03)

    i = 0xFF
    while True:
        n = self._rreg(0x05)
        i -= 1
        if not ((i != 0) and not (n & 0x04)):
            break

    return [self._rreg(0x22), self._rreg(0x21)]

def init(self):

    self.reset()
```

```
        self._wreg(0x2A, 0x8D)
        self._wreg(0x2B, 0x3E)
        self._wreg(0x2D, 30)
        self._wreg(0x2C, 0)
        self._wreg(0x15, 0x40)
        self._wreg(0x11, 0x3D)
        self.antenna_on()

    def reset(self):
        self._wreg(0x01, 0x0F)

    def antenna_on(self, on=True):

        if on and ~(self._rreg(0x14) & 0x03):
            self._sflags(0x14, 0x03)
        else:
            self._cflags(0x14, 0x03)

    def request(self, mode):

        self._wreg(0x0D, 0x07)
        (stat, recv, bits) = self._tocard(0x0C, [mode])

        if (stat != self.OK) | (bits != 0x10):
            stat = self.ERR

        return stat, bits

    def anticoll(self):

        ser_chk = 0
        ser = [0x93, 0x20]

        self._wreg(0x0D, 0x00)
        (stat, recv, bits) = self._tocard(0x0C, ser)

        if stat == self.OK:
            if len(recv) == 5:
                for i in range(4):
```

```
                ser_chk = ser_chk ^ recv[i]
            if ser_chk != recv[4]:
                stat = self.ERR
        else:
            stat = self.ERR

    return stat, recv

def select_tag(self, ser):

    buf = [0x93, 0x70] + ser[:5]
    buf += self._crc(buf)
    (stat, recv, bits) = self._tocard(0x0C, buf)
    return self.OK if (stat == self.OK) and (bits == 0x18) else self.ERR

def auth(self, mode, addr, sect, ser):
    return self._tocard(0x0E, [mode, addr] + sect + ser[:4])[0]

def stop_crypto1(self):
    self._cflags(0x08, 0x08)

def read(self, addr):

    data = [0x30, addr]
    data += self._crc(data)
    (stat, recv, _) = self._tocard(0x0C, data)
    return recv if stat == self.OK else None

def write(self, addr, data):

    buf = [0xA0, addr]
    buf += self._crc(buf)
    (stat, recv, bits) = self._tocard(0x0C, buf)

    if not (stat == self.OK) or not (bits == 4) or not ((recv[0] & 0x0F) == 0x0A):
        stat = self.ERR
    else:
        buf = []
        for i in range(16):
```

```
                buf.append(data[i])
            buf += self._crc(buf)
            (stat, recv, bits) = self._tocard(0x0C, buf)
            if not (stat == self.OK) or not (bits == 4) or not ((recv[0] & 0x0F) == 0x0A):
                stat = self.ERR
        return stat
```

1.1.3 readRFID.py

```
import mfrc522
from os import uname

print("Initialising Module=> "+str(uname()[0]))
rdr = mfrc522.MFRC522(sck=2, miso=4, mosi=3, cs=1, rst=0)
print("")
print("Place card before reader. READ ARRD: 0x08")
print("")

try:
    while True:
        (stat, tag_type) = rdr.request(rdr.REQIDL)
        if stat == rdr.OK:
            (stat, raw_uid) = rdr.anticoll()
            if stat == rdr.OK:
                print("CARD DETECTED")
                print(" -  TAG TYPE : 0x%02x" % tag_type)
                print(" -  UID      : 0x%02x%02x%02x%02x%02x" %
                      (raw_uid[0], raw_uid[1], raw_uid[2], raw_uid[3]))
                print("")
                if rdr.select_tag(raw_uid) == rdr.OK:
                    key = [0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF]

                    if rdr.auth(rdr.AUTHENT1A, 8, key, raw_uid) == rdr.OK:
                        data = rdr.read(8)
                        datastr = ""
                        hexstr = []
                        for i in data:
                            datastr=datastr+(chr(i))
```

```
                            hexstr.append(hex(i))
                        print("DATA: "+str(datastr))
                        print("RAW DATA: "+str(hexstr))
                        rdr.stop_crypto1()
                    else:
                        print("AUTH ERR")
                else:
                    print("Failed to select tag")

except KeyboardInterrupt:
    print("EXITING PROGRAM")
```

1.1.4 RFIDwrite.py

```
import mfrc522
from os import uname

print("Init. "+str(uname()[0]))

rdr = mfrc522.MFRC522(sck=2, miso=4, mosi=3, cs=1, rst=0)

print("")
print("Place card before reader. WRITE ADDR: 0x08")
print("")

try:
    while True:

        (stat, tag_type) = rdr.request(rdr.REQIDL)

        if stat == rdr.OK:

            (stat, raw_uid) = rdr.anticoll()

            if stat == rdr.OK:
                print("CARD DETECTED")
                print(" -  TAG TYPE : 0x%02x" % tag_type)
                print(" -  UID      : 0x%02x%02x%02x%02x%02x" %
```

```
                    (raw_uid[0], raw_uid[1], raw_uid[2], raw_uid[3]))
                print("")

                if rdr.select_tag(raw_uid) == rdr.OK:

                    key = [0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF]

                    if rdr.auth(rdr.AUTHENT1A, 8, key, raw_uid) == rdr.OK:
                        stat = rdr.write(8, b"\x53\x61\x6b\x65\x74")
                        rdr.stop_crypto1()
                        if stat == rdr.OK:
                            print("DATA WRITTEN TO ADDRESS 0x08")
                        else:
                            print("FAILED")
                    else:
                        print("AUTH ERR")
                else:
                    print("Failed to select tag")
except KeyboardInterrupt:
    print("EXITING PROGRAM")
```

1.1.5 KeyGen.py

```
import string
import random
from pip import main

class GetKey:
    def getRandomKey(self, keylen):
        LETTERS = string.ascii_letters
        NUMBERS = string.digits
        passlength = keylen
        printable = f'{LETTERS}{NUMBERS}'
        printable = list(printable)
        random.shuffle(printable)
        random_password = random.choices(printable, k=passlength)
        random_password = ''.join(random_password)
        return random_password
```

```
if __name__ == "__main__":
    DataLength = 16
    RNDKEY = GetKey()
    datastring = RNDKEY.getRandomKey(16)
    print("Key : " + datastring)
    datahex = []
    for i in datastring:
        datahex.append(hex(ord(i)))
    if len(datahex) != 16:
        Padding = 16 - len(datahex)

    ModData = ""
    for val in datahex:
        ModData = ModData + "\\" + str(val)[1:]

    if len(datahex) != 16:
        for i in range(0, Padding):
            ModData = ModData + "\\x00"
    print("HexData to Write : \"" + ModData + "\"")
```

1.1.6 CloneCardData.py

```
import mfrc522
from os import uname
from time import sleep

class RFIDCard:

    def __init__(self) -> None:
        self.rdr = mfrc522.MFRC522(sck=2, miso=4, mosi=3, cs=1, rst=0)

    def ReadData(self):
        print("")
        print("Place Original card before reader.")
        print("")
        try:
            while True:
```

```
                (stat, tag_type) = self.rdr.request(self.rdr.REQIDL)
                if stat == self.rdr.OK:
                    (stat, raw_uid) = self.rdr.anticoll()
                    if stat == self.rdr.OK:
                        print("CARD DETECTED")
                        print(" -  TAG TYPE : 0x%02x" % tag_type)
                        print(" -  UID      : 0x%02x%02x%02x%02x" %
                              (raw_uid[0], raw_uid[1], raw_uid[2], raw_uid[3]))
                        print("")
                        if self.rdr.select_tag(raw_uid) == self.rdr.OK:
                            key = [0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF]
                            if   self.rdr.auth(self.rdr.AUTHENT1A,   8,   key,
raw_uid) == self.rdr.OK:
                                data = self.rdr.read(8)
                                return (data)
                            else:
                                print("AUTH ERR")
                        else:
                            print("Failed to select tag")
        except KeyboardInterrupt:
            print("EXITING PROGRAM.")
    def WriteData(self, databytes):
        rdr = mfrc522.MFRC522(sck=2, miso=4, mosi=3, cs=1, rst=0)
        print("")
        print("Place Clone card before reader.")
        print("")
        try:
            while True:
                (stat, tag_type) = rdr.request(rdr.REQIDL)
                if stat == rdr.OK:
                    (stat, raw_uid) = rdr.anticoll()
                    if stat == rdr.OK:
                        print("CARD DETECTED")
                        print(" -  TAG TYPE : 0x%02x" % tag_type)
                        print(" -  UID      : 0x%02x%02x%02x%02x" %
                         raw_uid[0], raw_uid[1], raw_uid[2],raw_uid[3]))
                        print("")
```

```
                        if rdr.select_tag(raw_uid) == rdr.OK:
                            key = [0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF]
                            if  rdr.auth(rdr.AUTHENT1A,  8,  key,  raw_uid)  ==
rdr.OK:
                                stat = rdr.write(8, databytes)
                                rdr.stop_crypto1()
                                if stat == rdr.OK:
                                    print("DATA WRITTEN TO ADDRESS)
                                    return (0)
                                else:
                                    print("FAILED")
                            else:
                                print("AUTH ERR")
                        else:
                            print("Failed to select tag")
        except KeyboardInterrupt:
            print("EXITING PROGRAM.")

if __name__ == "__main__":
    print("Initialising Module=> " + str(uname()[0]))
    CardObject = RFIDCard()
    Secret = CardObject.ReadData()
    print(Secret)
    print(bytes(Secret))
    print("remove the card.")
    sleep(2)
    CardObject.WriteData(bytes(Secret))
```

1.1.7 SimpleRFIDScanner.py

```
import mfrc522
from os import uname

class RFIDCard:
    def __init__(self) -> None:
        self.rdr = mfrc522.MFRC522(sck=2, miso=4, mosi=3, cs=1, rst=0)

    def ConvertToHexString(self, data):
        hexstring = ""
```

```
        for i in data:
            hexstring = hexstring + (hex(i))
        return hexstring

    def AuthCard(self, HSD, FKD):
        FIXHASH = ""
        for i in FKD:
            FIXHASH = FIXHASH + hex(ord(i))

        if HSD == FKD:
            return 1
        return 0

    def ReadData(self):
        print("")
        print("Place the access card before the reader.")
        print("")
        try:
            while True:
                (stat, tag_type) = self.rdr.request(self.rdr.REQIDL)
                if stat == self.rdr.OK:
                    (stat, raw_uid) = self.rdr.anticoll()
                    if stat == self.rdr.OK:
                        print("CARD DETECTED")
                        if self.rdr.select_tag(raw_uid) == self.rdr.OK:
                            key = [0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF]
                            if   self.rdr.auth(self.rdr.AUTHENT1A,   8,   key,
raw_uid) == self.rdr.OK:
                                data = self.rdr.read(8)
                                return (data)
                            else:
                                print("AUTH ERR")
                        else:
                            print("Failed to select tag")

        except KeyboardInterrupt:
            print("EXITING PROGRAM.")
```

```
if __name__ == "__main__":
    CardObject = RFIDCard()
    FIXKEYSTR = "DwMo0G8ImmULsJDe"
    CardData = CardObject.ReadData()
    if       CardObject.AuthCard(CardObject.ConvertToHexString(list(CardData)),
FIXKEYSTR):
        print("AUTH SUCCESS. Access Granted.")
    else:
        print("Access Denied.")
```

5.2 Attack Scenario

The attack scenario is a mock-up of actual security implementation and follows the procedure inspired by real-world RFID attacks and methods.

Following is the attack scenario that we will use to demonstrate this project: -

> *"We can only pass the security system with our 'original' card (My ID card), this system imitates the basic RFID door lock systems that scan the data inside the cards and then grant access if the data is stored in the database with a proper access level. We will try to then clone this data to a blank RFID card and open the door with our new malicious clone card, theoretically, this should work as long as we will be using the same type of RFID card operating in standard frequency and transmitting the same binary data."*

5.3 Experiment/Demonstration

We will be using Raspberry Pi Pico micro-controller (Pico) for this implementation, we will connect Pico with our computer using USB Serial Interface. The main computer has an Intel Core i7-8550U (Kaby Lake-R U4+2) with 20GB DDR4 SDRAM running Microsoft Windows 11 Education (x64) Build 22000.675.

5.3.1 Sample Scanner

SampleRFIDScanner.py is a micro-python code that contains a fixed key that is also stored in the VIT ID[10] card (Original Card/Original ID), when executed it scans for any RFID signal and then checks the data stored in the card against a hardcoded key. In a real-world application, organizations usually fetch keys via a secure database and/or implement a rolling key mechanism, both strategies are out of the scope of this project report.

After a successful key match, we should see the following result on screen in Figure 18.

[10] Vellore Institute of Technology Bhopal campus has not activated RFID based technologies in campus as of 8th May, 2022. Hence using the college ID will not do any damage to university asset. Should we damage the IC inside the card, there will not be any consequences as there is no active use of the digital capabilities of the card so far in/off campus. **The author advises against using an active card which is being used in your workspace's operation without prior permission from supervisors and security department**, any such action might result in financial fine and/or related punishment for damaging assets and/or risking security infrastructure of the workplace.

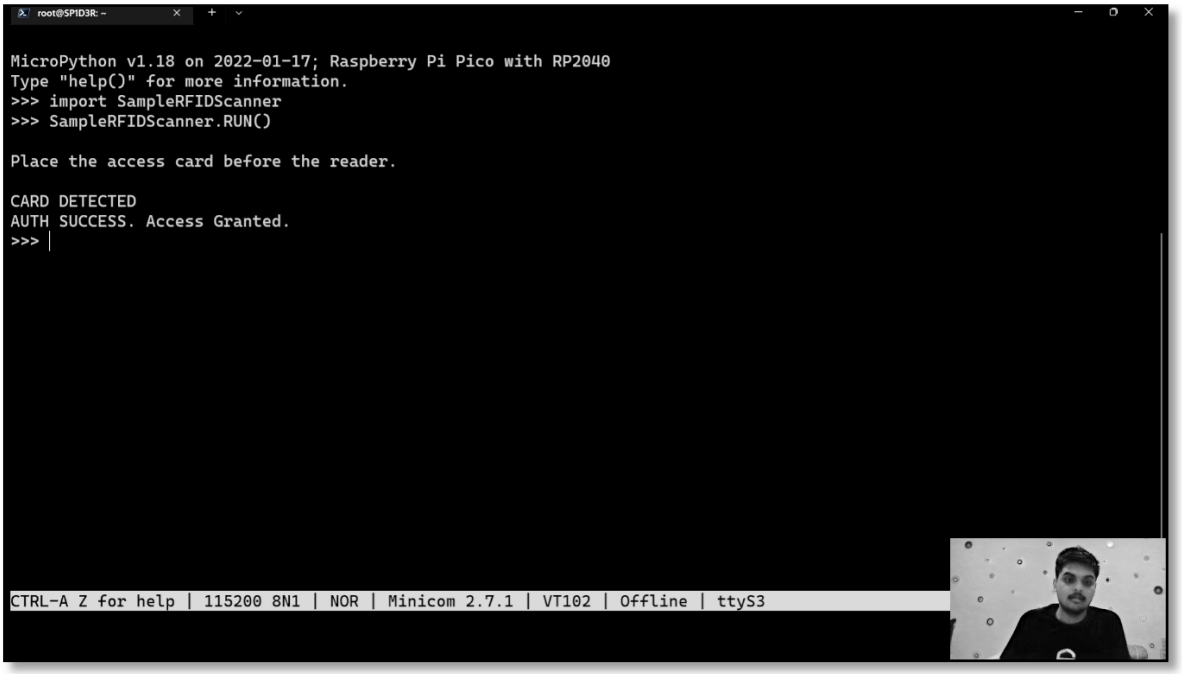

Figure 18: SampleRFIDScanner.py Correct ID card scan result.

When the keys do not match, we should see the second result as shown in Figure 19.

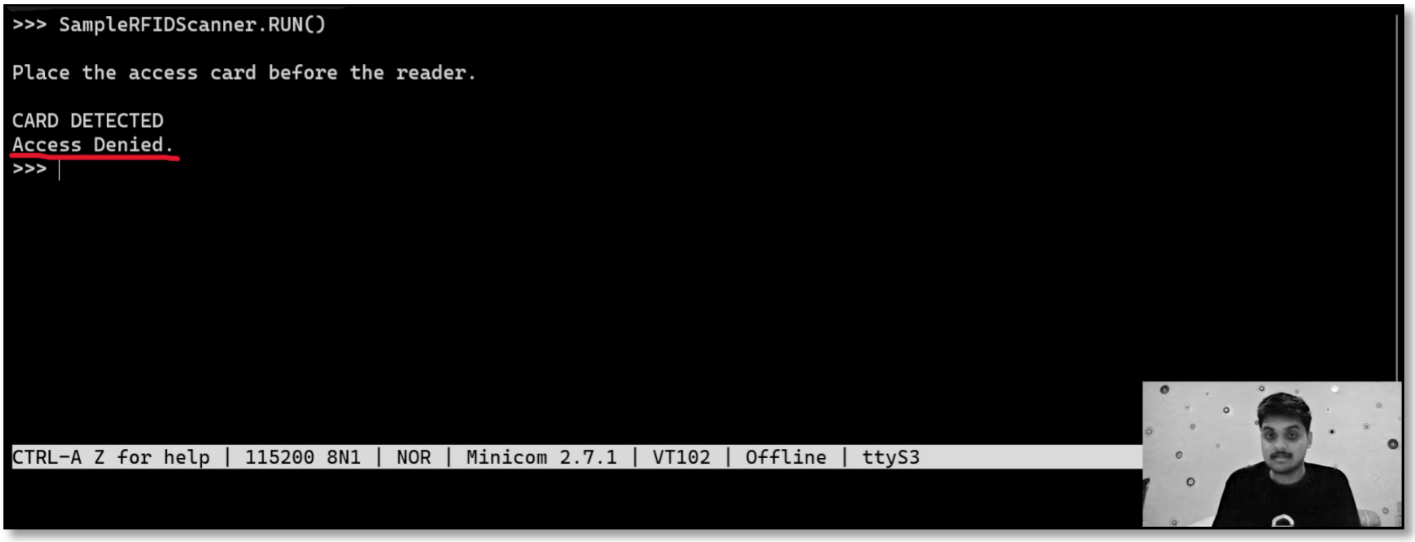

Figure 19: SampleRFIDScanner.py unauthorized access result.

5.3.2 Read/Write to MIFARE 1K Card

The `readRFID.py` and `writeRFID.py` scripts will read from and write into the card respectively.

The program and the used hardware can be used to read/write 16 bytes of data from the card at address 0x08. This data can also be represented in 16 characters where each character uses 1 byte of space in 8-bit ASCII. The KeyGen.py generates random 16byte data consisting of ASCII printable small and capital case letters and numbers, then stores it in a byte array that can be used to write the data in the card via the mfrc522 library. The mfrc522 library is slightly modified to work with Raspberry Pi Pico, adding the configuration for the “rp2” device broadcast.

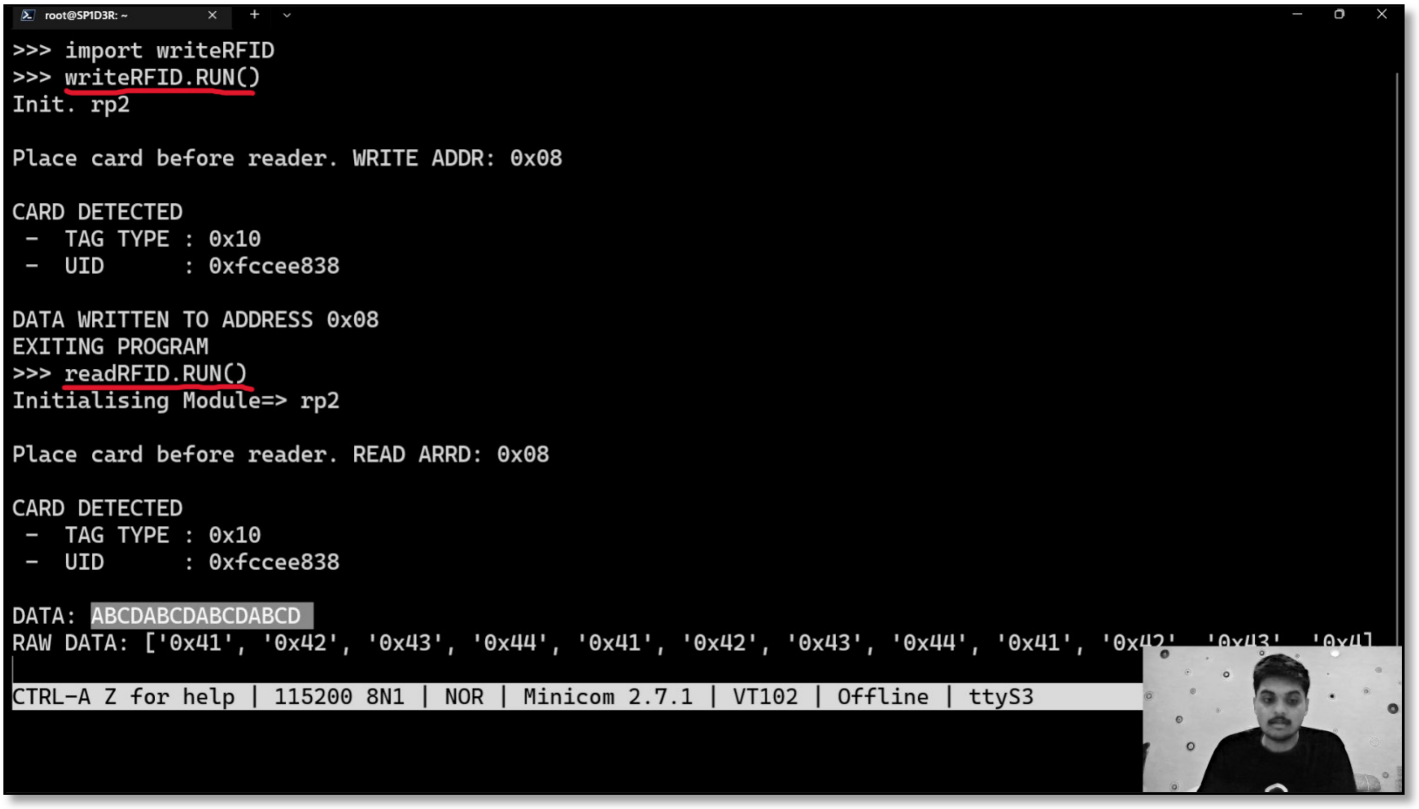

Figure 20: readRFID.py and writeRFID.py examples.

5.3.3 Communicating via Serial Connection

After connecting the microcontroller to the computer, we can find the USB Serial Port number that the device is connected to by using the following set of PowerShell commands-

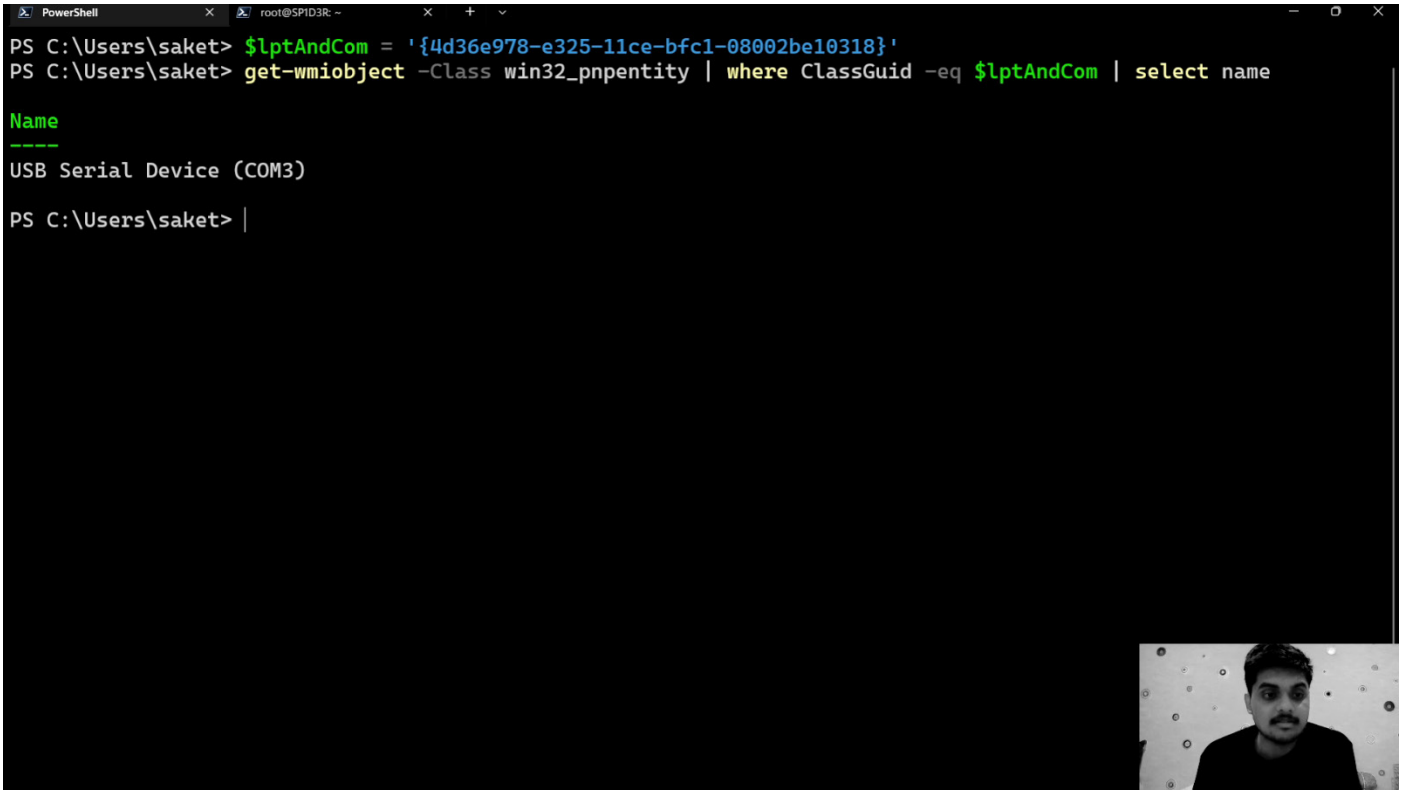

Figure 21: Finding the Serial COM port that Pico connects to via PowerShell

In the case above we can see that the connected port is "`COM3`" which can differ in different cases and might show more than one device if more than one serial connection is established.

While translating this to the WSL instance, COM3 can be accessed via `/dev/ttyS3`. Following the simple template of `COMβ = /dev/ttySβ.`

Microcontroller at COM3 can be accessed via 8bit data stream, no parity bits, and stop code ‘1’ at variable baud rate as the serial port is emulated.

To communicate with micro python, we can use `minicom -D /dev/ttyS3`

```
Welcome to minicom 2.7.1

OPTIONS: I18n
Compiled on Dec 23 2019, 02:06:26.
Port /dev/ttyS3, 20:57:41

Press CTRL-A Z for help on special keys

OK
MPY: soft reboot
raw REPL; CTRL-B to exit
>
MicroPython v1.18 on 2022-01-17; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>
```

Figure 22: Minicom interacting with COM3 Serial, micropython

Then we can import `SampleRFIDScanner.py` and `CloneCardData.py` to run the experiment.

Execute the `SimpleRFIDSamle.RUN()` and bring the original card and it should result in access granted, run it again, and can other tokens that should result in Access Denied message from the program. This establishes the fact that only the original card has the key to trigger unlock mechanism.

To clone the data, import `CloneCardData` and call the `RUN()` function.

Follow the instructions and the data from the original card will be copied to the second blank card/token.

Run the `SampleRFIDScanner.RUN()` and the clone token should be accepted as original.

```
root@SP1D3R: ~

>>> import CloneCardData
>>> CloneCardData.RUN()
Initialising Module=> rp2

Place Original card before reader.

CARD DETECTED
 -  TAG TYPE : 0x10
 -  UID      : 0xd4726a69

[68, 119, 77, 111, 48, 71, 56, 73, 109, 109, 85, 76, 115, 74, 68, 101]
b'DwMo0G8ImmULsJDe'
remove the card.

Place Clone card before reader.

CARD DETECTED
 -  TAG TYPE : 0x10
 -  UID      : 0x238b4715

DATA WRITTEN TO ADDRESS 0x08
>>>

CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | ttyS3
```

Figure 23: Cloning MIFARE Card Data

After the cloning process, the second RFID card should have the same data as in the Original Card.

That should enable us to get “Access Granted” from the `SampleRFIDScanner.RUN()`

CHAPTER 6: CONCLUSIONS AND RECOMMENDATIONS

6.1 Study so far

In this project, we studied current and potential proposed RFID security and authentication protocols and algorithms, we also saw how we can use cheap hardware to clone an RFID card and with some more financial investment we can attack a wider range of RFID cards, this, of course, poses a great risk in using the same technology, without modification, in next-generation authentication systems. But it also opens a wide door for research and development in this field.

We used MIFARE 1K card with default encryption/encoding and cloned the data from an in-production common RFID card to another blank card using minimal and inexpensive hardware under ₹2,000 (approx.) with RaspberryPi Pico Microcontroller and MFRC522 read/write module.

The code presented in this project is not harmful and cannot be utilized successfully to compromise any standard RFID-based system in a real-world application. This is only for educational purposes and to introduce young researchers to the basic low-level working of RFID technology.[11]

6.2 Limitations of this project

The limitation of this work is that we cannot change the UID of the RFID card with available hardware, due to global ship shortage [12][13][14] and nationwide lockdowns and online university operations due to the COVID-19 pandemic, I was not able to get my hands on more powerful RFID equipment. The plan was to be able to clone the complete card, including the card header at address 0x00.

Generally, the UID of a general MIFARE 1K Card is the first 4 bytes in block 0 of sectors 0 and it cannot be modified after being manufactured. But the 1st generation of UID Changeable Card (named Chinese Magic Card) can be changed by an external device, such as PN532, ACR122U, PM3, etc. The device needs to be used with `nfc-mfsetuid` on Linux [48].

6.3 Potential Future Work

Recent RFID authentication protocols are proposed to develop an efficient and secure RFID system. the security requirements of an RFID system must be satisfied, so the system could be able to defend against major attacks such as replay, man-in-the-middle, impersonation, desynchronization, DoS, and more. Since the RFID passive tag has limited resources to compute complex operations, the heavyweight and simple-weight protocols are not feasible for practical implementation. However, lightweight and ultra-lightweight protocols use only simple operations within the tag computation limits and show the lowest tag overhead level. Lightweight and ultra-lightweight protocols are considered the most suitable for the current applications. Another vital aspect when considering the appropriate RFID protocol is the security resistance to the attacks.

Researchers are encouraged to pay attention to the forward and backward compatible security since most protocols do not reflect on these two types of attacks. Finally, maintaining the basic security requirements for an RFID system is required to achieve protection against major attacks.

---

[11] Check Appendix II for complete disclaimer.
[12] https://www.theverge.com/2022/4/29/23049114/chip-shortage-intel-ceo-2024
[13] https://www.wsj.com/articles/global-chip-shortages-latest-worry-too-few-chips-for-chip-making-11651575601
[14] https://www.androidpolice.com/chip-shortage-forcing-bmw-ship-vehicles-without-android-auto-carplay/

REFERENCES

[1] P. Rotter, "A Framework for Assessing RFID System Security and Privacy Risks," *IEEE Pervasive Comput.*, vol. 7, no. 2, pp. 70–77, Apr. 2008, doi: 10.1109/MPRV.2008.22.

[2] G. Avoine, X. Carpent, and J. Hernandez-Castro, "Pitfalls in Ultralightweight Authentication Protocol Designs," *IEEE Trans. Mob. Comput.*, vol. 15, no. 9, pp. 2317–2332, Sep. 2016, doi: 10.1109/TMC.2015.2492553.

[3] Z. Kfir and A. Wool, "Picking Virtual Pockets using Relay Attacks on Contactless Smartcard," in *First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SECURECOMM'05)*, Athens, Greece, 2005, pp. 47–58. doi: 10.1109/SECURECOMM.2005.32.

[4] P. Peris-Lopez, J. C. Hernandez-Castro, J. M. Estevez-Tapiador, and A. Ribagorda, "M2AP: A Minimalist Mutual-Authentication Protocol for Low-Cost RFID Tags," in *Ubiquitous Intelligence and Computing*, Berlin, Heidelberg, 2006, pp. 912–923. doi: 10.1007/11833529_93.

[5] P. Peris-Lopez, J. C. Hernandez-Castro, J. M. E. Tapiador, and A. Ribagorda, "LMAP: A Real Lightweight Mutual Authentication Protocol for Low-cost RFID tags," p. 12.

[6] P. Peris-Lopez, J. C. Hernandez-Castro, J. M. Estevez-Tapiador, and A. Ribagorda, "EMAP: An Efficient Mutual-Authentication Protocol for Low-Cost RFID Tags," in *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*, Berlin, Heidelberg, 2006, pp. 352–361. doi: 10.1007/11915034_59.

[7] H.-Y. Chien, "SASI: A New Ultralightweight RFID Authentication Protocol Providing Strong Authentication and Strong Integrity," *IEEE Trans. Dependable Secure Comput.*, vol. 4, no. 4, pp. 337–340, Oct. 2007, doi: 10.1109/TDSC.2007.70226.

[8] Y. Tian, G. Chen, and J. Li, "A New Ultralightweight RFID Authentication Protocol with Permutation," *IEEE Commun. Lett.*, vol. 16, no. 5, pp. 702–705, May 2012, doi: 10.1109/LCOMM.2012.031212.120237.

[9] T. Li and R. Deng, "Vulnerability Analysis of EMAP-An Efficient RFID Mutual Authentication Protocol," in *The Second International Conference on Availability, Reliability and Security (ARES'07)*, Apr. 2007, pp. 238–245. doi: 10.1109/ARES.2007.159.

[10] T. Li and G. Wang, "Security Analysis of Two Ultra-Lightweight RFID Authentication Protocols," in *New Approaches for Security, Privacy and Trust in Complex Environments*, Boston, MA, 2007, pp. 109–120. doi: 10.1007/978-0-387-72367-9_10.

[11] Q. U. Ain, Y. Mahmood, U. Mujahid, and M. Najam-ul-islam, "Cryptanalysis of mutual ultralightweight authentication protocols: SASI amp; RAPP," in *2014 International Conference on Open Source Systems Technologies*, Dec. 2014, pp. 136–145. doi: 10.1109/ICOSST.2014.7029334.

[12] A. Tewari and B. B. Gupta, "Cryptanalysis of a novel ultra-lightweight mutual authentication protocol for IoT devices using RFID tags," *J. Supercomput.*, vol. 73, no. 3, pp. 1085–1102, Mar. 2017, doi: 10.1007/s11227-016-1849-x.

[13] K.-H. Wang, C.-M. Chen, W. Fang, and T.-Y. Wu, "On the security of a new ultra-lightweight authentication protocol in IoT environment for RFID tags," *J. Supercomput.*, vol. 74, no. 1, pp. 65–70, Jan. 2018, doi: 10.1007/s11227-017-2105-8.

[14] J. H. Khor and M. Sidorov, "Weakness of Ultra-Lightweight Mutual Authentication PRotocol for IoT Devices Using RFlD Tags," in *2018 Eighth International Conference on Information Science and Technology (ICIST)*, Jun. 2018, pp. 91–97. doi: 10.1109/ICIST.2018.8426178.

[15] M. Khalid, U. Mujahid, and M. Najam-ul-Islam, "Cryptanalysis of ultralightweight mutual authentication protocol for radio frequency identification enabled Internet of Things networks:," *Int. J. Distrib. Sens. Netw.*, Aug. 2018, doi: 10.1177/1550147718795120.

[16] S.-C. Huang, C.-W. Tsai, and T. Hwang, "Comment on 'cryptanalysis of a novel ultralightweight mutual authentication protocol for IoT devices using RFID tags,'" in *Proceedings of the 2018 International Conference on Data Science and Information Technology*, New York, NY, USA, Jul. 2018, pp. 23–27. doi: 10.1145/3239283.3239300.

[17] M. Gao and Y. Lu, "URAP: a new ultra-lightweight RFID authentication protocol in passive RFID system," *J. Supercomput.*, vol. 78, no. 8, pp. 10893–10905, May 2022, doi: 10.1007/s11227-021-04252-y.

[18] M. Shariq, K. Singh, P. K. Maurya, A. Ahmadian, and D. Taniar, "AnonSURP: an anonymous and secure ultralightweight RFID protocol for deployment in internet of vehicles systems," *J. Supercomput.*, vol. 78, no. 6, pp. 8577–8602, Apr. 2022, doi: 10.1007/s11227-021-04232-2.

[19] R. Baashirah and A. Abuzneid, "Survey on Prominent RFID Authentication Protocols for Passive Tags," *Sensors*, vol. 18, no. 10, Art. no. 10, Oct. 2018, doi: 10.3390/s18103584.

[20] O. M, "Efficient hash-chain based RFID privacy protection scheme," *Int. Conf. Ubiquitous Comput.-Ubicomp Workshop Priv. Curr. Status Future Dir. 2004*, 2004, Accessed: May 11, 2022. [Online]. Available: https://cir.nii.ac.jp/crid/1571980075673234944

[21] G. Avoine and P. Oechslin, "A scalable and provably secure hash-based RFID protocol," in *Third IEEE International Conference on Pervasive Computing and Communications Workshops*, Mar. 2005, pp. 110–114. doi: 10.1109/PERCOMW.2005.12.

[22] G. Avoine, E. Dysli, and P. Oechslin, "Reducing Time Complexity in RFID Systems," in *Selected Areas in Cryptography*, Berlin, Heidelberg, 2006, pp. 291–306. doi: 10.1007/11693383_20.

[23] D. Molnar, A. Soppera, and D. Wagner, "A Scalable, Delegatable Pseudonym Protocol Enabling Ownership Transfer of RFID Tags," in *Selected Areas in Cryptography*, Berlin, Heidelberg, 2006, pp. 276–290. doi: 10.1007/11693383_19.

[24] T. Van Le, M. Burmester, and B. de Medeiros, "Universally composable and forward-secure RFID authentication and authenticated key exchange," in *Proceedings of the 2nd ACM symposium on Information, computer and communications security*, New York, NY, USA, Mar. 2007, pp. 242–252. doi: 10.1145/1229285.1229319.

[25] E.-K. Ryu and T. Takagi, "A hybrid approach for privacy-preserving RFID tags," *Comput. Stand. Interfaces*, vol. 31, no. 4, pp. 812–815, Jun. 2009, doi: 10.1016/j.csi.2008.09.001.

[26] L. Zheng, Y. Xue, L. Zhang, and R. Zhang, "Mutual Authentication Protocol for RFID Based on ECC," in *2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, Jul. 2017, vol. 2, pp. 320–323. doi: 10.1109/CSE-EUC.2017.245.

[27] M. R. Alagheband and M. R. Aref, "Simulation-Based Traceability Analysis of RFID Authentication Protocols," *Wirel. Pers. Commun.*, vol. 77, no. 2, pp. 1019–1038, Jul. 2014, doi: 10.1007/s11277-013-1552-7.

[28] J. Wang, C. Floerkemeier, and S. E. Sarma, "Session-based security enhancement of RFID systems for emerging open-loop applications," *Pers. Ubiquitous Comput.*, vol. 18, no. 8, pp. 1881–1891, Dec. 2014, doi: 10.1007/s00779-014-0788-x.

[29] E.-K. Ryu, D.-S. Kim, and K.-Y. Yoo, "On Elliptic Curve Based Untraceable RFID Authentication Protocols," in *Proceedings of the 3rd ACM Workshop on Information Hiding and Multimedia Security*, New York, NY, USA, Jun. 2015, pp. 147–153. doi: 10.1145/2756601.2756610.

[30] R. Songhela and M. L. Das, "Yet Another Strong Privacy-Preserving RFID Mutual Authentication Protocol," in *Security, Privacy, and Applied Cryptography Engineering*, Cham, 2014, pp. 171–182. doi: 10.1007/978-3-319-12060-7_12.

[31] J.-S. Chou, "An efficient mutual authentication RFID scheme based on elliptic curve cryptography," *J. Supercomput.*, vol. 70, no. 1, pp. 75–94, Oct. 2014, doi: 10.1007/s11227-013-1073-x.

[32] Q. Yao, J. Ma, S. Cong, X. Li, and J. Li, "Attack gives me power: DoS-defending constant-time privacy-preserving authentication of low-cost devices such as backscattering RFID tags," in *Proceedings of the 3rd ACM Workshop on Mobile Sensing, Computing and Communication*, New York, NY, USA, Jul. 2016, pp. 23–28. doi: 10.1145/2940353.2940361.

[33] M. S. Farash, "Cryptanalysis and improvement of an efficient mutual authentication RFID scheme based on elliptic curve cryptography," *J. Supercomput.*, vol. 70, no. 2, pp. 987–1001, Nov. 2014, doi: 10.1007/s11227-014-1272-0.

[34] Z. Zhang and Q. Qi, "An Efficient RFID Authentication Protocol to Enhance Patient Medication Safety Using Elliptic Curve Cryptography," *J. Med. Syst.*, vol. 38, no. 5, p. 47, Apr. 2014, doi: 10.1007/s10916-014-0047-8.

[35] B.-C. Chen, C.-T. Yang, H.-T. Yeh, and C.-C. Lin, “Mutual Authentication Protocol for Role-Based Access Control Using Mobile RFID,” *Appl. Sci.*, vol. 6, no. 8, Art. no. 8, Aug. 2016, doi: 10.3390/app6080215.

[36] H. Fernando and J. Abawajy, “Mutual Authentication Protocol for Networked RFID Systems,” Nov. 2011, pp. 417–424. doi: 10.1109/TrustCom.2011.54.

[37] X. Chen, T. Cao, and J. Zhai, “Untraceability Analysis of Two RFID Authentication Protocols,” *Chin. J. Electron.*, vol. 25, no. 5, pp. 912–920, Sep. 2016, doi: 10.1049/cje.2016.08.013.

[38] C. Chen, Z. Qian, I. You, J. Hong, and S. Lu, “ACSP: A Novel Security Protocol against Counting Attack for UHF RFID Systems,” in *2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, Jun. 2011, pp. 100–105. doi: 10.1109/IMIS.2011.60.

[39] M. Safkhani, N. Bagheri, and A. Mahani, “On the security of RFID anti-counting security protocol (ACSP),” *J. Comput. Appl. Math.*, vol. 259, pp. 512–521, Mar. 2014, doi: 10.1016/j.cam.2013.06.016.

[40] Y. Li, Y. Cho, N. Um, and S. Lee, “Security and Privacy on Authentication Protocol for Low-cost RFID,” in *2006 International Conference on Computational Intelligence and Security*, Nov. 2006, vol. 2, pp. 1101–1104. doi: 10.1109/ICCIAS.2006.295432.

[41] M. Burmester and J. Munilla, “Lightweight RFID authentication with forward and backward security,” *ACM Trans. Inf. Syst. Secur.*, vol. 14, no. 1, p. 11:1-11:26, Jun. 2011, doi: 10.1145/1952982.1952993.

[42] S. Lee, T. Asano, and K. Kim, “RFID Mutual Authentication Scheme based on Synchronized Secret Information,” p. 6.

[43] Y. Zuo, “Survivability Experiment and Attack Characterization for RFID,” *IEEE Trans. Dependable Secure Comput.*, vol. 9, no. 2, pp. 289–302, Mar. 2012, doi: 10.1109/TDSC.2011.30.

[44] C. C. Chang, W. Y. Chen, and T. F. Cheng, “A Secure RFID Mutual Authentication Protocol Conforming to EPC Class 1 Generation 2 Standard,” in *2014 Tenth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, Aug. 2014, pp. 642–645. doi: 10.1109/IIH-MSP.2014.166.

[45] H. Liu, H. Ning, Y. Zhang, D. He, Q. Xiong, and L. T. Yang, “Grouping-Proofs-Based Authentication Protocol for Distributed RFID Systems,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 7, pp. 1321–1330, Jul. 2013, doi: 10.1109/TPDS.2012.218.

[46] Raspberry Pi Foundation, “Raspberry Pi Pico Python SDK,” p. 47.

[47] S. Upadhyay, *PiPicoRFID*. 2022. Accessed: May 16, 2022. [Online]. Available: https://github.com/Saket-Upadhyay/PiPicoRFID

[48] “Can UID be changed on Mifare 1K Card? | W!K!” https://why.yuyeye.cc/post/can-uid-be-changed-on-mifare-1k-card/ (accessed May 18, 2022).

APPENDIX I

Open Source License for this Project[15]

MIT License

[15] https://github.com/Saket-Upadhyay/PiPicoRFID/blob/main/LICENSE

APPENDIX II

Disclaimer

The ideas and procedures presented in this project are strictly for educational purposes and the author shall not be liable for its use otherwise.

All information in the Code and the Project Report is provided in good faith, however, the author makes no representation or warranty of any kind, express or implied, regarding the accuracy, adequacy, validity, reliability, availability, or completeness of any information in the code. UNDER NO CIRCUMSTANCE shall the author have any liability to you for any loss or damage of any kind incurred as a result of the use of the project or reliance on any information provided on the project report and opensource code. Your use of the project report of "RFID++" and its code and your reliance on any information is solely at your own risk.

All trademarks used or referred to in this report are the property of their respective owners. Except as otherwise noted.