Security Analysis of Mobile Point-of-Sale Terminals

Mahshid Mehr Nezhad*, Elliot Laidlaw*, and Feng Hao*

* Department of Computer Science, University of Warwick, UK
{Mahshid.Mehr-Nezhad,Elliot.Laidlaw,Feng.Hao}@warwick.ac.uk

Abstract. The increasing prevalence of Card Present (CP) transactions has driven the growth of mobile Point-of-Sale (mPoS) terminals. These compact, wireless, and low-cost terminals allow merchants to process transactions conveniently by utilizing a mobile phone. In this paper, we analyze the security implications of mPoS terminals with a focus to study the merchants' mobile phones as a key component in the mPoS ecosystem. Our examination covers the security aspects of the mobile phone's communication with the mPoS terminal and the payment provider server, and also the security risks in the mobile phone application itself. We perform an eavesdropping attack to reveal the cryptographic keys in the BLE (Bluetooth Low Energy) communication between the mPoS terminal and the merchant phone, execute a man-in-the-middle (MITM) attack to tamper with the mPoS terminal messages transmitted between the mPoS terminal and the payment provider server, and reverse engineer the mobile phone application to disable the security features that are controlled by the mobile phone.

Keywords: EMV · Payment Systems · Contactless Payment · mPoS Terminals

1 Introduction

Card Present (CP) transactions, also known as face-to-face (F2F) transactions, are growing in popularity as consumers increasingly use credit and debit cards for purchases [8], in contrast to Card Not Present (CNP) transactions. CP transactions are performed when the card is physically present, typically at a Point-of-Sale (PoS) terminal, while CNP transactions occur when neither the cardholder nor the credit card is physically present at the time of the transaction [28]. The focus of this paper is CP transactions.

Traditionally, PoS terminals have been used to process CP transactions. These terminals are typically large, fixed devices that are found in retail stores and other locations where goods and services are sold. They are connected to a payment processor through a wired or wireless network. However, with the growing demand for more flexible and cost-effective payment solutions, mobile PoS (mPoS) terminals have emerged as an alternative to traditional PoS terminals due to their flexibility and affordability, especially for small businesses. Examples

are Sumup [29], Square [27], and iZettle [14]. These terminals are small, compact, low-cost, wireless and easy to configure, requiring a few simple steps. They are equipped to accept various payment methods such as debit/credit/prepaid cards with magnetic strips or embedded chips, contactless payments through mobile wallets, QR codes, and/or cash and checks [9]. They offer the ability for anyone with a bank account to establish their own payment terminal, mostly without requiring a business account or a fixed contract.

Although they provide convenience for merchants and customers, they raise potential risks that can be exploited for malicious purposes. This can include holding an mPoS terminal near a victim's payment device (credit/debit card or Near Field Communication (NFC)-enabled devices such as smartphones or wearable devices (e,g, smartwatches) without their knowledge, in conjunction with other emulation hardware, to perform malicious activities. An example of such exploitation is using these mPoS terminals in a man-in-the-middle (MITM) attack setup, as shown in [22], which bypasses the lock screen on mobile phones as a method of cardholder verification when a Visa card is utilized on Apple Pay with transit mode enabled. Furthermore, studies conducted by researchers at ETH Zurich have revealed various methods for bypassing the Personal Identification Number (PIN) on contactless cards during transactions above the contactless limit, including PIN bypass on Visa cards [3] and on Mastercard cards [4, 5] by using an mPoS terminal with emulators. Another example is the mPoS-based passive attack (also known as digital pick-pocketing), which effectively combines all the required emulation components in a relay attack in a single mPoS terminal for a fraudulent merchant to perform passive relay attacks in order to steal money from users via contactless transactions without their knowledge [17].

The management of these terminals is usually done with a mobile device such as a mobile phone or tablet which plays a crucial role in various aspects of the transaction process, including the establishment of a Bluetooth connection with the mPoS terminal, the connection to the payment provider server over the internet, and the installation of an application on the device to manage the mPoS terminal. In this paper, the potential security risks and vulnerabilities of mPoS terminals are analyzed with a focus on the involvement of mobile phones in their management, which is owned by the merchant. Specifically, the security aspects of the communication between the mobile phone and the mPoS terminal, the communication between the mobile phone and the payment provider server, and the mobile phone application itself are examined. The security of the Bluetooth Low Energy (BLE) communication between the mobile phone and the mPoS terminal is analyzed, and methods for revealing the cryptographic keys used in this communication are explored. Furthermore, a MITM attack is performed to demonstrate the vulnerability of the communication between the mobile phone and the payment provider server. Additionally, the feasibility of reverse engineering the mobile phone application code is shown, and the modification of the security features of the mPoS terminals controlled by the mobile phone is demonstrated. We summarize our contributions as follows:

- Performing an eavesdropping attack on the BLE communication between the mobile phone and the mPoS terminal to extract the cryptographic keys used for communication;
- Performing a MITM attack between the mobile phone and the payment server to intercept and tamper with the messages to be displayed on the terminal;
- Demonstrating the feasibility of reverse engineering the mobile phone application code and the alteration of the security features of the mPoS terminals that are controlled by the mobile phone.

This paper employs the terms *card reader*, *terminal*, and *mPoS terminal* interchangeably. The rest of the paper is organized as follows. In Section two, we provide the background and the related work on studying the mPoS terminals vulnerabilities. Section three explains encryption security, with a focus on the BLE communication between the mPoS terminal and the mobile phone. Section four explains network security, with a focus on the security vulnerabilities of the HTTP communication between the mobile phone and the payment server. In Section five, we investigate the mobile application installed on the mobile phone and demonstrate the feasibility of bypassing the security features, followed by a discussion in Section six. Finally, we conclude the paper in Section seven.

2 Background and Related Work

The installation of an mPoS terminal requires a series of straightforward steps. These steps include purchasing the device, which can vary in price based on its features (with options starting as low as £19), registering for an online account (usually done via the vendor website), installing the corresponding application on the merchant's mobile phone, pairing the phone with the terminal, and finally, making transactions.

The ecosystem of mPoS terminals and their communication with various entities in transactions are depicted in Fig. 1. The mPoS terminal is operated by a mobile phone, owned by the merchant. The merchant downloads an application on their mobile phone and uses it to connect to the mPoS terminal. This enables the merchant to initiate and request payments. When the payment is sent from the merchant's mobile phone to the mPoS terminal, the user is ready to pay.

As shown in Fig. 1, the user has the option to make a payment transaction through either a contactless or chip-and-PIN method by tapping, inserting, or swiping their payment device against the mPoS terminal (1). The payment is then transmitted from the mPoS terminal to the merchant's mobile phone through Bluetooth communication (2). The transaction information is then transmitted from the merchant's mobile phone to the payment provider server for authorization (3). The payment provider, in turn, communicates with the acquirer bank to verify the transaction details and ensure its security and accuracy (4). The acquirer verifies the authenticity of the customer's payment card and checks the available funds with the payment network (5), which communicates with the card issuer (6). Upon receiving approval from the card issuer,

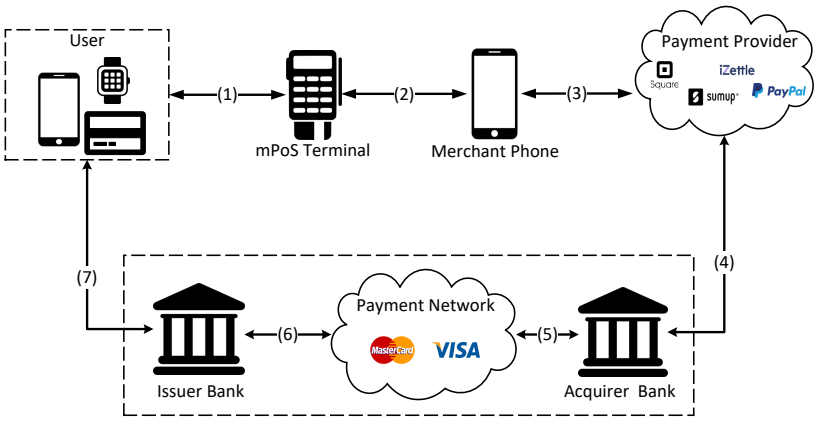

Fig. 1. Mobile Point-of-Sale (mPoS) Terminals Ecosystem

the customer's account is charged, and the customer is notified (7). The merchant's account is credited, and the notification is propagated all the way back to the merchant's mobile phone.

The mPoS terminals have been the subject of security studies in the past decade. One of the first studies, by Frisby et. al. [10] in 2012, investigated the smartphone-based PoS systems that consist of a software application combined with an audio-jack magnetic stripe reader (AMSR) on a smartphone. The study focused on mPoS systems that relied on a smartphone, incorporating an AMSR and a corresponding application running on an Android smartphone. The security assessment concluded that any application running on the smartphone could potentially disable the magnetic stripe reader and obtain confidential cryptographic keys. However, the architecture of mPoS terminals has since evolved, and the current study is not centred around AMSR but shifts the focus from audio-jack magnetic stripe smartphone-based PoS systems to mPoS terminals that are controlled via smartphones.

A subsequent study on mPoS terminals is by Mellen et. al. [18] where they demonstrated potential attack vectors for Square [27] mPoS terminals, both in the software and hardware. In software, their research found security weaknesses in the old Square terminals, which were later deprecated, and discovered vulnerabilities in the encrypted Square reader S4 model and Square registration application, which have since been addressed. In the hardware, the researchers discovered that the Square Reader devices used a chip for point-of-swipe encryption, but were able to bypass the encryption by jumping the connection from the magnetic head reader to the headphone jack input or by crushing the encryption chip. The attack tool, called Swordphish, was developed to record unencrypted swipes and transmit the credit card information to an external server.

In another study published in [15], the security of mPoS terminals, with a specific emphasis on the Miura [30] Shuttle chip-and-PIN reader, was thor-

oughly investigated. The researchers demonstrated the capability of performing arbitrary code execution as a root user on the device, utilizing both the USB and Bluetooth interfaces. Additionally, they exhibited how they could gain root access to the terminal via the chip-and-PIN mode, thereby manipulating the display and keyboard of the device to elicit the entry of the user's PIN, by changing the displayed message to "Try Again" and downgrading to magnetic stripe (magstripe) mode. However, this vulnerability was remediated by 2014.

In 2018, researchers in [11] conducted a follow-up investigation, exploiting a vulnerability that existed at that time through the Bluetooth interface. It was found that the SumUp [29] terminal transmitted commands in plaintext over Bluetooth, thereby allowing for the sending of arbitrary commands and tampering of amounts, following the reverse engineering of the terminal's characteristics and functions. As a result, researchers were able to perform a similar attack vector as outlined in [15], by manipulating the displayed messages to prompt the user to swipe their card, with a message that reads "Please Swipe Card". Our subsequent analysis of transaction data collected from SumUp terminals, however, revealed that the vulnerability had been addressed by the vendor, with the implementation of encryption for all messages. More details will be provided in Section 3. Thus, the demonstrated attack vector is no longer viable, as a successful attacker would require knowledge of the encryption key to send valid messages to the card reader through Bluetooth communication. The researchers also explored the manipulation of amounts in magstripe mode transactions, through the forcing of card swiping. Finally, the study highlights the use of a tamper detection circuit in the tested terminals, which would render the device inoperable in the event of attempted tampering.

Having previously addressed vulnerabilities from various angles on different mPoS terminals, in this paper, we explore the mPoS terminal ecosystem from a novel standpoint, examining the capacity of merchant's mobile phones to initiate attacks as it is a crucial part of the mPoS ecosystem. This study involves a comprehensive analysis of the mobile application and the communication protocols between the mPoS terminal, merchant phone, and payment provider server. The aim of the analysis is to identify and examine security weaknesses at various layers, in order to provide insights into the mitigation of associated risks. The results of our analysis will be presented in the subsequent sections of this paper.

3 Encryption Security

The deployment of an mPoS terminal requires the establishment of a wireless communication channel with the merchant's device, typically a mobile phone which is owned by the merchant. Bluetooth Low Energy (BLE) is a widely used technology for this purpose. The merchant first pairs an mPoS terminal with their mobile phone and uses that established communication link to send and receive transactions to/from the mPoS terminal. However, it is critical to consider the security implications of this communication channel, as exploitation of vulnerabilities can result in extracting the cryptographic keys. As previously

stated, the attack vector described in [11] is no longer viable; our analysis of Bluetooth traffic contradicts the findings in [11], where certain commands sent to the SumUp terminal were discovered in plaintext. Subsequent security improvements made to the SumUp platform have made both packet analysis and arbitrary command execution more challenging since all the packets on the BLE communication are encrypted now.

To carry out the arbitrary command execution attack, an attacker would need knowledge of the encryption key in order to send valid messages to the mPoS terminal through Bluetooth communication. In this section, we first provide background information on BLE communication with a focus on the pairing session and then demonstrate how it is possible to capture the cryptographic keys of the BLE communication by exploiting existing vulnerabilities in the pairing session between the mPoS terminal and the merchant's mobile phone.

3.1 BLE Communication

The BLE protocol stack is comprised of three main architectural layers: the Controller, Host, and Application. The Host Controller Interface (HCI) serves as a bridge between the Host and Controller. The Security Manager Protocol (SMP) located in the Host layer is of particular importance in this context, as it is responsible for establishing secure connections and facilitating secure data exchange between devices. SMP outlines the procedures for pairing, authentication, and encryption of links between devices. During the pairing process, keys are generated for encrypting links and shared through a key distribution protocol for future connections and verification of data. The two devices involved in pairing are differentiated as the initiating device and the responding device. In the context of this paper, the initiating device is the merchant's mobile phone and the responding device is the mPoS terminal.

Based on the BLE specification [26], the SMP carries out pairing in three phases: phase 1, phase 2, and phase 3. In phase 1, the devices engage in a Pairing Feature Exchange using the SMP Pairing Request and Pairing Response commands. During this exchange, information such as Input/Output (I/O) capability, Out-of-Band (OOB) data flags, Bonding flags, MITM protection and Secure Connection (SC) requirements are shared between the devices. The Key Press (KP) flag is only relevant in the Passkey Entry protocol and is ignored in other protocols. Based on this information, both devices determine their I/O capabilities and select the appropriate pairing mechanism for use in the next phase of the pairing process, according to the mapping table specified in the BLE specification.

In phase 2 of the pairing process, the devices utilize the information exchanged in the Pairing Feature Exchange to determine the suitable pairing mechanism, either Low Energy Legacy (LE Legacy) pairing or Secure Connection (SC) pairing. In **LE Legacy** pairing, the devices exchange a Temporary Key (TK) and use it to create a Short Term Key (STK) which is used to encrypt the connection. If the I/O capabilities of a device, either the initiating or responding device, has a display capability, then it will display a randomly generated

passkey value between "000000" and "999999". The other device should have an input capability like a keyboard so a user can input the value displayed for the TK. If the I/O capabilities of both the initiating and responding devices do not have display capabilities but only have a keyboard, the user needs to guarantee that the TKs between the initiating and responding devices are the same. This is a special case for Passkey Entry. After the generation of the TK, it is then combined with two random numbers to produce the STK; *Mrand* for the initiating device, *Srand* for the responding device. The *Mconfirm* and *Sconfirm* are 128-bit confirmation values that can be calculated using the confirm value generation function *c1*. The detail for this function is out of the scope of this research and can be found in Bluetooth Specification [26]. The security of this process depends greatly on the pairing method used to exchange the TK. In Legacy Pairing, the pairing method can be Just Works, Out of Band (OOB), or Passkey. In Just Works, the TK is set to zero. In OOB, the TK is exchanged using a different wireless technology such as NFC. In Passkey, the TK is a 6-digit number that is passed between the devices by the user.

In LE **Secure Connection**, instead of using a TK and STK, LE Secure Connections use a single Long Term Key (LTK) to encrypt the connection. This LTK is generated and exchanged using the Elliptic Curve Diffie Hellman (ECDH) protocol. In addition to supporting the pairing methods in the LE Legacy, it also supports the Numeric Comparison pairing method. It is similar to Just Works but adds another step at the end. Once the devices confirm that the confirmation values match, then both devices will independently generate a final 6-digit confirmation value using nonces. They both then display their calculated values to the user and the user manually checks both values match and confirms the connection.

In phase 3, the devices use the secure communication channel established in the previous phase to share the LTKs which will be used for link encryption. Each LTK is a 128-bit random number that may be generated along with a 16-bit Encrypted Diversifier (EDIV) and 64-bit Random Number (Rand) by both the slave and master device. The exact function of EDIV and Rand keys may vary depending on the implementation of the BLE protocol, but they are typically used to identify or derive the LTK for future connections. In order to conserve energy and storage, the slave device may not retain these values, leaving the responsibility of encrypting future communications to the master device, which in this case is the smartphone.

3.2 Eavesdropping to extract Cryptographic Keys

The attacker, who may be a malicious merchant or an eavesdropper, can extract the cryptographic keys by capturing the pairing session between the mPoS terminal and the merchant's mobile phone. These keys are then used to carry out various attacks. Malicious merchants can capture their phone's pairing session with their terminal during the initial BLE communication setup to obtain the cryptographic keys. These keys can then be utilized to access future transaction data exchanged between the phone and the terminal. An eavesdropper

can also sniff the established BLE communication to compromise the encryption. As demonstrated in [24], the attacker can exploit the vulnerability of the BLE communication by jamming the connection, which forces the master and slave to reconnect and establish a new pairing session. During this process, the eavesdropper can inject appropriate control packets to initiate a key renegotiation to obtain the keys. Our proposed model takes advantage of the vulnerability present in the BLE communication between the merchant's phone and the mPoS terminal without requiring physical access to the mPoS terminal.

Eavesdropping: There are two primary methods for eavesdropping on BLE traffic: using the HCI Snoop Log on the merchant's mobile phone and using over-the-air Bluetooth sniffers. The HCI Snoop Log approach involves capturing and analyzing the HCI data packets on the merchant's Android phone, which can provide detailed information about the BLE communication between the phone and other devices. The over-the-air Bluetooth sniffers, on the other hand, capture BLE communication in the air by using specialized hardware and software. This approach is useful for monitoring and analyzing the Bluetooth traffic between multiple devices over a larger area. Both of these approaches have their own advantages and disadvantages and it depends on the specific requirements of the task and the environment in which it is being performed.

The utilization of HCI snoop logs, which requires the *Developers Options* setting to be enabled on the Android phone, offers several advantages. Firstly, the HCI snoop log is immune to missing packets during the capture process, which is a prevalent issue with over-the-air Bluetooth sniffers. Secondly, as the HCI protocol is situated above the Link Layer (LL) in the Bluetooth protocol stack, the contents of all packets are already decrypted by the LL. This results in a more straightforward analysis of the packets, as they are not impacted by the encryption performed by the LL. However, it has a limitation for some of the mPoS terminals, such as Square [27], that is equipped with the ability to recognize whether Developer Options are enabled on the smartphone, thereby disabling any transactions during this period. As a result, over-the-air Bluetooth sniffers would be a better choice for these mPoS terminals. We used the combination of HCI Snoop Log and Bluefruit BLE sniffer [1] to eavesdrop on the pairing session of the mPoS terminal's BLE communication with an Android phone.

We used Pixel6 as our phone and tested SumUp Air and Square mPoS terminals to capture their pairing session with the phone. The pairing session of the Square [27] terminal is very similar to the SumUp [29] terminal. Hence, for our proof-of-concept, we show the pairing session for a SumUp terminal in Fig. 2, with detailed Pairing Request and Pairing Response shown in Table 1.

Extracting Cryptographic Keys: The pairing request, as depicted in Fig. 2, is initiated by the smartphone and details the desired parameters for the BLE connection. This includes the type of pairing, the I/O capabilities of both devices (the keyboard and display), the request for bonding for future connections, and the demand for a secure connection with MITM protection. The Max Encryption Size field of the request is set to 16, and the Initiator Key

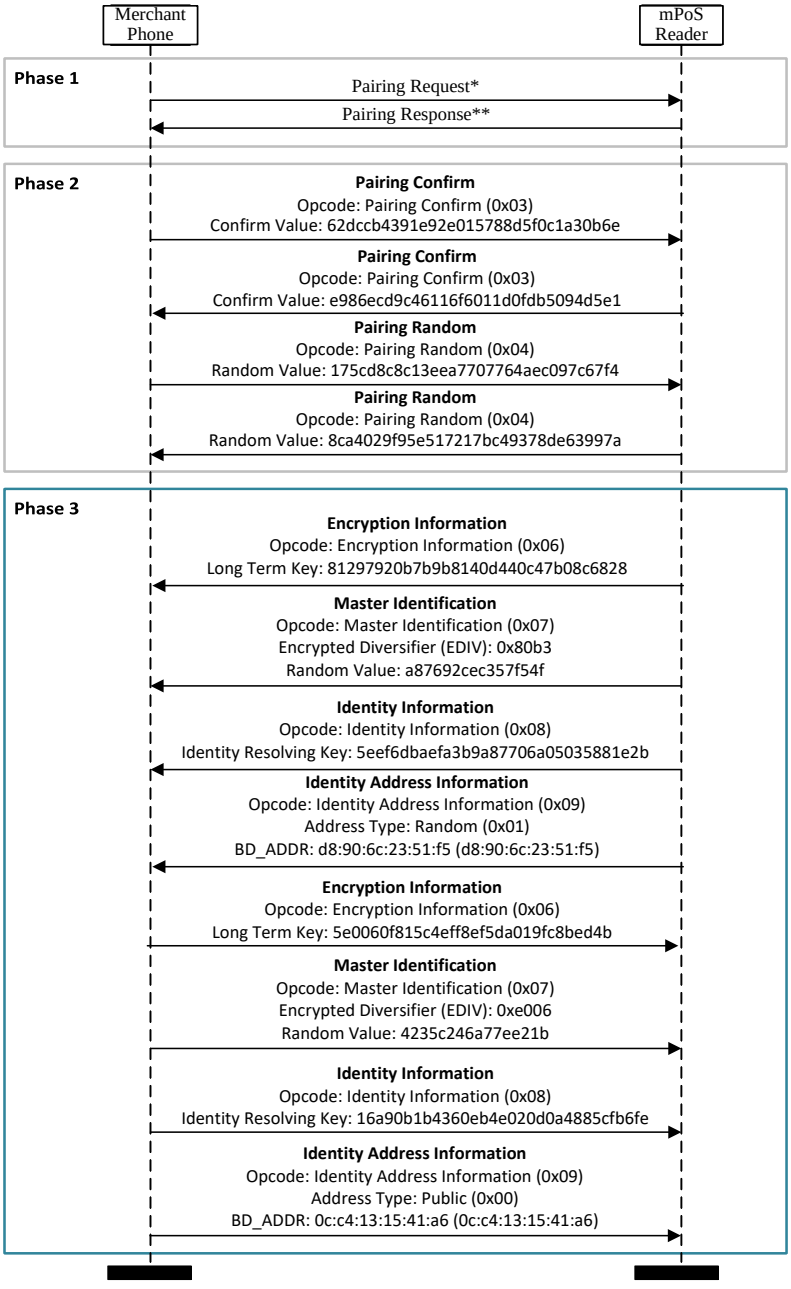

Fig. 2. Pairing Session- SumUp Card Reader

Distribution and Responder Key Distribution fields specify that all of the encryption keys (LTK, Identity Key (IRK), Signature Key (CSRK), and Link Key) should be distributed to both devices. This ensures that both the smartphone and the mPoS terminal have all of the necessary keys for secure and encrypted communication.

Table 1. Pairing Request and Response- SumUp Card Reader

Field	**Pairing Request Value**	**Pairing Request Meaning**	**Pairing Response Value**	**Pairing Response Meaning**
Code	0x01	Pairing Request	0x02	Pairing Response
I/O	0x04	Keyboard/Display	0x03	No I/O
OOB	0x00	NOT Present	0x00	NOT Present
Authentication Request				
Bonding	0x1	Bonding	0x1	Bonding
MITM	1	True	0	False
SC	1	True	0	False
KP	0	False	0	False
Reserved	0x0	-	0x0	-
Max Enc.	16	Max Enc. Size	16	Max Enc. Size
Initiator Key Distribution				
LTK	1	True	1	True
IRK	1	True	1	True
CSRK	1	True	0	False
Link Key	1	True	0	False
Reserved	0x0	-	0x0	-
Responder Key Distribution				
LTK	1	True	1	True
IRK	1	True	1	True
CSRK	1	True	0	False
Link Key	1	True	0	False
Reserved	0x0	-	0x0	-

However, the response from the SumUp card reader to the pairing request is surprising in that it indicates a lack of I/O capabilities despite having both a keyboard and a display. Additionally, the respondent refuses to establish a secure connection and protection against MITM attacks. As a result, **LE Legacy** pairing will be used. The Initiator Key Distribution and Responder Key Distribution fields in the response specify that only the Encryption Key (LTK) and Id Key (IRK) will be shared between the devices, whereas the Signature Key (CSRK) and Link Key will not be exchanged.

It is determined from the mapping of I/O capabilities to the key generation method in the BLE specification (as specified in Table 2.8 of the Bluetooth Core Specification v5.3 [26]) that, given the initiator has a keyboard and display and

the responder claims to have no input or output capabilities, the ***Just Works-Unauthenticated*** key generation method will be employed. The utilization of the Just Works pairing method results in the generation of the TK and STK. The Just Works STK generation method provides no protection against eavesdropping or MITM attacks during the pairing process. Both devices set the TK value utilized in the authentication mechanism to **zero**, leading to a lack of protection against such attacks. The STK is not explicitly shared between the devices, rather the participating devices share random values and calculate the STK individually.

Due to the lack of utilization of the mPoS terminal's keyboard and display for a secure pairing method, the attacker can have access to the distributed keys in phase 3, as shown in Fig. 2. The access to security keys used in a LE Legacy pairing session by an attacker grants them the ability to eavesdrop on the data being transmitted between the two devices. This is because these keys are used to encrypt and secure communication, and having access to them would enable the attacker to decrypt the data and have access to it. For instance, if the attacker possesses the LTK, they could use it to encrypt the data exchanged between the two devices, allowing them to intercept and manipulate the data. Crackle [23] is one of the tools that can be used for this purpose. With the "Decrypt with LTK" feature, crackle uses a user-supplied LTK to decrypt communications between a master and slave.

Not utilizing the I/O capabilities for secure pairing is not common practice across all mPoS terminals. The examination of the SumUp Air mPoS terminal in this study revealed that it does not employ such mechanisms, in contrast to other terminals like iZettle, which do incorporate secure pairing techniques. Specifically, iZettle's method involves the presentation of a numerical value on the terminal's display, which the user must then confirm as matching the corresponding value on their paired device [13].

4 Network Security

The implementation of a mobile application on a smartphone connected to an mPoS terminal requires interaction with servers of the payment service providers through the Internet. In this section, we investigate the analysis of decrypted Hypertext Transfer Protocol Secure (HTTPS) packets and the feasibility of modifying these packets. The subsequent sections present the specifics of our intercepted network traffic, followed by a demonstration of a tampering attack on this traffic, serving as proof of concept for MITM attacks.

4.1 HTTPS Interception

The merchant's mobile phone uses HTTPS packets to communicate with payment providers over the Internet. This protocol employs Transport Layer Security (TLS) to encrypt network traffic. In order to gain access to the contents of these packets, a MITM attack is employed using a proxy server. The proxy

server is able to intercept and decrypt the HTTPS packets, as the smartphone establishes a secure connection with it, believing it to be the intended recipient of the network traffic. The proxy server subsequently forwards the packets to the payment server. Details of communication over the course of a transaction for a SumUp terminal can be seen in Fig. 3. As shown in this figure, a transaction begins with a Checkout Request from the merchant's mobile phone, which requests the appropriate resources to display in the application during the transaction from the payment server. Other information in this request includes the currency, transaction amount, location and mPoS terminal device information, which is sent to the SumUp device for logging and handling purposes. For example, the transaction will fail and the sequence will end if the battery level of the terminal is too low. Continuing from the Checkout Request is a Transaction Request, where the beginning of the transaction is requested from a payment endpoint within SumUp's payment server. This is also the point at which the merchant's mobile phone begins to act as a proxy for communications between the terminal and payment server, which exchange messages without the SumUp application's influence. After this response to the transaction request, we then see four or five request-response pairs to and from the payment endpoint, depending on the payment method (chip-and-PIN or contactless). After successful payment, the transaction ends with a response from the payment endpoint and a value *stop*. The SumUp application processes this action to end the transaction and reject any other responses from the terminal. The transaction officially ends when the merchant phone sends two messages to the terminal on behalf of the payment server, signalling a successful closure of the transaction.

In our attack scenario, the Mitmproxy tool [19] is utilized as the proxy server on a desktop computer to perform a MITM attack between the SumUp application and the payment server. This tool is designed as an interactive, SSL/TLS-capable intercepting proxy for HTTP/1, HTTP/2, and web sockets, as it allows the attacker to monitor, capture and alter connections in real time. On the smartphone, a manual proxy configuration is set up, with the local IPv4 address being used as the server address and 8080 as the port. The Mitmproxy's Certificate Authority (CA) is then installed on the smartphone.

When an application establishes an HTTPS connection, it verifies the legitimacy of the server's certificate through comparison with the trusted system certificate authorities listed in the Android operating system. The list of CA is fixed and secure, but some applications may choose to implement their own custom certificate validation process, known as "Certificate Pinning". We bypass this process by using the Apk-mitm [20] tool. This is accomplished through the application of a series of steps, including 1) decoding the APK file with Apktool (more details in Section 5), 2) replacing the application's network security configuration to allow user-added certificates, 3) modifying the source code to disable various certificate pinning implementations, fourth, encoding the patched APK file with Apktool, and finally, 4) signing the patched APK file with Uber-apk-signer [21]. The application of the Apk-mitm to the extracted SumUp APK file results in the creation of a modified version of the app. This modified app

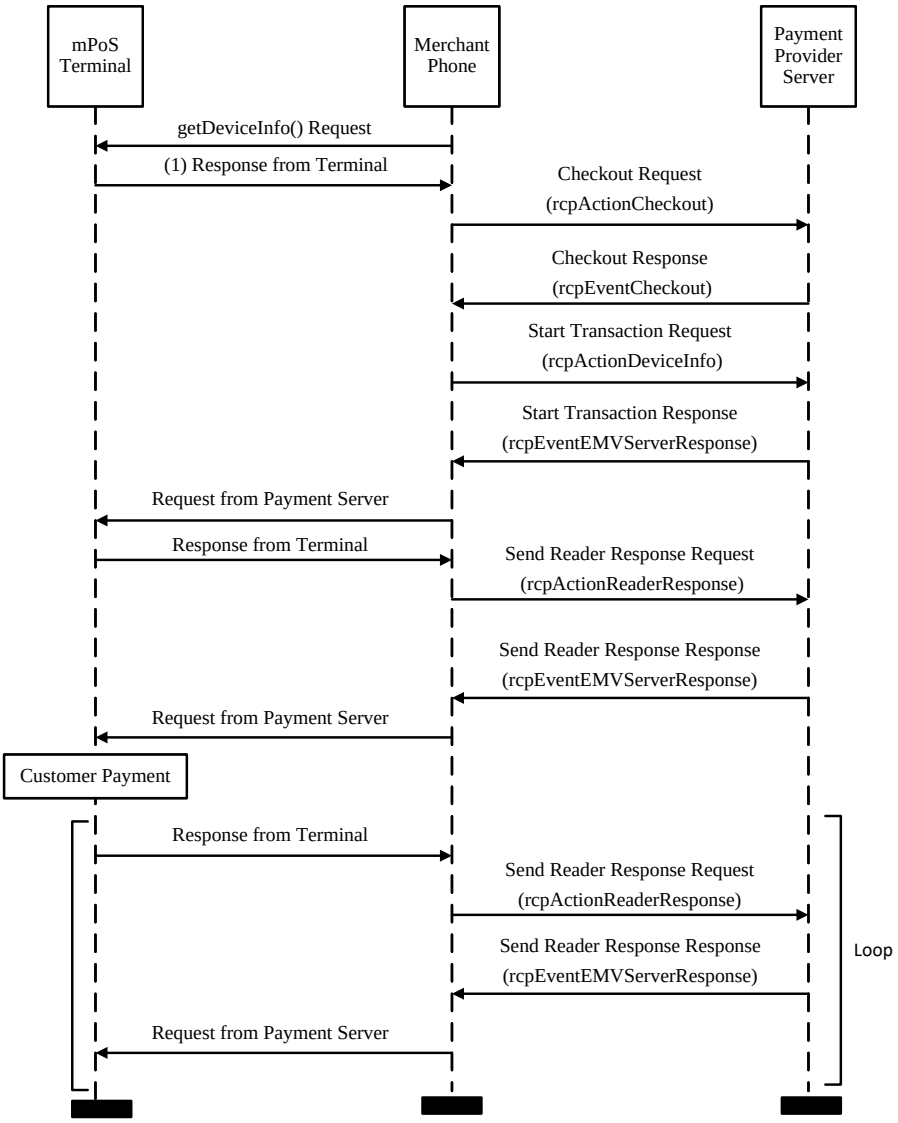

Fig. 3. Sequence Diagram of the Exchanged Messages

now trusts the Mitmproxy certificate, which is added to Android's built-in list of trusted system certificate authorities, allowing for the interception of traffic sent to SumUp's payment provider servers.

4.2 Tampering Attack

In this proof-of-concept demonstration, we present a tampering attack that highlights the feasibility of data modification. In this scenario, a MITM attack is utilized to intercept and manipulate the communication transmitted during a transaction.

By tampering with the messages sent by the payment server for the terminal, we can change the behaviour of the terminal for fraudulent purposes. The messages from the payment server are commands that tell the terminal what to do next to proceed with a transaction. Aside from the messages that we see in network traffic analysis, there are two commands exposed in the application source code, as can be seen in Table 2. The PINPLUS SHOW DEFAULT MESSAGE command is used to show a default message of "SumUp PIN+" on the terminal's display. If we decode the command into hexadecimal, the command contains this string in plaintext ASCII. This means that we can insert arbitrary ASCII into this command to display arbitrary text on the terminal's display.

Table 2. Exposed Commands in SumUp Application Source Code

Command Name	**Base64-Encoded Command**
PINPLUS DEVICE POWER OFF COMMAND	AAIBAQ4=
PINPLUS SHOW DEFAULT MESSAGE	ABUBAQsAAAABAAtTdW1VcCBQSU4rAP8A

However, there are limitations to this attack. Protected messages cannot be altered, as the terminal will reject them, resulting in an error message. Additionally, unprotected messages are not accepted by the terminal during protected message exchange. This presents a problem as modification and sending of commands are desired during a transaction, which largely involves protected message exchanges. The "leave_protected_session" command, which is sent in response to the payment server during a protected message exchange, provides a solution. Tracing its usage in the source code as shown in Fig. 4, reveals its sole purpose is to end a protected message exchange in case of errors. This allows us to propose an attack on the SumUp terminal by exploiting the ability to exit a protected message exchange at any point during a transaction.

The ability to leave a protected message exchange at any point in a transaction allows us to propose an attack on the SumUp terminal. At the end of a normal transaction, the payment server will send two commands to the terminal to inform it that the transaction was successful. In our attack, we replace these

```
@Override
public void onError(i.t.n.a.c.b bar, @Nullable List<j> list, h
    hVar) {
    String str = "onError event received. error code: " + hVar;
    if ((hVar == i.t.n.a.d.b.NOT_ALLOWED ||
    hVar == i.t.n.a.d.b. INVALID_SEQUENCE_NUMBER_IN_PROTECTED
        _MODE  && ReaderCoreManager
    ReaderCoreManager.this.leave_Protected_Mode();
    }

    else {
        WReaderModuleCoreState.getBus().m(new
            CardReaderErrorEvent(bar, ReaderCoreManager.this.
            isReadyToTransmit(), list));
    }
}
```

Fig. 4. Usage of Leaving a Protected Session in the SumUp's Application Source Code

two commands to trick the terminal into displaying that the payment method was declined. First, we use the "leave_protected_session" command sent earlier in the transaction to exit the protected message exchange, allowing us to send an unprotected command. This is followed by the PINPLUS SHOW DEFAULT MESSAGE command that has been modified to display the text "Declined" on the terminal's display. The result of this attack is a successful transaction with the terminal displaying that the transaction was not successful. This is shown in Fig. 5. This vulnerability could be part of a social engineering attack and multiple transactions could be carried out.

5 Software Security

The security of mPoS terminals can be analyzed through the reverse engineering of their code. Reverse engineering refers to the systematic examination of the code of a software program to comprehend its functioning, identify its vulnerabilities, and potentially modify it. In this section, we demonstrate the viability of reverse engineering the code of mPoS terminals mobile applications. In particular, we employ an Android smartphone to analyze the source code and demonstrate the capability of modifying the behaviour of the mPoS terminal through the alteration of the mobile application code. In our case study, we use the SumUp Air mPoS terminal and the Android mobile application. To this end, we outline the procedures involved in the reverse engineering process and present the results of our case study. Our findings underscore the significance of adopting secure code development and deployment practices for mPoS technology to prevent potential security threats.

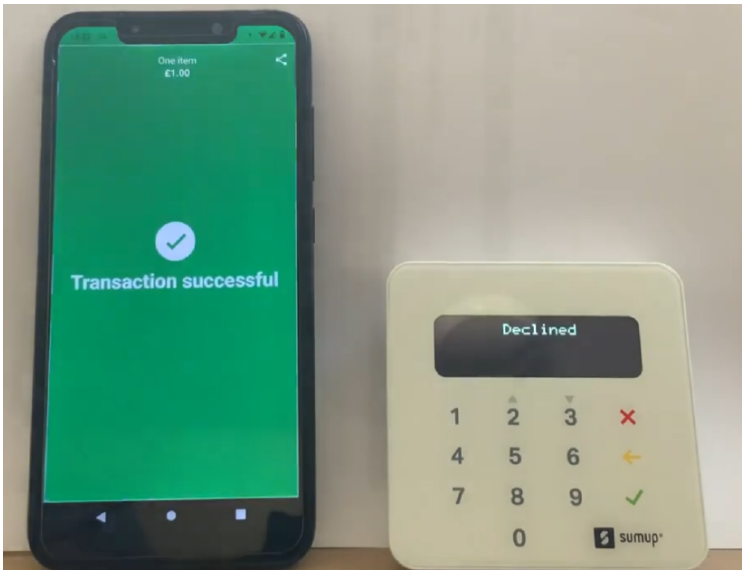

Fig. 5. Tampering Attack on Transaction Messages

5.1 Reverse Engineering

The Android applications are primarily written in Java and are stored as Android packages in the Android Package Kit (APK) file format, which is essentially zip files that encompass resources and assembled Java code. The process of reverse engineering the APK files on Android phones includes several steps: decompiling, making modifications, re-compiling, and signing the APK to be used on Android phones. We use the APK of the SumUp application and decompiled using two methods, Apktool [20] and a standard Java decompiler [7]. The first tool produces Smali code, while the second produces Java code. We use two different tools as they are complementary. Smali code is more difficult to read, therefore we use Java code to understand the application code and identify the vulnerable parts that can be exploited, apply the changes in the relevant part of the Smali code and use it to rebuild and sign the code. To do this, we reverse the decompiling process by rebuilding and signing the APK. The APK was rebuilt using Apk-mitm [25], which uses Apktool to encode the patched APK file and the Uber-apk-signer [21] tool to sign and verify the APK.

5.2 Software Modification Attack

As outlined in Section 4.1, modification of the code can circumvent the Certificate Pinning mechanism, thereby allowing the attacker to execute MITM and tampering attacks on the communication between the merchant's mobile phone and the server of the service provider. Here, we demonstrate another software

modification attack, showcasing how this vulnerability can be exploited to neutralize an additional security feature: *beep sound*.

The process of performing a contactless payment on an mPoS terminal is often accompanied by an audible beep sound as a security feature, which alerts the user to the transaction taking place. This serves as a notification to the user regarding the ongoing transaction and is essential in the prevention of relay attacks. However, a study of the SumUp Air card reader application showed that it is possible to compromise this security feature through modification of the app software.

The analysis of the code revealed that the volume of the beep sound is controlled by the *PlaySoundEffect* method within the *AudioManagers* class. By modifying this method, it is possible to completely control the sound and disable this security feature. In addition, the keyboard input sound made by the SumUp app can also be muted through modification of the code. This involved removing all function declarations and calls related to the *PlaySoundEffect* method from the code base. The recompilation and installation of the modified application showed that the sound is no longer played when keyboard inputs are used during the charge creation process. This highlights the vulnerability of the application to modification and raises concerns about the potential for malicious actors to manipulate the app and compromise the security protocols designed to protect customers. This finding underscores the importance of employing more secure solutions to ensure the safety of user transactions. Relying solely on an audible beep sound as a security feature is insufficient and poses a significant risk to users.

6 Discussion

6.1 Ethical Disclosures

The present study was performed within a controlled setting. The authors purchased commercially available mPoS terminals and used their own bank accounts to demonstrate the proof-of-concept attacks. Our research primarily focused on the SumUp Air mPoS terminal. We have shared our findings with the vendor for their review and feedback. We are currently in discussions with them to further address these issues.

6.2 Mitigating the Vulnerabilities

During our study, we have identified possible solutions for the security issues of mPoS terminals. These solutions include secure pairing methods for encryption security, code obscuring [32], anti-tampering (AT) [6], and abuse detection [2] techniques for traffic security and application code protection. In future research, we plan to study these potential solutions further and evaluate the feasibility and effectiveness of these countermeasures in addressing the identified security issues.

6.3 Tap-to-Phone Technology

The next generation of acceptance terminals, like Tap-to-Phone [31] (also known as Tap-to-Pay [16]), offers potential solutions to the security risks and vulnerabilities associated with mPoS terminals. This technology utilizes Near Field Communication (NFC), allowing merchants to accept contactless payments through their mobile devices. On the other hand, contactless payments have seen a significant increase in popularity in the UK, accounting for over a quarter of all payments made, with mobile payments playing a significant role in this growth. The trend towards contactless payment methods continues to grow, as the spending limit in the UK has increased progressively over the years, reaching £100 in 2021 [12]. Tap-to-Phone technology provides a more convenient and cost-effective solution to accepting these increasingly contactless payments without the need for a dedicated mPoS terminal. However, new systems are still susceptible to security risks, which require further research.

7 Conclusion

This paper analyzes the security implications of mobile Point-of-Sale (mPoS) terminals and their relationship with merchant's mobile phones as a key component of the mPoS system. The security aspects of communication between the (merchant's) mobile phone and the mPoS terminal, the mobile phone and the payment server, and also the security risks in the mobile phone application itself are examined. An eavesdropping attack is performed to reveal cryptographic keys in the BLE communication, a man-in-the-middle (MITM) attack is performed to tamper with mPoS terminal messages, and the mobile phone application is reverse engineered to alter the security features of the mPoS terminals controlled by the mobile phone.

Future research directions for this study include examining other mPoS terminals for their security vulnerabilities and investigating potential solutions to the attacks and vulnerabilities identified in this study. These steps will contribute to a more comprehensive understanding of the security landscape of mPoS terminals and aid in the development of effective security measures to mitigate the risks.

Acknowledgements

The third author is supported by Royal Society (ICA\R1\180226) and EPSRC (EP/T014784/1).

References

1. Adafruit. Adafruit bluefruit ble sniffer. Available at `https://www.adafruit.com/product/2269`. Accessed 10 May 2022.

2. Android. Safetynet attestation api. Available at https://developer.android.com/training/safetynet/attestation. Accessed 12 March 2023.
3. D. Basin, R. Sasse, and J. Toro-Pozo. The emv standard: Break, fix, verify. In *2021 2021 IEEE Symposium on Security and Privacy (SP)*, pages 1766–1781, Los Alamitos, CA, USA, may 2021. IEEE Computer Society.
4. David Basin, Ralf Sasse, and Jorge Toro-Pozo. Card brand mixup attack: Bypassing the PIN in non-visa cards by using them for visa transactions. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 179–194. USENIX Association, August 2021.
5. David Basin, Patrick Schaller, and Jorge Toro-Pozo. Inducing authentication failures to bypass credit card pins. *32rd USENIX Security Symposium (USENIX Security*, 2023.
6. Stefano Berlato and Mariano Ceccato. A large-scale study on the adoption of anti-debugging and anti-tampering protections in android apps. *Journal of Information Security and Applications*, 52:102463, 2020.
7. Java Decompiler. Java online decompiler. Available at http://www.javadecompilers.com/apk. Accessed 13 May 2022.
8. EMVCo. Worldwide emv deployment statistics. Available at https://www.emvco.com/about-us/worldwide-emv-deployment-statistics/. Accessed 11 January 2023.
9. Forbes. What is pos and how does it work? Available at https://www.forbes.com/advisor/in/banking/what-is-pos-and-how-does-it-work/. Accessed 11 January 2023.
10. WesLee Frisby, Benjamin Moench, Benjamin Recht, and Thomas Ristenpart. Security analysis of smartphone point-of-sale systems. In *WOOT*, pages 22–33, 2012.
11. Leigh-Anne Galloway and Tim Yunusov. For the love of money: Finding and exploiting vulnerabilities in mobile point of sales systems. Available at https://leigh-annegalloway.com/for-the-love-of-money/. Accessed 11 January 2023.
12. United Kingdom Government. 2021 budget plan. Available at https://www.gov.uk/government/publications/budget-2021-documents. Accessed 01 June 2021.
13. iZettle. In-app pairing guide. Available at https://developer.zettle.com/docs/ios-sdk/user-guides/manage-in-app-pairing. Accessed 12 March 2023.
14. iZettle. izettle card reader. Available at https://www.izettle.com/. Accessed 11 January 2023.
15. MWR Labs. Mission mpossible: Mobile card payment security. Available at https://www.youtube.com/watch?v=iwOP1hoVJEE. Accessed 11 January 2023.
16. Mastercard. Mastercard tap to pay on iphone. Available at https://partner.visa.com/site/programs/visa-ready/tap-to-phone.html. Accessed 11 January 2023.
17. Mahshid Mehr Nezhad and Feng Hao. Opay: an orientation-based contactless payment solution against passive attacks. In *Annual Computer Security Applications Conference*, pages 375–384, 2021.
18. Alexandrea Mellen, John Moore, and Artem Losev. Mobile point of scam: Attacking the square reader. *Black Hat USA*, 2015.
19. Mitmproxy. How mitmproxy works. Available at https://docs.mitmproxy.org/stable/concepts-howmitmproxyworks/. Accessed 11 January 2023.

20. Patrickfav. Apk tool- a tool for reverse engineering android apk files. Available at https://ibotpeaches.github.io/Apktool/. Accessed 13 May 2022.
21. Patrickfav. Uber apk signer. Available at https://github.com/patrickfav/uber-apk-signer. Accessed 13 May 2022.
22. Andreea-Ina Radu, Tom Chothia, Christopher J.P. Newton, Ioana Boureanu, and Liqun Chen. Practical emv relay protection. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1737–1756, 2022.
23. Mike Ryan. Crackle. Available at https://github.com/mikeryan/crackle. Accessed 24 May 2022.
24. Mike Ryan. Bluetooth: With low energy comes low security. In *7th {USENIX} Workshop on Offensive Technologies ({WOOT} 13)*, 2013.
25. shroudedcode. apk-mitm. Available at https://github.com/shroudedcode/apk-mitm. Accessed 13 May 2022.
26. Bluetooth SIG. Bluetooth core specification, v5.2. Available at https://www.bluetooth.com/specifications/specs/core-specification-5-2/. Accessed 9 May 2022.
27. Square. Square card reader. Available at https://squareup.com/gb/en. Accessed 11 January 2023.
28. Square. What is a card-not-present (cnp) transaction and why does it cost more. Available at https://squareup.com/gb/en/townsquare/what-is-a-card-not-present-transaction. Accessed 11 January 2023.
29. Sumup. Sumup card reader. Available at https://www.sumup.com/en-gb/. Accessed 11 January 2023.
30. Miura Systems. Miura card reader. Available at https://www.miurasystems.com/. Accessed 11 January 2023.
31. Visa. Visa tap to phone. Available at https://partner.visa.com/site/programs/visa-ready/tap-to-phone.html. Accessed 11 January 2023.
32. Dominik Wermke, Nicolas Huaman, Yasemin Acar, Bradley Reaves, Patrick Traynor, and Sascha Fahl. A large scale investigation of obfuscation use in google play. *Proceedings of the 34th annual computer security applications conference*, pages 222–235, 2018.

8 Appendix

Table 3: List of Acronyms

Acronym	**Stands For**	**Description**
APK	Android Package Kit	The file format for applications used on the Android operating system (OS).
AT	Anti Tampering	A security approach that hampers or prevents the reverse engineering or modification of the software or application.

Continued on next page

Table 3 – continued from previous page

Acronym	Stands For	Description
AMSR	Audio-jack Magnetic Stripe Reader	A device that plugs into the audio jack of a smartphone or tablet and reads the magnetic stripe on a credit or debit card for mobile payment processing.
BLE	Bluetooth Low Energy	A power-efficient variant of the classic Bluetooth technology, used for connecting and exchanging data between devices over short distances.
CA	Certificate Authority	An entity that stores, signs, and issues digital certificates.
CNP	Card Not Present	A payment term for transactions where the cardholder does not physically present the card to the merchant (like online purchases).
CP	Card Present	A payment term for transactions where the card is physically swiped, inserted, or tapped at a payment terminal.
CSRK	Signature Key	Encryption Key used in BLE Protocol.
ECDH	Elliptic Curve Diffie Hellman	A key agreement protocol that allows two parties, each having an elliptic-curve public–private key pair, to establish a shared secret over an insecure channel.
EDIV	Encrypted Diversifier	A 16-bit stored value used to identify the LTK distributed during LE legacy pairing.
F2F	Face to Face	A payment term for transactions where the payment device is physically present.
HCI	Host Controller Interface	A standardized communication interface in BLE that provides a layer for transmitting and receiving data between the host and the controller.
HTTP	Hypertext Transfer Protocol	The secure version of HTTP.
HTTPS	Hypertext Transfer Protocol Secure	A protocol used for communication between a web server and a client.
I/O	Input/Output	The capabilities of the devices to enter (input) or display (output) information.
IRK	Identity Key	Encryption Key used in BLE Protocol.
KP	Key Press	The notifications sent between devices to indicate when a key on one device is pressed during the passkey entry pairing method.
LE Legacy	Low Energy Legacy	A method of pairing devices in Bluetooth Low Energy (BLE) prior to the introduction of Secure Connections, which provides a lower level of security compared to Secure Connections.

Continued on next page

Table 3 – continued from previous page

Acronym	**Stands For**	**Description**
LL	Link Layer	A layer in Bluetooth protocol stack responsible for managing the connection and communication between Bluetooth devices.
LTK	Long Term Key	Encryption Key used in BLE Protocol.
MITM	Man-in-the-middle	A type of cyber attack where a malicious actor intercepts and possibly alters the communication between two parties without their knowledge.
mPoS	Mobile Point-of-Sale	Similar to PoS, but smaller compact PoS terminals that are portable and are usually managed by a smartphone (merchant's phone).
NFC	Near Field Communication	A wireless communication technology allowing data exchange between devices in close proximity.
OOB	Out-of-band	A method for sharing pairing information using an external channel, separate from the standard BLE channel.
PIN	Personal Identification Number	A numerical code used in payment cards providing a layer of security by verifying the user's identity.
PoS	Point-of-Sale	A device used by merchants to accept card payments.
SMP	Security Manager Protocol	The protocol responsible for pairing and key distribution between devices.
SC	Secure Connection	A protocol that authenticates two Bluetooth devices and derives a shared secret key between them.
STK	Short Term Key	Encryption Key used in BLE Protocol.
TK	Temporary Key	Encryption Key used in BLE Protocol.
TLS	Transport Layer Security	A cryptographic protocol that provides secure communication between devices on Internet communications.