Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Information Security and Communication Technology

Øystein Løkken Sigholt

Keeping Connected When the Mobile Social Network Goes Offline

Master's thesis in Communication Technology
Supervisor: Besmir Tola
June 2019

NTNU
Norwegian University of Science and Technology

Øystein Løkken Sigholt

Keeping Connected When the Mobile Social Network Goes Offline

Master's thesis in Communication Technology
Supervisor: Besmir Tola
June 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

Title: Keeping Connected When the Mobile Social Network Goes Offline

Student: Øystein Løkken Sigholt

Problem description:

Smartphone users today are used to being connected at all times. At the same time, there are vast amounts of remote areas that are without cell coverage. In addition, natural disasters may cause Internet connectivity outages. As a user moves out off cellular coverage, or is unable to reach the central server for any reason, connectivity is lost. As a consequence, Internet-based mobile social applications will not be able to provide their services and the end-users will become isolated until Internet connectivity is restored, even if the users that are exploiting the service are within a few meters of each other.

Modern smartphones are equipped with a number of radio interfaces that enable wireless communication among devices in close proximity. Leveraging these capabilities could be used to establish connectivity between neighboring devices even in the most remote out-of-coverage locations.

Important challenges concerning this approach are the security issues that arise when the centralized mutually trusted entity becomes unavailable. How can end users make sure that they are not communicating with an impostor, and how to protect the confidentiality and integrity of each message?

The goal of this work is to propose, implement and experimentally validate a simple system that can be used to securely and easily enable communication in existing social applications in situations with no Internet connectivity. The proposed system will use WiFi Direct to connect neighboring devices, and apply mutual TLS to facilitate fast, reliable and secure identification of connected parties based on identity certificates issued by the central server before connection loss.

Responsible professor: Yuming Jiang, IIK

Supervisor: Besmir Tola, IIK

Abstract

The objective of this project is to propose, implement and test a technique that social smartphone applications can use to keep their users connected in situations where they are unable to connect to the application server, but close enough to each other to support device-to-device communication.

The proposed idea and key contribution of this project are as follows: Each client is issued a certificate by a central authorization entity that they can use to identify themselves at any time. If a client is unable to connect to the application server, it will start looking for nearby peers using WiFi Direct, and attempt to form a group with them. Once a WiFi Direct group is formed, the group owner will temporarily assume the role of application server. Each group member and the group owner will then verify each others identity and connect using mutual Transport Layer Security (mTLS), facilitating secure communication.

Sammendrag

Dette prosjektets mål er å foreslå, implementere og teste en teknikk som sosiale smarttelefonapplikasjoner kan bruke for å koble sammen brukere i situasjoner der de ikke er i stand til å nå applikasjonsserveren, men nær nok hverandre til å støtte peer-til-peer-kommunikasjon.

Den foreslåtte ideen og hovedbidraget fra prosjektet er som følger: Hver bruker får utstedt et sertifikat fra en sentral autorisasjonsenhet som de kan bruke til å identifisere seg. Hvis en klient ikke er i stand til å nå applikasjonsserveren, vil den begynne å lete etter andre enheter i samme situasjon i nærheten ved å bruke WiFi Direct og forsøke å danne en gruppe med dem. Når en slik gruppe har blitt dannet, vil gruppeeieren midlertidig overta rollen som applikasjonsserver. Hvert gruppemedlem og gruppens eier vil så verifisere hverandres identitet og koble til hverandre ved hjelp av gjensidig Transport Layer Security (mTLS), og dermed muliggjøre sikker kommunikasjon.

Preface

This thesis is being submitted as part of a master's degree at the Department of Information Security and Communication Technology at the Norwegian University of Science and Technology (NTNU).

As a result of the current work, a scientific paper has been produced which is soon to be submitted to WiMob 2019, The 15th International Conference on Wireless and Mobile Computing, Networking and Communications.

I would like to thank my supervisor Besmir Tola for his invaluable advice and continuous feedback, and professor Yuming Jiang for his encouragement and guidance.

Trondheim, Thursday 6th June, 2019
Øystein Løkken Sigholt

Contents

List of Figures

List of Tables

List of Acronyms

ADB Android Debug Bridge.

AES Advanced Encryption Standard.

AP Access Point.

API Application Programming Interface.

ASN.1 Abstract Syntax Notation One.

BLE Bluetooth Low Energy.

BSS Basic Service Set.

CA Certificate Authority.

CBC Cipher Block Chaining.

CDF Cumulative Distribution Function.

CRL Certificate Revocation List.

CSR Certificate Signing Request.

DH Diffie-Hellman.

DN Distinguished Name.

DNS Domain Name System.

DoS Denial of Service.

DSA Digital Signature Algorithm.

E2E End-to-End.

EAP-TLS Extensible Authentication Protocol Transport Layer Security.

EC Elliptic Curve.

ECC Elliptic Curve Cryptography.

ECDLP Elliptic Curve Discrete Logarithm Problem.

ECDSA Elliptic Curve Digital Signature Algorithm.

ECIES Elliptic Curve Integrated Encryption Scheme.

EU European Union.

GAS Generic Advertisement Service.

GB Gigabyte.

GM Group Member.

GO Group Owner.

HMAC Hash-based Message Authentication Code.

IDE Integrated Development Environment.

IEEE Institute of Electrical and Electronics Engineers.

IM Instant Messaging.

IoT Internet of Things.

IP Internet Protocol.

IV Initialization Vector.

JSON JavaScript Object Notation.

JVM Java Virtual Machine.

KA Key Agreement.

KDF Key Derivation Function.

LTE Long-Term Evolution.

MAC Message Authentication Code.

mTLS mutual Transport Layer Security.

NAN Neighbor Awareness Networking.

NFC Near Field Communication.

NSD Network Service Discovery.

OAEP Optimal Asymmetric Encryption Padding.

OS Operating System.

OSI Open Systems Interconnection.

OTR Off-the-Record Messaging.

P2P Peer-to-Peer.

PGP Pretty Good Privacy.

PIN Personal Identification Number.

PKCS Public Key Cryptography Standards.

PKI Public Key Infrastructure.

PSK Pre Shared Key.

QoS Quality of Service.

RSA Rivest-Shamir-Adleman.

SD Service Discovery.

SDK Software Development Kit.

SDR Software Defined Radio.

SE Standard Edition.

SEC Standards for Efficient Cryptography.

SHA Secure Hash Algorithm.

SSID Service Set Identifier.

SSL Secure Sockets Layer.

TB Terabyte.

TCP Transmission Control Protocol.

TLS Transport Layer Security.

TTP Trusted Third Party.

UPnP Universal Plug and Play.

USB Universal Serial Bus.

WLAN Wireless Local Area Network.

WPA2 WiFi Protected Access II.

WPS WiFi Protected Setup.

X3DH Extended Triple Diffie-Hellman.

Chapter 1
Introduction

When a mobile device is not able to connect to the Internet, applications that require an active connection to a centralized server can no longer provide its services to the user, even if the service in question is as simple as communicating with another device. If that other device is within a reasonable range, connectivity can be restored in a Peer-to-Peer (P2P) fashion. However, with no centralized server available, the user must take over the responsibility to verify that they are not communicating with an impostor, which is usually the duty of the application server. This raises the question of how to provide authentication and trust in an out-of-coverage situation.

This thesis focuses on security and trust in social mobile applications in a P2P scenario where a previously available mutually trusted server has become unavailable.

This chapter will motivate the thesis by introducing previous work on the topic and some of the challenges that are yet to be addressed, the scope of this work, as well as the methodology that was chosen to address these issues.

1.1 Background and Motivation

Smartphone users today are used to being connected at all times. At the same time, there are vast amounts of remote land that is without cell coverage, and natural disasters may cause Internet connectivity outages even in populated areas.

When a user moves out of cellular coverage, or is unable to reach the central server of a service for any reason, connectivity is lost. As a consequence, Internet-based mobile social applications will not be able to provide their services and the end-users will become isolated until Internet connectivity is restored, even if the users that are making use of the service, are within a few meters of each other.

Modern smartphones are equipped with a number of radio interfaces that enable wireless communication among devices in close proximity. These capabilities could be

used to establish connectivity between neighboring devices even in the most remote out-of-coverage locations. Thereby maintaining the ability to chat, or transfer any other type of data.

WiFi Direct, a wireless technology allowing WiFi devices to connect directly to each other, is one of the methods for nearby mobile devices to establish connectivity. It is particularly widely available due to it not requiring any specialized hardware apart from a typical WiFi chip, and easy to use as addressing can be done using the familiar Internet Protocol (IP).

Even though WiFi Direct can be used to reestablish connectivity, a social application usually depends on a central server that users trust. When connected to this server, users have confidence that their messages are delivered to their respective recipients and that no other users of the service can access their private conversations.

When clients are isolated from the Internet, and have to rely on direct connections to each other, this server can no longer provide credibility and trust on demand. Without additional measures, end users cannot know if they are communicating with an impostor, and they cannot trust that the contents of the messages they are transmitting are not being observed or altered by a malicious third party. This raises the question of how to secure communication on mobile social networks when offline.

1.2 Related Work

A number of commercial applications for P2P communication exist for the Android ecosystem. The most prominent, FireChat, made headlines in 2014 when it accomplished half a million downloads over a period of two weeks as Hong Kong protestors used its P2P functionality to organize efficiently even in areas with heavily congested network traffic [Sha14]. It uses a proprietary combination of WiFi and Bluetooth to form a mesh network of devices, and even boasts interoperability between Android and iOS [Fir].

An open source privacy-focused, decentralized, alternative to FireChat, Briar, also uses Bluetooth and WiFi to communicate in addition to the Tor network [Bria]. It uses public key cryptography to manage identities and secure the communications link, but its decentralized nature and lack of a universally trusted entity makes exchanging identities a difficult problem that in practice requires the two parties to physically meet and manually exchange keys before a connection can be made [Braa]. Supporting Bluetooth also means that Briar cannot easily reuse secure socket implementations from the operating system (as they have not been designed to be carried by the Bluetooth protocol), and resorts to maintaining its own secure transport protocol [Brab].

A commercial framework by the name of Bridgeify promises simple P2P communication over a proprietary protocol carried by WiFi Direct and Bluetooth. They even provide multi-hop transmission by joining a mesh of devices using the same Software Development Kit (SDK). The Android version does, however, only support Bluetooth transport, which requires hardware support for Bluetooth Low Energy (BLE) advertising mode [Brib].

A large-scale research effort by the name of the Serval Mesh aims to create an independent network by relying on WiFi devices to form a mesh network based on WiFi ad-hoc mode [GSP11]. Unfortunately, WiFi ad-hoc mode is mostly unavailable on consumer smartphones without modifications, thus making it unsuitable for many use cases.

It is also worth mentioning that work is being done to facilitate device to device communication in the Long-Term Evolution (LTE) standard [FDM+12, LZLS12]. If this technology were to make it to consumers, it might mitigate some of the need for developers to maintain their own P2P solution, as it could for instance provide connectivity to a device by routing its traffic through other devices until it reaches a cell tower. Though helping to resolve the same problems, it makes use of radically different technology than this project aims to evaluate.

Shahin and Younis present a framework for P2P networking of Android devices using WiFi Direct that covers discovery, connection establishment, peer management and communication between peers in a single group [SY14].

Wong et al. noted that connecting Android devices with WiFi Direct uses WiFi Protected Setup (WPS) which requires manual user interaction to accept the connection prompt. They propose using WiFi Direct to create Access Points (APs) that advertise their Service Set Identifier (SSID) and Pre Shared Key (PSK) using Network Service Discovery (NSD) instead of leveraging fully fledged WiFi Direct connection establishment. Any WiFi capable client can then connect to the WiFi Direct AP like they would connect to any other WiFi AP (referenced as connecting as a legacy client in WiFi Direct terminology) without the need for users manual verification [WVNA14].

A WiFi Direct device may support being a member of more than one group simultaneously by having access to more physical radio interfaces, or by simulating multiple interfaces on the same physical hardware [Wi-16, Section 2.3]. Unfortunately, this is not currently supported by the Android operating system. Multiple authors have worked around the inability to connect to two WiFi Direct groups simultaneously by connecting to one of them natively, and the other as a legacy client to facilitate multi-hop communication between multiple WiFi Direct groups [CCP+15, FTH17]. This mode of operation requires intermittent connecting and disconnecting to various

Group Owners (GOs) to maintain a fully connected network. The manual user interaction required by GOs to accept new connections makes this an impractical solution.

An alternative approach to multi-hop transmission was proposed by Liu et al. in 2016. By turning all the clients into GOs, they are all ready to be connected to at any time. When a device wishes to transmit a message to a neighboring peer, it simply disbands its own group, and connects to the peer as a Group Member (GM). By maintaining a simple routing table, a novel ad-hoc network is achieved. They do, however, note that the rapid switching of GMs cause unstable peer discovery that impairs the network functionality [LSY+16].

1.3 Open Challenges

WiFi Direct enforces mandatory encryption, but only through the WPA2-Personal standard, and as such, lacks enterprise authentication methods such as Extensible Authentication Protocol Transport Layer Security (EAP-TLS). This means that WiFi Direct, by itself, cannot provide authentication of the connecting parties without manual interaction such as entering a PIN number, pushing a button to verify the connection, or through out-of-band verification over some medium such as Near Field Communication (NFC) or Universal Serial Bus (USB) [Wi-16, DBL17].

A simple means of automated authentication of connected devices is therefore missing from the WiFi Direct standard. Users may securely transmit messages to nearby WiFi Direct capable devices, but cannot, through the WiFi Direct standard alone, verify the identity of the user they are communicating with.

The goal of this work is therefore to propose, implement, and experimentally validate a combination of upper layer measures that can be used to securely and easily enable authenticated communication in existing social applications, also in situations with no Internet connectivity. Focus has been placed on making the system as reasonable to implement as possible by making it an application of currently existing technology as opposed to an entirely new scheme.

1.4 Scope

In this work, P2P communication to realize communication between devices is only considered in a scenario where all participants have previously been connected to the Internet and have signed up to the service in question. The focus is on authentication, trust and security, but a solution for connectivity is also provided and demonstrated through a sample implementation.

Instant Messaging (IM) has been chosen as a sample use case in this work, but the proposed solution can be directly used or easily adapted to support a number of other services.

1.5 Methodology and Tools

To achieve functional secure message transmission over WiFi Direct, the project was divided into three major phases. Each phase gradually brings the project closer to a reliable IM application that not only allow users to communicate securely when connected to the Internet, but also when they are not.

1.5.1 The Design Phase

In the first phase of this project, a system to facilitate secure communication that does not rely on an always-online third party to relay messages was designed.

The system consists of logical components and specification of their operation. The connectivity issue can be solved with WiFi Direct. Transmitting messages using the network formed by WiFi Direct is trivial, but when a device other than the trusted server available over the Internet can end up forwarding messages, the integrity and confidentiality of said messages cannot be guaranteed without additional measures.

Focus was therefore placed on the message frame design to ensure that messages are kept private and delivered in an uncompromised state. The design makes use of the fact that each user has to have been online at one point to sign up to the service. Credentials can therefore be stored on their device for use during out-of-coverage operation.

1.5.2 The Implementation Phase

To verify that the proposed design was useable in practice, a proof of concept application was developed for the Android operating system in Java using Android Studio 3.3.1[1], an Android Integrated Development Environment (IDE).

The implementation phase verifies that the Java standard library and Android Platform Application Programming Interfaces (APIs) contains sufficient cryptography and connection tools to easily and reliably support the proposed design on Android without the need for time consuming custom implementations and/or rooting the devices (obtaining privileged (superuser) permissions normally not available to the user on consumer devices).

[1] Android Studio, https://developer.android.com/studio/

Six Nexus 6P devices running Android 8.1.0 were used during development and testing.

1.5.3 The Validation Phase

After development, the proof of concept application was analyzed to validate the implementation and its compliance with the main goal. Network traffic was examined at multiple network layers to experimentally verify that an adversary with access to this layer could not learn anything significant about the users and their messages, and the overhead from the system was experimentally measured through benchmarking and simplified calculations.

The overhead and security properties of the proposed system was also compared to the popular messaging application WhatsApp[2] by review of the WhatsApp security white paper and debugging of the open source Signal IM application[3], which implements the messaging protocol that WhatsApp claims to use [wha17b].

Packet captures were made with a TP-LINK WN722N wireless USB adapter and an Ettus USRP B210 Software Defined Radio (SDR) using gr-ieee802-11[4], an IEEE 802.11 a/g/p transceiver for GNU Radio.

Data captures were subsequently analyzed with the network protocol analyzer Wireshark[5].

Some interaction with, and debugging of, NSD was also done with `wpa_cli`, a text-based frontend program for interacting with `wpa_supplicant`. `wpa_supplicant` being a software implementation of an IEEE 802.11 client available on many Operating Systems (OSs), including the Ubuntu based development machine.

[2]WhatsApp, https://www.whatsapp.com/
[3]Signal, https://signal.org/
[4]gr-ieee802-11, https://github.com/bastibl/gr-ieee802-11
[5]Wireshark, https://www.wireshark.org/

Chapter 2
Proposed Approach

This chapter covers the proposed solution for security and trust in social mobile applications in a P2P scenario. An architecture for an application that allow users to authenticate one another and securely communicate both when connected to the Internet and when not is presented. Technologies that meet the requirements for implementation of said architecture are then detailed.

2.1 System Architecture

A system consisting of three logical components is proposed. These components can ble implemented to facilitate secure communication between users both when connected to the Internet and when out-of-coverage.

2.1.1 The Authentication Component

The authentication component is the single mutually trusted entity (Trusted Third Party (TTP)) among the members of the social network. In Public Key Infrastructure (PKI) terms, this entity represents a Certificate Authority (CA), with the sole responsibility of managing the identities of the users. This component is only available over the Internet, and can therefore not be reached in out-of-coverage operation.

To reliably verify the identity of users and authenticity of messages, the system requires a cryptographic suite that supports key- and signature generation. The choice of cryptographic schemes are implementation specific, and is further discussed in chapter 3.

Identity Certificates

All users of the system have to posses a digital certificate in the X.509 format. The X.509 standard is a widespread format for defining digital certificates that consists

of a public key as well as an identity. The format also includes a signature that has to be generated by the authentication component.

Signing up

Signing up to the service is done in the same fashion as in a traditional PKI system. The client generates a key pair and creates a Certificate Signing Request (CSR) that is sent to the authentication component for signing. The CSR contains the public key as well as the identity (or Distinguished Name (DN)) that the certificate is for.

The DN should contain some human readable component that users can later use to distinguish between their contacts. This can for example be a username or an email address. In the case of an email address the authentication component should verify ownership of the address by sending out a verification link before issuing the certificate.

After the authentication component has approved the CSR, the applicant is issued a signed digital identification certificate that contains a signature that binds the public key to the DN. This means that the authentication component has approved the public key contained by the certificate as belonging to the specific DN also contained in the certificate. The resulting X.509 certificate can be used to authenticate to other entities in the system, offline or not, as shown in Figure 2.1.

For example, users that want to communicate with `some_user@example.com` can ask anyone posing as that user to produce a certificate with `DN=some_user@example.com` that is signed by the authentication component and subsequently ask them to prove knowledge of the private key corresponding to the public key contained by that certificate (for example by asking them to sign a nonce[1]).

Certificate Revocation

Excluding users from the system might be necessary if they have violated the terms of conditions or otherwise acted in an unwanted fashion. Even if they have done neither of this, a user's private credentials may have become compromised, and new ones have had to be issued.

To prevent these unwanted users and old compromised credentials from being a part of the system, a Certificate Revocation List (CRL) can be maintained. A Certificate Revocation List (CRL) is a simple list of certificates that have been flagged as not to be trusted. Users in possession of this list can cross check any certificate they come across with this list before proceeding with communication.

[1] A nonce is an arbitrary/random number that is intended to be used just once.

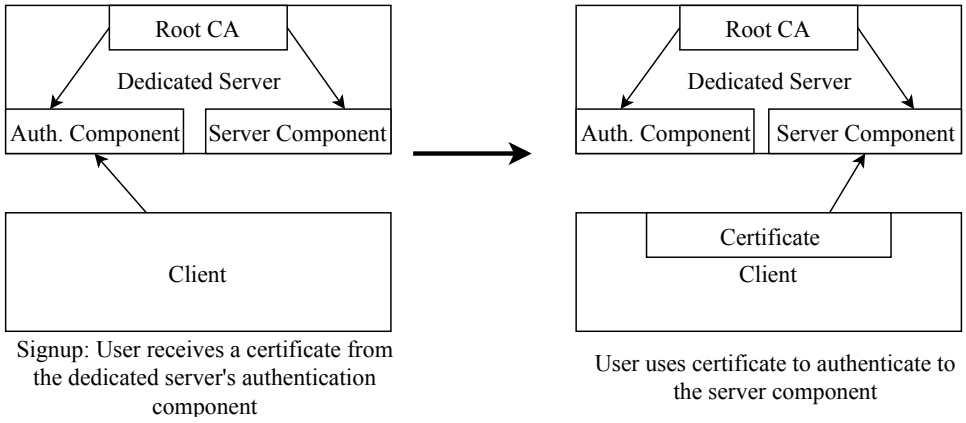


Figure 2.1: A client signing up to the service by obtaining a digital certificate from the authentication component and using it to authenticate to the server component.

A drawback to this approach is that a CRL can grow to a significant size if a large enough number of certificates are added to it. To remedy this, certificates can be issued with relatively short expiry times. Expired certificates are always rejected, and does therefore not need to be included on such a list even if they are not to be trusted. If older entries to the list can be pruned sooner, the list can be kept minimal in size and clients can request an updated copy from the authentication component every time they are online.

A prolonged offline period increases the chance that a CRL update has been issued that a user has not yet fetched. As these lists are signed by the authentication component, they do not need to be obtained directly assuming the authentication component signature cannot be forged. List updates may therefore be propagated between peers if an offline client has obtained an update that another has not.

Limits to how long a user can remain offline before having to connect to obtain a new revision of the CRL can also be implemented, but will of course subtract from the user experience as a user might find themselves unable to use the application when they need it the most.

2.1.2 The Server Component

The server component is responsible for message forwarding to connected clients both over the Internet and during out-of-coverage operation.

Connectivity

The server component accepts incoming TCP/IP connections on a predefined port. The server requires a cryptographic network protocol that allows both the client and the server to exchange and authenticate each others identities.

By verifying that identity certificates are issued by the appropriate authority (the authentication component), both parties (the client and server) confirm that the other entity is a registered user of the service, and learn the other party's identity. The server then maintains a simple forwarding table linking the connected clients identity (public key) to the appropriate socket.

Out-of-coverage Operation

The server component can either run on the dedicated server and be available over the Internet, or on a user's device in an out-of-coverage situation. When running on a user device it is also the server components responsibility to set up and manage P2P connectivity to nearby devices.

If the user device is unable to locate any other nearby P2P devices hosting an instance of the server component, the server is instantiated on the device in out-of-coverage mode. It will autonomously form a P2P device group, and broadcast the information needed to connect to said group using some service discovery mechanism.

When the group is formed, the device will accept incoming connections from nearby devices and manage message routing just like it would during regular operation on the dedicated server. Because the server component is designed to accept TCP/IP connections, the chosen P2P data link must support carrying the Internet Protocol (IP).

If Internet connectivity were to be restored at any time, the device will simply detect the connectivity change and reconnect to the dedicated server, tearing down any open P2P connections.

Message Routing

A message packet with five fields as seen in Figure 2.2 is transmitted by the clients to the server when they wish to communicate with another user. The recipients public key indicates which client the message should be forwarded to and the signature (protecting the integrity of the other fields) is verified using the attached public key belonging to the sender.

Caching and retransmission of messages is the responsibility of the client, making the operation of the server rather simple. If signature verification fails, the message

is discarded. If not, the server examines its active connections and checks if a client has connected with the identity of the receiver. If so, the message is forwarded to the appropriate client. If the server does not have an open connection to the correct recipient, the packet is also discarded.

Timestamp	Recipient	Sender	Ciphertext
Signature			

Figure 2.2: A message packet consisting of a recipient and senders public key, ciphertext, timestamp and a signature on the other elements.

2.1.3 The Client Component

The client component is responsible for connecting to and authenticating the server component as well as managing messaging.

Connectivity

The client component must be expected to connect to and disconnect from multiple server component instances in a single session as a device might move in and out of range of cellular coverage and P2P groups.

If not connected to a server component the client will first attempt to establish a connection to the central dedicated server via the Internet. At the same time, it will start the device discovery process to locate nearby P2P groups. If the dedicated server is not reachable and no group is found, the device will set up an instance of the server component and form its own P2P group.

The first device to form a group will advertise its connection information so that nearby devices can discover and join the P2P group. Upon joining a group, the client will attempt to connect to the server component instance running on the device that advertised the group, instead of the dedicated server.

Just like the server component, the client enforces secure connections and will only connect to server components that offer identification certificates from the authentication server over the chosen application layer cryptographic protocol. Figure 2.3 shows how two devices form a P2P group and reestablish connectivity if the dedicated server is unreachable.

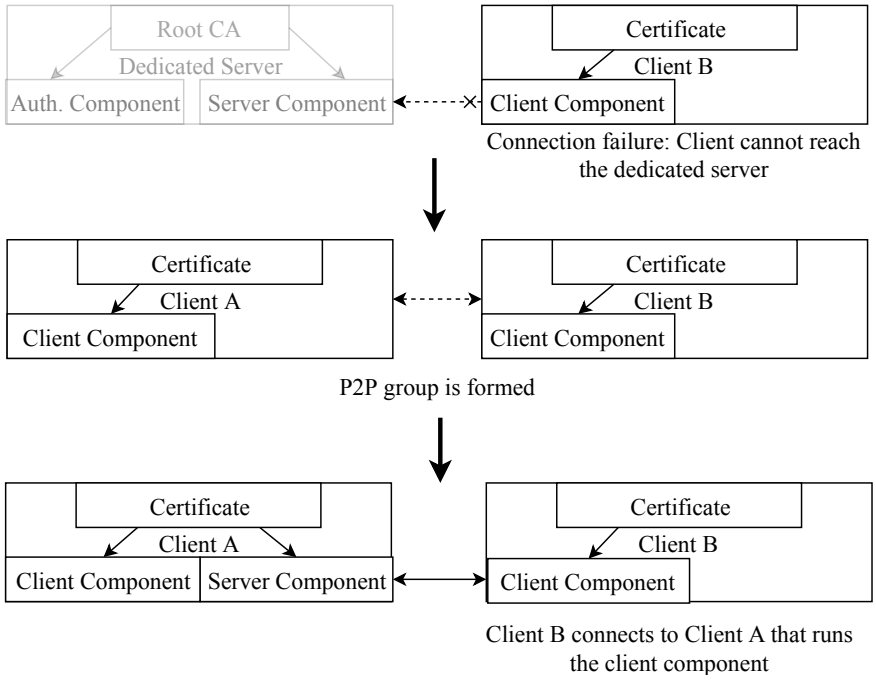


Figure 2.3: A client unable to reach the dedicated server establishes a connection with a nearby device. Arrows from the certificates to the client and server indicate that these components use the certificates to authenticate when connecting to each other.

Messaging

Message packets (see Figure 2.2) are transmitted to any server component as soon as possible. If the recipient does not acknowledge the message, the client assumes it was not delivered and caches it for retransmission. If the client connects to another server component it will immediately attempt to transmit all queued messages, otherwise it will periodically attempt to retransmit them with an exponential backoff strategy where the client periodically retransmits the message with increasing delays between attempts.

2.2 Exploited Technologies

The proposed approach requires a P2P technology, and an application layer cryptographic protocol. Technologies that meet the required specifications outlined in the previous sections are introduced here along with a short overview of modern cryptography.

2.2.1 WiFi Direct

Since its conception two decades ago, the Institute of Electrical and Electronics Engineers (IEEE) 802.11 standard, more commonly known as WiFi, has constantly evolved in the form of a series of amendments to better fit the growing number of connected devices and their user's ever changing use cases and requirements.

The most commonly used mode of 802.11 connectivity, infrastructure mode, relies on an AP to act as an intermediary between connected devices (and the Internet). The lesser common WiFi ad-hoc mode allows direct communication directly between devices. However, this mode never saw widespread use, and has multiple drawbacks such as a lack of sufficient power saving and Quality of Service (QoS) features [CMGSS13].

WiFi Direct (also known as WiFi P2P), builds upon IEEE 802.11 infrastructure mode, but also allows WiFi devices to connect to one another without the need for an AP. This is done by forming clusters of devices known as groups where one device takes over the functionality usually handled by a dedicated AP [Wi-16]. Unlike WiFi ad-hoc mode, it is also available in all Android 4.0 (API level 14) or later devices with WiFi support [Andc].

Architecture

WiFi Direct consists only of uniform devices with equal capabilities, unlike traditional infrastructure WiFi which has a clear distinction between an AP and client.

These P2P devices are required to support both the GO and GM role. The GO acts as an AP-like entity, while the GM acts like a client. All WiFi Direct devices are required to implement a discovery mechanism as well as WPS.

WPS (originally known as WiFi Simple Config) is a standard for authenticating and associating devices with secured WiFi networks through simple actions such as pressing a button or entering a short Personal Identification Number (PIN), rather than a long passphrase.

Even though the GO must provide AP-like functionality to associated GMs, it is not required to provide communication between connected devices or provide them with access to the Internet. If providing direct connectivity between GMs, the GO will set the *Intra-BSS Distribution* flag in the group capability field of its beacon frames. If providing Internet connectivity, the *Cross Connection* flag will be set. Using WiFi Direct may prevent a device from simultaneous Wireless Local Area Network (WLAN) connections depending on the implementation [Wi-16].

Device Discovery

Device discovery uses active scanning to quickly discover nearby devices that are available for connecting.

Before looking for other peers, a device first scans for already existing groups using the scanning process from IEEE 802.11-2012 [iee12]. To limit the scan to WiFi Direct devices, the transmitted probe request frames may be targeted only at SSIDs that are prefixed by the word `DIRECT` as seen in Figure 2.4. After the scan phase, the device can move onto the find phase where it can locate nearby peers.

When searching for other devices in the find phase, a device alternates between the listen state and the search state. In the listen state, the device monitors one of the designated social channels (implementation specific, but typically 1, 6 and 11 in 2.4 GHz) for a random amount of time for probe request frames from other devices. In the search state, the device transmits one or more probe request frames on each of the social channels. By alternating between the two states, and with the random duration of the listen state, devices will eventually converge on a common channel where one of the devices is in the listen state and the other in the search state.

When receiving a probe request in the listen state, a device will transmit a probe response frame back to the sender, and the two devices will be aware of each other and can continue with group formation if desired.

Time	Source	Destination	Protocol	Length	Info
14.689328	fe:3f:7c:a6:00:57	ff:ff:ff:ff:ff:ff	802.11	260	Probe Request, SN=2076, FN=0, Flags=........, SSID=DIRECT-
15.065508	de:09:4c:94:1f:c7	fe:3f:7c:a6:00:57	802.11	381	Probe Response, SN=2915, FN=0, Flags=....R..., BI=100, SSID=DIRECT-Uq-SecureComms

Figure 2.4: WiFi Direct devices exchanging probe requests/responses.

Service Discovery

An optional service discovery procedure may be performed before deciding to connect to a device or not. This procedure can allow devices to learn which services are offered by other devices before connecting. It is a flexible procedure that can carry a number of upper layer protocols to announce and discover configuration artifacts such as IP-addresses, device names and device capabilities.

It follows the Generic Advertisement Service (GAS) protocol detailed in the 2012 revision of the IEEE-802.11 standard [iee12]. Service Discovery (SD) query frames can be transmitted by unicast (one-to-one transmission from one device to another) to discovered devices without having to form a group. The queries are to be answered by a SD response frame carrying information about the responding device. These SD frames have vendor specific fields that can carry varying information depending on the upper layer discovery protocol chosen.

Android has implemented support for two Domain Name System (DNS) based protocols that can be used for WiFi Direct service discovery: Bonjour[2] and Universal Plug and Play (UPnP)[3]. Both these protocols can carry DNS TXT records which can contain arbitrary text.

Group Formation

Before being able to transmit arbitrary data, devices that have discovered each other have to form a group with one GO. The group formation procedure handles negotiation of which device should be the GO and provisioning of the network.

The GO negotiation process is a simple three message procedure between two devices that includes the group configuration and an intent value from each device. The intent is a numeric value representing the devices desire to become the GO. The device that declares the highest intent value is elected as GO. A tie breaker bit is also included in the case that the devices declare the same intent value. The elected GO must then start the provisioning phase.

Provisioning is started by WPS where the GO acts as an AP, and the GM serves as the enrollee. After the GM has obtained the WPS credentials, it may connect to the group, and the formation is considered complete. Other clients may now also connect to the network.

If a device does not want to negotiate with other peers, it may also autonomously form a group, thus becoming the GO and skipping these steps [CMGSS13].

Operation

WiFi Direct operation once a group has been formed resembles IEEE 802.11 infrastructure mode where the GO has assumed the role of the AP.

The group is formed with a star topology (as seen in Figure 2.5) where the central node (GO) has a connection to all the others. The other nodes (GMs) may freely leave and join the group, but if the GO disconnects, the group disbands. The GO can therefore be considered a single point of failure.

In addition to P2P group operation, allowing GMs to connect via WPS, a GO must also maintain an SSID and a passphrase that clients can use to connect via WPA2-PSK like they would connect to any other AP. This mode of operation is referred to as legacy mode.

[2]Bonjour, https://developer.apple.com/bonjour
[3]UPnP, https://openconnectivity.org/developer/specifications/upnp-resources

As proposed by Wong et al. in their 2014 paper *Automated Android-based Wireless Mesh Networks*, using the service discovery capabilities of WiFi Direct to connect in legacy mode bypasses the manual interaction required by connecting with WPS [WVNA14]. Connecting to a client in this special mode of operation leverages a mix of WiFi Direct capabilities and regular WiFi operation seen in Figure 2.6 that will prove useful for automating the connection procedure of the proposed system.

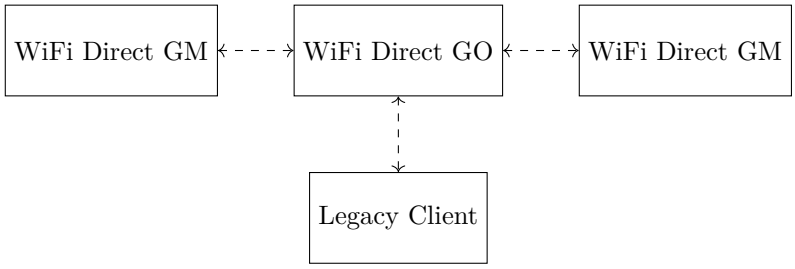


Figure 2.5: Topology of a WiFi Direct group consisting of three WiFi Direct devices and one legacy client.

2.2.2 Mutual Transport Layer Security (mTLS)

The Transport Layer Security (TLS) protocol provides authentication, confidentiality and integrity protection between two parties communicating over a *reliable, in-order data stream*. When connecting using the bare bones TLS protocol, the client typically verifies the authenticity of the server, and client authentication (if desired) is left up to the application layer (for example in the form of a login page).

However, an optional mode known as mutual Transport Layer Security (mTLS) also allows the server to authenticate the identity of the client during the handshake on the transport layer, as seen in the full TLS handshake message flow in Figure 2.7. In this mode both parties request and verify each others identity [Res18, Section 1].

TLS supports many different means of key exchange, encryption and integrity protection. Which algorithms to use to ensure these connection properties are negotiated during the session setup. Each client may have different capabilities and preferences, and may therefore place restrictions on this choice that the other party has to meet. A choice of security algorithms in TLS is often referred to as a cipher suite.

2.2.3 Cryptography

The proposed system architecture makes use of a number of cryptographic schemes to protect identities and transmitted data. These schemes fall into one of the

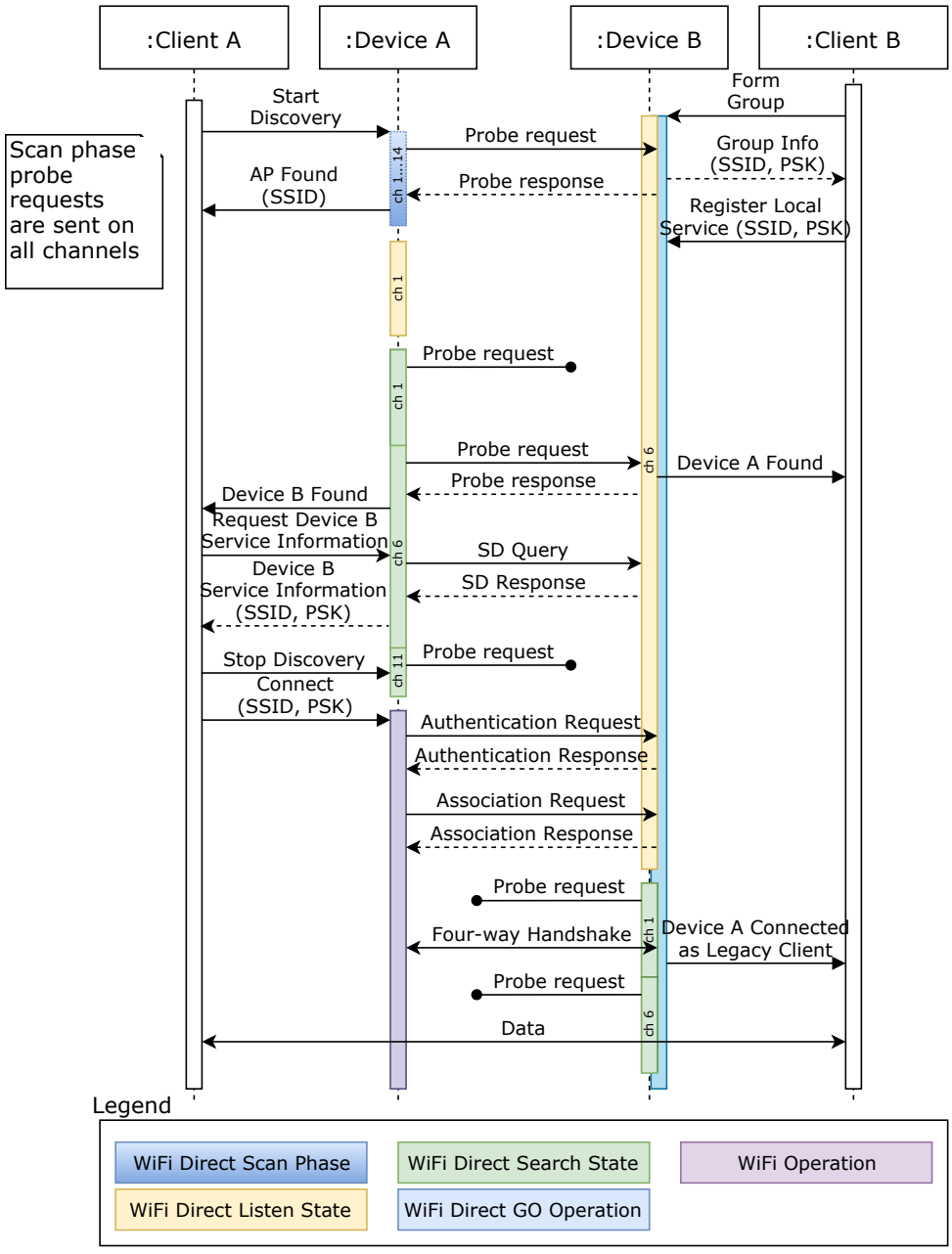


Figure 2.6: Sequence diagram detailing how a WiFi Direct capable device may connect as a legacy client by obtaining credentials using service discovery.

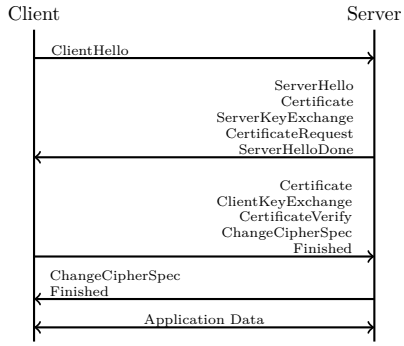


Figure 2.7: Full TLS handshake protocol message flow. Adapted from [EAA09, Figure 1].

following two broad categories. Hybrid cryptography combining schemes from the two categories is also used, as it benefits from the strengths of each type.

Symmetric Cryptography

Data transmitted over a socket secured by TLS can be considered secure as it has been encrypted by symmetric cryptography. Symmetric cryptography is relatively computationally inexpensive and provides a high level of security given the right choice of algorithm. Common for all these algorithms is the use of a shared key that is used for both encryption and decryption. The shared key is typically negotiated between the two communicating parties on a per-session basis.

As seen in Figure 2.8, to recover the original message from the ciphertext, one must posses the corresponding shared key. Agreeing on, or sharing a symmetric key without the possibility of an attacker learning it, is an important challenge that must be solved before being able to safely use symmetric cryptography.

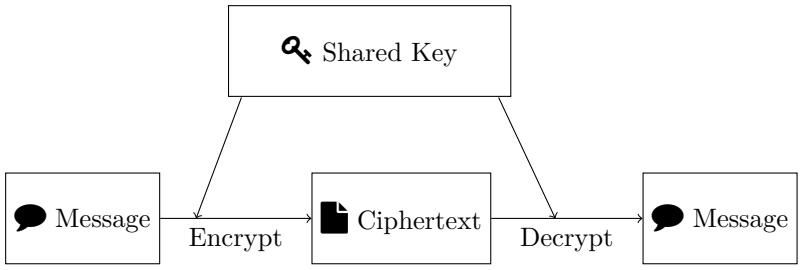


Figure 2.8: Symmetric cryptography encryption and decryption.

Public Key Cryptography

Public key cryptography (or asymmetric cryptography) can be used to authenticate the identity of the connecting parties during TLS session setup. This optional capability is vital to use in order to prevent *man-in-the-middle* attacks where an adversary secretly places themselves in the middle of two parties and relays their messages, potentially listening in on or altering them.

In a public key cryptography system, a key pair consisting of a public and a private key representing an identity is held by each party. The public key is a unique identity intended for sharing. The private key is, as the name implies, never to be revealed. Mere knowledge of a private key proves ownership of the corresponding public key and its associated identity.

A key pair is randomly generated on the device that will use it based on a random seed (as seen in Figure 2.9) so that the private key never has to be transmitted. This is done to minimize the possibility of the private key being learned by anyone but the original owner.

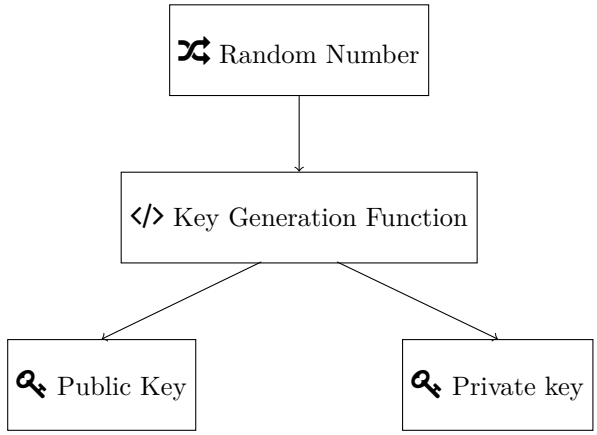


Figure 2.9: Generation of a key pair for use in a public key cryptography system.

Keys are often packaged in digital certificates that tie them to an identity before being shared. The procedures behind the issuing and management of digital certificates is often referred to as Public Key Infrastructure (PKI). The purpose of a PKI is to manage trust in the certificates. Since anyone can generate a public- and private key pair, they are not particularly valuable by themselves without trust in their origin.

A common way of managing this trust is using a Trusted Third Party (TTP). If the entities that wish to connect to each other can agree on a third party that

they both trust, the TTP can vouch for the authenticity of a digital certificate. The TTP has its own key pair and its public key is distributed to all clients. Before communicating with anyone else, each client submits their digital certificate to the TTP, known as a CA in PKI, and receive a digital signature back that can be sent to anyone along with the digital certificate. A digital signature from the CA can only be generated using the CA's private key, and can be verified to be authentic by anyone in possession of the CA's public key.

When receiving a certificate with a digital signature attached, anyone in possession of the CA's public key can then verify that the digital signature is valid. As shown in Figure 2.10, only a public key is required to verify if a signature was generated by the corresponding private key. By trusting the CA and its ability to verify the digital certificate ownership, it is also safe to assume that the certificate represents the identity it claims to if the signature can be verified.

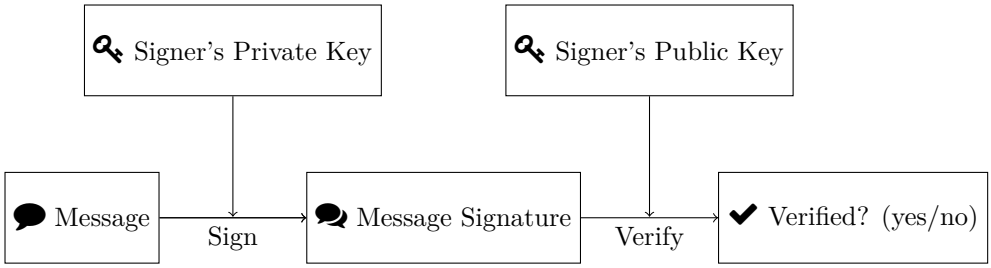


Figure 2.10: Generation and verification of a signature based on public key cryptography.

As seen in Figure 2.11, encryption can also be done if one is in possession of the recipients public identity. By using the recipients public key, a message can be encrypted by anyone so that only the recipient can decrypt it as he or she is the only entity in possession of the private key.

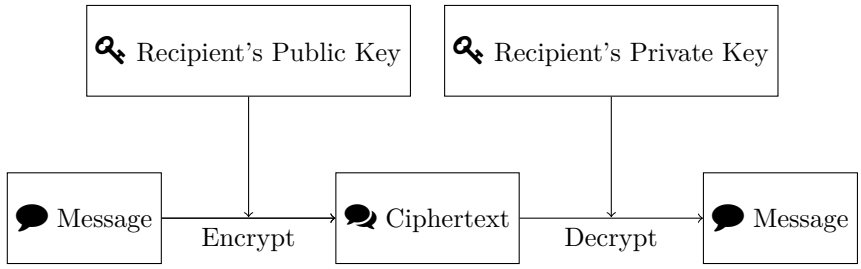


Figure 2.11: Encryption and decryption of a message using public key cryptography.

Hybrid Cryptography

The combination of symmetric and asymmetric cryptography used in TLS can be classified as hybrid cryptography. Symmetric cryptography is efficient, but requires a common shared secret. Public key cryptography does not, but often relies on more complex calculation and has a large overhead that make it less efficient. By using public key cryptography to encapsulate and share key material for use in symmetric cryptography, a desirable result combining the best features of each scheme is obtained.

The overhead induced by the public key encryption scheme is reduced as the bulk data transmission is done using symmetric encryption, and the issue on how to share the key material required for symmetric cryptography is solved by a public key encryption scheme.

Chapter 3
Implementation

The three logical components are implemented in a server and a client application. The following section details how, and outlines which steps were taken to develop a working implementation of the proposed system.

3.1 The Dedicated Server

The dedicated server is a traditional server reachable using the Internet. It implements both the server component to facilitate messaging and the authentication component to facilitate sign-up.

Servers that can handle sign up and authentication of users are widely deployed and well understood. It is therefore assumed that the dedicated server already exists and that it is able to handle sign up and authentication of users. The interested reader may find resources for implementing a chat application with a corresponding server from popular online learning resources such as Udemy[1].

3.1.1 Certificate Generation

The dedicated server must have its own key pair that can be used to issue credentials to users and establish mTLS connections to connecting clients. In the case of the sample implementation, a single root level certificate was used, but a security conscious developer may choose to issue an immediate certificate for this purpose. An immediate certificate is simply a certificate issued by the root certificate that can, in itself, be used to issue other certificates. Using one means that the root certificate can be protected by additional layers of security such as not storing it on a server available over the Internet, thus minimizing the odds of it being compromised by an attacker.

[1] The Complete Android N Developer Course | Udemy: https://www.udemy.com/complete-android-n-developer-course/

A root level certificate may be valid for decades (though it should be configured to eventually expire), so it is reasonable to manually generate it. In the case of the sample implementation, it was done using the OpenSSL toolkit[2], a feature packed toolkit for the TLS and Secure Sockets Layer (SSL) protocols.

Signing of CSRs from users can be automated in many ways, but given the small number of devices used in the sample implementation testing, development of an automated authentication server would not be a particularly beneficial use of resources, so user credentials were also manually issued using the OpenSSL toolkit.

3.2 The Client Application

The client application does not only implement the client component, but also the server component. It consists of two main activities and one debug activity to inspect messages being sent and received.

3.2.1 The Main Activity

The main activity (seen in Figure 3.1) contains a list of a user's contacts and the connection status of the client component and the server component.

The connection status has four distinct values representing different stages of the application connection phase.

- **Setting up:** The application is loading
- **Connecting:** The client component is looking for a server to connect to
- **Connected:** The client component has successfully connected to a server component
- **Hosting:** The client component has not been able to connect, and has started the server component hosting a group

Selecting one of the contacts in the list opens up the chat activity for that particular user. Pressing the info circle (ℹ) in the top right of the screen opens the debug activity.

3.2.2 The Chat Activity

The chat activity (seen in the two leftmost screens shown in Figure 3.2) allow users to send basic text messages to their contacts. It shows messages sent/received in

[2]OpenSSL, https://www.openssl.org/

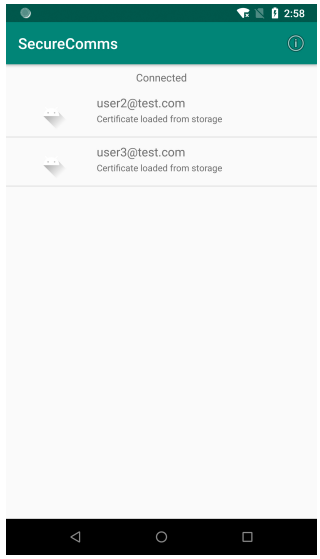


Figure 3.1: A user browsing their list of contacts in the **Main Activity**. Selecting one brings up the chat view.

chronological order with sent messages on the right, and received messages on the left. Messages are only shown if the signature can be verified and the user has the sender in his or her contact list. The name of the contact the user is chatting with is displayed at the very top.

Tapping the message field brings up the on screen keyboard and allow users to type out messages that can be sent by tapping the send button or the return key on the keyboard (⏎). The back button brings the user back to the main activity.

3.2.3 The Debug Activity

The debug activity (seen in the rightmost screen in Figure 3.2) show the messages being sent from/to, or being relayed by a user. It loads the name of the users from the contact list if an entry is found corresponding to the public keys contained in the message.

It displays the state of the attached signature by verifying it using the attached sender public key, and displays a checkmark (✔) if the signature can be verified, and a cross (✖) if the verification fails.

It also decrypts the message ciphertext if it possesses the private key corresponding to the recipient.

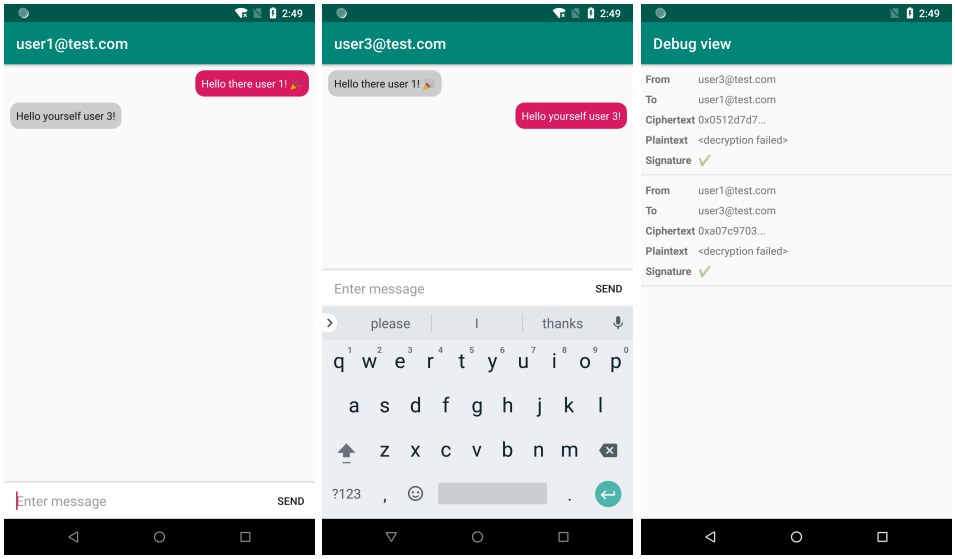


Figure 3.2: Two users chatting using the **Chat Activity**, with a third user inspecting the **Debug Activity** whilst acting as a server.

It is not possible to interact with any of the messages other than scrolling up and down if the messages have overflowed the available screen space. The back button brings the user back to the main activity.

3.2.4 Technical Details

As the application must manage a client with a connection to a server and possibly an instance of the server component as well, a simple state machine seen in Figure 3.3 is maintained to discover, prioritize and connect to the appropriate server.

To ease development, all certificates in the test application are stored on the devices external storage. This includes both the contact list and the users own credentials. This is great for development, as the credentials can be managed from the development machine by mounting the device's external storage as USB storage. The obvious drawback being that the certificates can easily be read or modified by other applications or extracted through USB.

In a fully fledged implementation it is vital that the certificates (especially ones containing private keys) are stored on the device's internal storage. This storage protects its contents from both the user and other applications on the device [Anda].

In the current version of Android (Android Pie), the GOs IP address appears to be

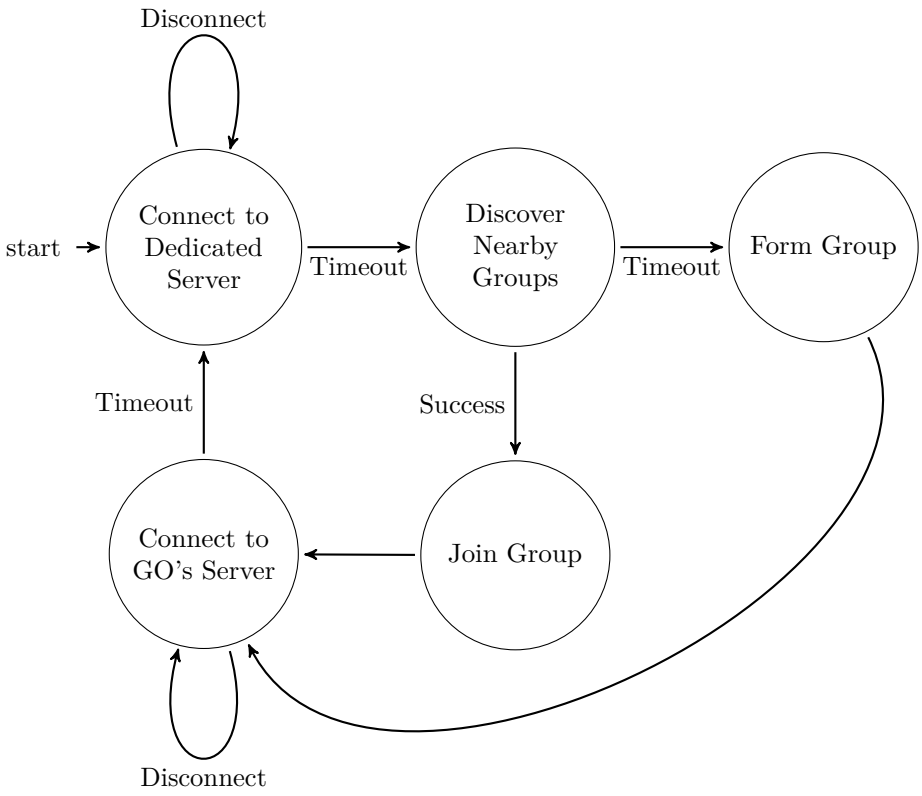


Figure 3.3: A finite state machine outlining the operation of the connectivity behavior of the application. Omitted transitions include various failure conditions that disband/leave any WiFi Direct group and return to the start state.

hard coded to `192.168.49.1` [And18]. However, this cannot be guaranteed in future versions of the OS, as it is not part of the official WiFi Direct specification [Wi-16]. The GO therefore always advertises its IP address over NSD along with the PSK for the WiFi network and the port that the server component is bound to.

3.3 Cross Platform Code Reuse

As the server component is written in Java, which can run on both user devices running Android and on any server capable of running the Java Virtual Machine (JVM), development cost in terms of codebase size can be lowered. However, a

number of considerations must be made to ensure consistent operation across both runtime environments.

3.3.1 API Variation

The code targeting both Android and the desktop should not require dependencies that are only available on one platform. Relying only on the Java standard library is a great way of ensuring this, as it can be expected to have been thoroughly tested on both platforms.

The server component does rely on some Android-only APIs to manage WiFi Direct operation, but as there is no need for the dedicated server to use WiFi Direct, this is not a problem.

An important difference to note is the variations in the cryptographic libraries available for the two platforms. The Bouncy Castle Crypto APIs[3] are available on both Android and on the desktop, albeit in slightly different flavors which is why the Spongy Castle[4] distribution of Bouncy Castle was used in this implementation.

3.3.2 Serialization

Attempting to directly serialize cryptographic primitives such as `PublicKey` objects on an Android device and deserializing them on the dedicated server will result in compatibility issues, due to minor variation between the Android and desktop version of Bouncy Castle.

For example, the `BCRSAPublicKey` object is functionally equivalent on the two platforms, but the Android variant (`com.android.org.bouncycastle.jcajce.provider.asymmetric.rsa.BCRSAPublicKey`) cannot be directly deserialized into a `org.bouncycastle.jcajce.provider.asymmetric.rsa.BCRSAPublicKey` due to their dissimilar fingerprint.

Care must therefore be taken to break the objects down into less complex objects that can be reconstructed upon delivery before transmission over the wire. The Bouncy Castle APIs supports the standardized Abstract Syntax Notation One (ASN.1) encoding of most cryptographic objects for efficient transfer between implementations. Using this over Java's built in object serialization not only solves these compatibility issues, but has a considerably lower overhead and avoids dealing with the security issues of Java serialization [Kri18].

[3]Open Source Crypto APIs for Java and C# https://www.bouncycastle.org/

[4]Stock Bouncy Castle libraries with a couple of small changes to make it work on Android https://rtyley.github.io/spongycastle/

Fixed cryptography parameters, making the length of message packet fields static, can also be used to further compress message sizes, as object identifiers and headers can be cut down on.

3.4 Cryptography

The application was implemented with two cryptosystems for comparison. Rivest-Shamir-Adleman (RSA) and Elliptic Curve Cryptography (ECC) can both be used for encryption and signing messages, but have slightly different properties. While RSA can be used by itself to encrypt short messages, a hybrid scheme is often used in ECC.

3.4.1 Rivest Shamir Adleman (RSA)

RSA, first published in 1978 by the cryptographers whose names make up its acronym name, is based on the premise that it is difficult to factor large numbers [RSA78].

RSA is a fully fledged cryptosystem with clearly defined key generation, encryption, decryption and signing. PKCS #1 covers the industry standard implementation of this functionality, which is slightly different from the version presented in the original paper [KMKJR16].

It is vital that secure padding is being used when encrypting with RSA to make it semantically secure. A semantically secure cryptosystem is a cryptosystem where information about the plaintext cannot be extracted from the ciphertext without the ability to decrypt it. If no sufficiently secure padding is used, the encryption is deterministic, and an attacker can simply try to encrypt their guess for what the plaintext might be and see if it results in the same ciphertext. In the case of short text messages, it would not take long to guess correctly. Optimal Asymmetric Encryption Padding (OAEP), a secure padding scheme often paired with RSA, was therefore used.

The maximum size of the message to be encrypted (including padding) is the modulus (ie. key size). This was not an issue for the short nature of text messages, but could become an issue for use cases with bigger payloads. Ths most common way of solving this is by implementing a hybrid scheme such as Pretty Good Privacy (PGP) in which the data is encrypted using a symmetric key that is shared using public key cryptography.

3.4.2 Elliptic Curve Cryptography

ECC is an alternative public key cryptography approach. It requires significantly shorter keys than RSA to provide the same level of security and is based on the

premise that it is difficult to find the discrete logarithm of an elliptic curve point [Van97].

ECC encryption was done using Elliptic Curve Integrated Encryption Scheme (ECIES), as it is one of the most straightforward ECC encryption schemes. It is considered a hybrid scheme due to it being a combination of asymmetric and symmetric cryptography, and is semantically secure.

To encrypt a message, a Key Agreement (KA) function is used to create a shared secret value based on a randomly generated ephemeral key pair and the recipients public key. A Key Derivation Function (KDF) is then used to generate symmetric keys for encrypting and signing the message contents from the shared secret. The actual message encryption is performed using a symmetric encryption algorithm and a digest function is used with the signing key to generate a Message Authentication Code (MAC). The recipient can then obtain symmetric keys to verify the MAC and decrypt the message by using the ephemeral public key and recipient private key, along with the encrypted message [MEÁ10].

The MAC from ECIES only preserves the integrity of the message itself, so Elliptic Curve Digital Signature Algorithm (ECDSA) was used to sign the entire message frame including the sender and recipient fields. An ECDSA signature is the Elliptic Curve (EC) variant of Digital Signature Algorithm (DSA). It is calculated using the senders private key and a random value, and can be verified by knowing the random value and the senders public key [JMV01].

Optimization

Public keys are often transferred, which motivates a compression scheme that lowers the amount of data that needs to be sent. Because they are represented as points on the elliptic curve geometric construction, public keys are coordinates comprised of an X value and an Y value. They also have to satisfy the curve equation often written as $Y^2 = X^3 + aX + b$, where a and b are the parameters that define the curve.

If one knows the curve parameters and the X coordinate of such a point, one can therefore use the curve equation to recompute Y^2. This yields two possible solutions for Y, so a single bit indicating which of the two possible points is correct is therefore appended. Further calculations will, for simplicity, ignore this extra bit, and assume that an elliptic curve point can be compressed into a single X value.

Curve Parameters

Not all curve parameters are created equal, so it is vital to select the correct ones to ensure that the Elliptic Curve Discrete Logarithm Problem (ECDLP) (ie. recovering

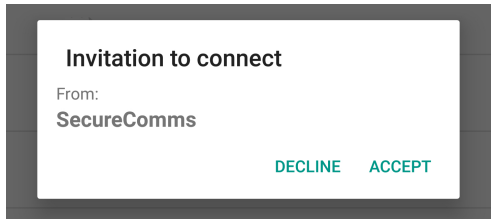


Figure 3.4: A WiFi Direct connection prompt as seen by a GO requiring manual interaction to allow a new GM to join the group.

a private key given the corresponding public key) remains a difficult problem.

Following suspicions of a back door in the standardized Dual EC random number generator, the safety of a number of widely deployed curve parameters have been questioned [BLN16, BCLN16]. For assistance in choosing these parameters, the reader is therefore urged to consult a reputable standardization body that they trust.

There are a number of standards that cover the selection of these parameters. An introduction to the most common ones as well as some valuable evaluation criteria are available from the SafeCurves initiative[5].

3.5 User Experience

The cryptography is, given the correct choice of parameters, not something that heavily influence the user experience of the application. However, some of the characteristics of the sample implementation make it possible for the user to discern whether or not the application is operating in out-of-coverage mode.

The OS shows two interface elements of note that impact the users experience of the application. The first being the pop up dialog seen in Figure 3.4. It was avoided by using Wong et al.'s NSD technique.

However, when a client connects as a legacy client, the OS will display the connection status in the notification bar at the top of the screen. As a user might associate the WiFi symbol with an active Internet connection, the OS will show a small cross letting the user know that there is no Internet connectivity (see Figure 3.5).

[5]SafeCurves, https://safecurves.cr.yp.to/

Figure 3.5: A symbol in the notification bar of an Android device letting the user know that it is connected to WiFi, but no Internet connection can be made.

Chapter 4
Experimental Results

This chapter covers an experimental validation of the proposed implementation. Network operation is analyzed to investigate the time it takes to set up connections and deliver messages, and a theoretical computation of message overheads is performed to compare various cryptosystems and their parameters. Finally, the overheads of the popular IM application WhatsApp and the P2P IM application Briar is investigated for comparative purposes.

4.1 Security Validation

The following section covers an analysis of what can be learned about the operation of the sample implementation and its users by examining a selection of network layers (based on the Open Systems Interconnection (OSI) conceptual model).

4.1.1 Data Link Layer Security

As messages are the only data frames transmitted in the WiFi Direct group, it is trivial for an unauthenticated third party monitoring the network to see which devices are transmitting messages. If the aforementioned third party is able to link the physical addresses of each device (seen in Figure 4.1) to an identity, it can keep track of when messages are sent and to/from whom.

Wong et al.'s technique of broadcasting the PSK of the formed network using NSD to allow devices to connect to the WiFi network without the need for manual verification prompts makes connection establishment considerably easier for the user [WVNA14]. However, this also means that anyone can use service discovery to learn the PSK. As seen in the screenshot of `wpa_cli` from a computer near a GM in Figure 4.2, it is trivial to discover the broadcasted service information. A user must therefore not be considered trustworthy based only on their ability to connect to the WiFi network.

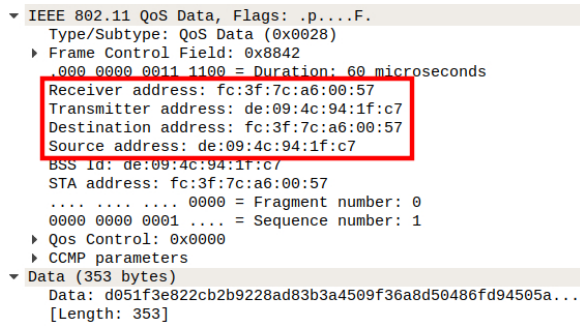


Figure 4.1: An IEEE 802.11 data frame transmitting a message from the GM to a GO (highlighted) as seen by a third party.

Encrypting the password with a static pre shared secret like in the original paper would only make the WiFi PSK marginally more difficult to recover, as the secret would have to be distributed onto every users device, from which it could be recovered and the password subsequently decoded. If the application were to be kept private, and not publicly distributed, this could be a worthwhile addition, as it would be much more difficult to recover the secret without access to the application binary.

Either way, there is no way to set the PSK required to join the group as it is randomly generated by the operating system. These randomly generated passwords appear to never be longer than the minimum PSK length (8 characters) using a 36 character alphabet (A-Za-z0-9). This might not be a sufficiently strong password to resist a brute force attack [Gol11].

The WiFi connections between the devices must therefore be considered insecure channels, as an adversary that witnesses a client handshake must be considered able to decode transmitted frames [Mac05].

4.1.2 Transport Layer Security

As the data link layer security is not sufficient to protect data over the air, the system relies on upper layer security. On the transport layer, the data is transferred over TLS. A number of attacks on TLS have been published that can defeat this protection [SHSA15b]. It is therefore vital that both the server component and the connecting clients enforce the most recent best practices for TLS [SHSA15a].

Because the engineers behind the deployment of this system have full control over the implementation of both servers and clients (unlike web server developers that often have to support older browsers), strict requirements can be placed on the connection without worrying about compatibility.

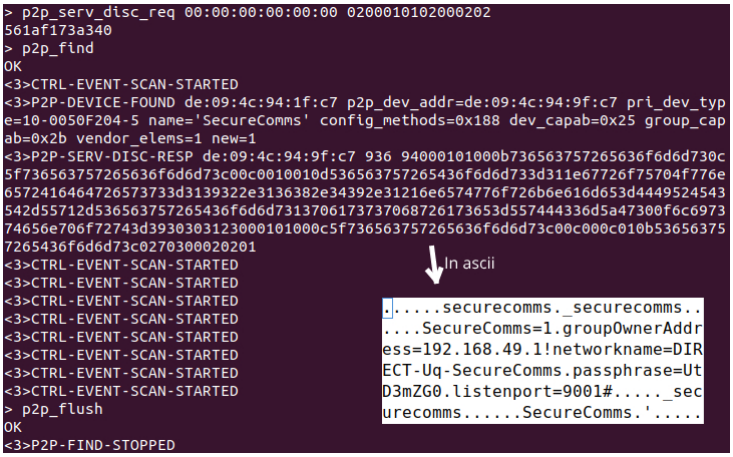


Figure 4.2: WiFi PSK transmitted using NSD as seen by a nearby device.

In addition to confidentiality and integrity protection, the transport layer also provides the access control that is lacking on the lower layers. As both connecting parties require authentication of the other party using mTLS it is impossible for a user to connect without an identity from the authentication server. If attempting to connect without the appropriate credentials, the TLS handshake will fail (as seen in Figure 4.3), and the device will be unable to communicate with other users.

It can therefore be concluded that unauthorized clients are prevented from communicating with users at the transport layer, and that the data being carried is both confidential and integrity protected due to the properties of TLS. It is however important to note that the TLS connections are not End-to-End (E2E), as they are terminated at the server component running on the GO. Upper layer measures are therefore required to protect users from a dishonest GO.

Time	Source	Destination	Protocol	Length	Info
0.650274880	192.168.49.238	192.168.49.1	TLSv1.2	309	Client Hello
0.692118913	192.168.49.1	192.168.49.238	TLSv1.2	1514	Server Hello
0.692150144	192.168.49.1	192.168.49.238	TLSv1.2	1367	Certificate, Server Key Exchange, Certificate Request, Server Hello Done
0.763972353	192.168.49.238	192.168.49.1	TLSv1.2	148	Certificate, Client Key Exchange
0.768427171	192.168.49.1	192.168.49.238	TLSv1.2	73	Alert (Level: Fatal, Description: Handshake Failure)

Figure 4.3: TLS handshake failure caused by a client attempting to connect without an identity certificate signed by the authentication component.

4.1.3 Application Layer Security

As both the data link layer and transport layer security terminates at the GO it is a natural component to study to evaluate the application layer security.

As seen in Figure 4.4, message senders and recipients are visible to the GO, and signatures may be verified, but message contents cannot be deciphered without the appropriate private keys. This means that the server component can keep track of who is chatting, but has no way of knowing the contents of the messages.

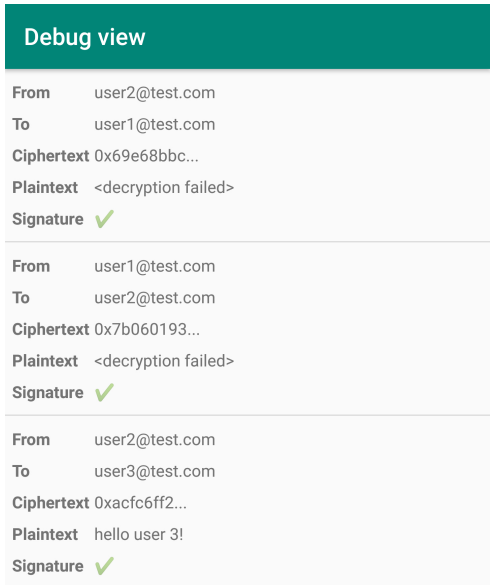


Figure 4.4: Message flow as seen by the GO `user3@test.com`. Note how the GO is unable to decipher message contents not intended for itself.

A dishonest server component could potentially attempt to modify the messages it is forwarding. In Figure 4.5, a malicious GO has modified the *from* field of a message packet, but was unable to correctly sign the message as it does not possess the private key of `user1@test.com`. The recipient (`user3@test.com`) therefore discards the message due to the signature verification failure.

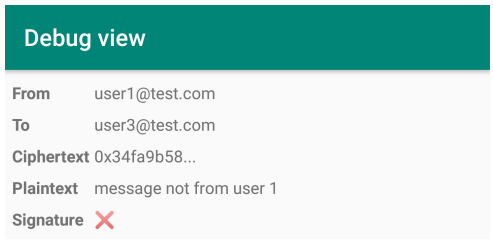


Figure 4.5: Message signature validation fails if a message packet has been altered by an unauthorized entity, as seen by the GM `user3@test.com`.

In short, the message packet format (see Figure 2.2) successfully protects the messages confidentiality and integrity E2E, but does not protect the identity of the sender and recipient.

The GO can, however, choose not to deliver, delay or even deliver messages multiple times. The latter, known as a *replay attack* has an especially undesirable impact if used maliciously. In a high security context, messages that are received long after they were created should be discarded or the user should be notified as a stale message could indicate malicious activity. Duplicate messages (same sender/recipient, timestamp, ciphertext and signature) should also be discarded.

4.2 Overhead

As with most systems that add value, the number of steps taken to secure reliable instant messaging in this research add some overhead that must be considered in the evaluation of the project. This overhead may be measured in terms of the amount of transferred data, battery usage, computation required and/or time needed to set up connectivity.

4.2.1 Connection

Using TLS adds some overhead to both connection setup and data transfer. It has, however, become extremely common, and is not unique to this system. As seen in Figure 4.3, the per session data overhead is in the kilobyte range, and the time it takes to establish the connection is barely noticeable by a human (note that the referenced figure covers a failed handshake, with slightly less data transferred than one would expect from a successful one).

In the event that the Internet connection fails, however, significant time is required to reform connectivity using WiFi Direct. Camps-Mur et al. measured the group formation delay in their 2013 overview paper of WiFi Direct, and noted that the WiFi Direct discovery mechanism introduces some randomness to the time it takes to connect to a group [CMGSS13].

As a device should go through the standard discovery phase to discover NSD advertisements from other groups that it should consider joining before forming its own, this discovery delay must be considered.

Timing of this delay was obtained by logging the time it took one Nexus 6P running the sample implementation to connect to another which had already autonomously formed a group and started broadcasting connection credentials using NSD. The process was repeated 500 times by a simple Bash script that controlled each device over Android Debug Bridge (ADB). It power cycled each device, opened the application

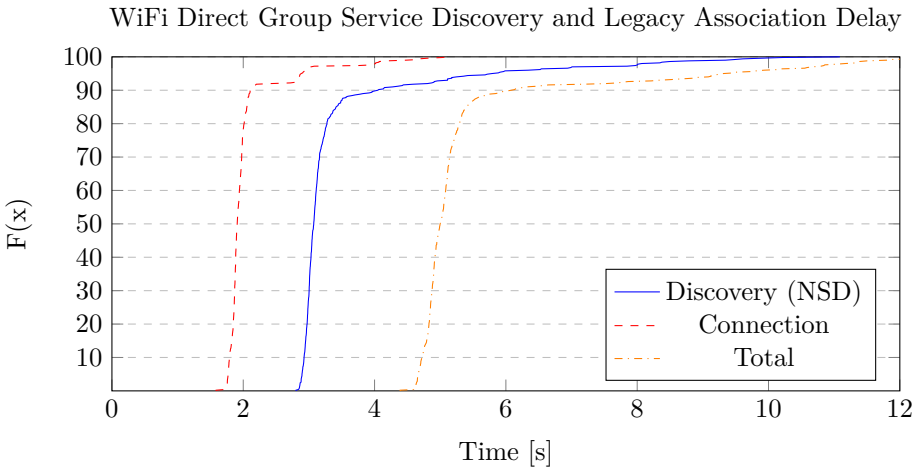


Figure 4.6: CDF of the time to discover credentials over NSD and connect to a WiFi Direct group as a legacy client.

	Mean μ	**Median**	**Standard Deviation σ**
Discovery (NSD)	3.42	3.07	1.17
Connection	2.03	1.90	0.46
Total	5.45	5.00	1.54

Table 4.1: Measurements in seconds of the time used to discover credentials over NSD and connect to a WiFi Direct group as a legacy client.

on one device, and waited for it to become GO before launching the application on the other device.

As seen in Figure 4.6, this credential discovery process may take up to ten seconds to complete. Autonomously forming the group takes next to no time, but some time is required for other clients to connect. After a group has been formed, it takes around five seconds (discovery delay plus connection delay) for the first GM to discover the broadcasted credentials and join, as seen in Table 4.1. This new client goes through the discovery process which can be expected to take three to four seconds, and uses the credentials discovered to connect to the group. However, as seen in Figure 4.6, in about 10% of cases, one of these steps does take up to twice the expected time, making the discovery and connection establishment phase slightly unpredictable.

The test implementation experienced the same issues regarding undesirable NSD behavior mentioned in Wong et al.'s original paper [WVNA14]. In some instances,

Encryption	Mean μ	Median	Standard Deviation σ
RSA 2048	8.1	7	2.4
RSA 4096	33.0	33	1.0
secp224r1	8.7	8	3.2
secp384r1	11.7	11.5	1.3
Decryption	**Mean μ**	**Median**	**Standard Deviation σ**
RSA 2048	7.6	8	0.7
RSA 4096	47.9	48	0.8
secp224r1	5.3	5	1.5
secp384r1	20.6	20	1.4

Table 4.2: Measurements in milliseconds of the time used to both instantiate (encrypt contents) and decrypt a message packet.

one or more of the test devices did not discover NSD broadcasts from other devices until they had been power cycled. The client then sees the already existing WiFi Direct group, but is unable to identify it as a group offering the chat service and to obtain the PSK required to connect. It then forms and advertises its own group resulting in two isolated groups in the same area competing for members.

It was also observed that the sample application consumed a significant amount of battery. Trifunovic et al. measured the battery usage of multiple P2P communication technologies in their 2013 paper and not only found that WiFi Direct consumes more power than similar alternatives, but also that the GO consumed an unfairly large portion compared to the GMs. This raises the question of the fairness of the group topology, and if the GO role should be periodically transferred amongst the group members even though this would cause a short service interruption [TPHH13].

4.2.2 Messaging

The message format facilitating forwarding, confidentiality and integrity protecting adds significant overhead in terms of data transfer. Asymmetric cryptography is also significantly more computationally expensive than symmetric cryptography.

As seen in Table 4.2, the time required to calculate the required cryptographic ciphertext and signatures are not insignificant. There is a noticeable gap in timings between RSA with various key sizes. Messages using EC cryptography perform well with both tested key sizes. As seen by the relatively low standard deviation and in the CDFs of the measurements in Figure 4.7 and Figure 4.8, the time it takes to perform these cryptographic operations are consistent across the tested encryption schemes.

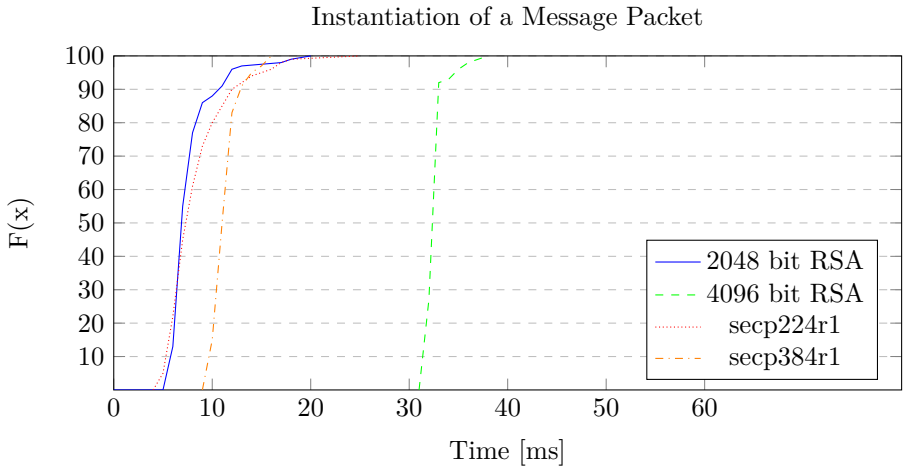


Figure 4.7: CDF of the time required to instantiate a message packet (including calculation of the appropriate ciphertext and signature) on a Nexus 6P.

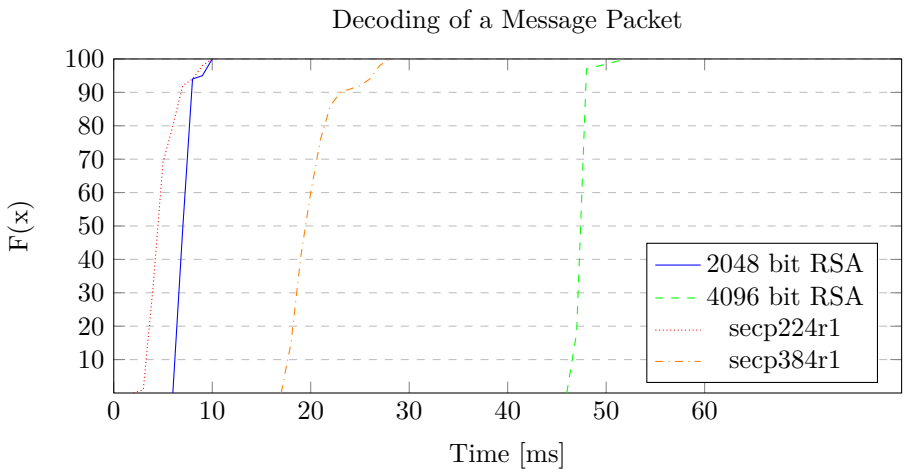


Figure 4.8: CDF of the time required to verify the signature and decode the contents of a message packet on a Nexus 6P.

The results were collected by measuring the time it took to instantiate a message packet object for sending and subsequently verifying the signature and decrypting the message on the same device 100 times. EC cryptography was tested using Spongy Castle, as the Bouncy Castle distribution shipped with Android lacks support for ECIES. The RSA ciphertext was padded using OAEP (`RSA/ECB/OAEPPadding`) and the signatures were generated using the Java preset `SHA256withRSA`.

The EC ciphertext was generated using ECIES, a hybrid encryption scheme which essentially encrypts a key using asymmetric EC cryptography. The key is then used in a symmetric encryption scheme to actually encrypt the data. It was used with Advanced Encryption Standard (AES) in Cipher Block Chaining (CBC) mode (`ECIESwithAES-CBC/NONE/PKCS5PADDING`) with a 128-bit key size for the block cipher in secp224r1 and with a 192-bit key size for the block cipher in secp384r1. The signatures were generated with ECDSA (`SHA256withECDSA`).

Using a message format with significant additional data sent with every message (see Figure 2.2) also results in additional data transfer. Most of the overhead comes from transmitting two full public keys (sender and recipient) with every message. Table 4.3 summarizes the size of a single message packet given various key sizes. The transmitted message contains 29 bytes of data, representing a typical IM message [LB07].

A number of simplifications were made in the calculation of the message sizes. The timestamp is assumed to be 32 bits (4 bytes), which is sufficient to store time with second-precision until 2106[1]. Calculations assume that all the RSA keys have been restricted to only allow keys with the same public exponent so that transmission of the few extra bytes associated with this value does not have to be considered. In the same fashion, the EC calculations assume that the EC cryptography are done on the same curve so that curve characteristics does not have to be transferred. Points on the elliptic curve are also assumed to be compressed into a single coordinate on the curve rather than two full uncompressed values.

RSA Key Strength	Message Size	EC Curve	Message Size
2048 bit	1028 bytes	secp224r1	192 bytes
4096 bit	2052 bytes		
		secp384r1	292 bytes

Table 4.3: Message packet size in bytes given various key sizes (sorted with the weakest at the top and the strongest at the bottom). The named curves are specified in SEC 2 [Res00].

[1] According to the Java Platform SE 8 `java.util.Date` which parses the unsigned 32 bit integer max value ($2^{32} - 1$) to 2106-02-07T06:28:15.000

In the case of RSA, the length of each cryptographic field is defined by the key size, making the total message length $4 \cdot keySize + 4$ bytes. In the case of EC, the sender and receiver are represented by points on the elliptic curve. The ECDSA signature consists of two numbers of the same size as the curve parameter. The ECIES ciphertext consists of one point on the curve, a 128 bit Initialization Vector (IV) as well as the encrypted message (which ends up being two 128 bit AES blocks including padding in these cases). A MAC is also typically included to protect the integrity of the ciphertext, but this is not necessary in this case, as the ECDSA signature already protects the integrity of the full message packet.

4.2.3 Other Messaging Applications

The following subsection contains an overview of the data overhead of other IM applications. They were chosen based on the availability of an open protocol specification and the existence of an open source Android implementation of said protocol.

WhatsApp

WhatsApp does not support P2P/offline instant messaging, so it does not solve the issue that this work covers, but as a popular E2E encrypted IM application, its communication can be compared to this work to validate that the proposed P2P communication solution does not add an unreasonable encryption overhead.

In 2017, the popular IM service WhatsApp published a blog post claiming that their systems handle 55 billion messages per day [wha17a]. Had they been using the message format proposed by this work with 4096 bit RSA, those messages would have been 112.9 Terabytes (TBs) in size.

Given a typical network usage cost of $0.08 per Gigabyte (GB)[2] this would mean a monthly $542,000 bill for network transfer alone. Reducing the message size tenfold by selecting EC cryptography with the associated 192 byte message format would therefore clearly be beneficial to online operation of this system at scale.

WhatsApp claims to employ the Signal protocol from Open Whisper Systems where messages are protected with AES256 encryption and HMAC-SHA256 authentication with curve parameters from Curve25519. The symmetric keys for use in this hybrid encryption, is initially derived using a modified version of the Diffie-Hellman (DH) key agreement protocol (known as Extended Triple Diffie-Hellman (X3DH)) and continuously updated by the Double Ratchet algorithm [wha17b]. The interested reader may refer to the original specification of the algorithm for more details [PM16].

[2]Google Cloud egress pricing to most countries for users exceeding 10 TB of transfer per month from the European Union (EU) region. See https://cloud.google.com/storage/pricing.

```
message SignalMessage {
    optional bytes  ratchetKey      = 1;
    optional uint32 counter         = 2;
    optional uint32 previousCounter = 3;
    optional bytes  ciphertext      = 4;
}
```

Figure 4.9: The Signal Protocol Message data structure from the open source Signal protocol library for Android[3].

As seen in the data structure from the open source Android implementation of the Signal protocol (see Figure 4.9), messages consist of ciphertext as well as a header containing an ephemeral key (also known as the ratchet key) and two counters indicating where in the complex chain of message keys the key material for the attached ciphertext can be found [MP16, PM16].

Assuming that the encryption is using PKCS#7 padding (as recommended in the protocol specification), a minimal message size for a 29 byte plaintext can be expected to be around 104 bytes excluding serialization and session setup overhead. In addition to this, the message needs to be wrapped in an implementation specific envelope which includes a recipient identifier.

By placing breakpoints in the open source Signal messenger application, from which the WhatsApp messaging protocol originates, size overhead of messages can be logged for comparison with this system. It appears that message envelopes are JavaScript Object Notation (JSON) encoded with Base64 encoded message objects serialized using Google Protocol buffers. Metadata and serialization overhead from the implementation specific message data structure brings the ciphertext of a 29 byte plaintext up to 17 32-byte AES blocks (plus a single version byte), and the total JSON encoded envelope for transfer up to 897 bytes as seen in Figure 4.10, which is significantly larger than the ECC variant of the proposed solution.

Briar

In the same fashion as the Signal messenger application, Briar's data overhead can easily be investigated by placing breakpoints in the open source messaging application. Briar uses its own open source suite of protocols by the name of Bramble that includes, among others, an application layer synchronization protocol for message encapsulation and a transport layer protocol for security.

[3] libsignal-protocol-java on GitHub, https://github.com/signalapp/libsignal-protocol-java

```
{
    "destination":"+4791694525",
    "messages":[
        {
            "content":byte[548],
            "destinationDeviceId":1,
            "destinationRegistrationId":14010,
            "type":6
        }
    ],
    "online":false,
    "timestamp":1554437133465
}
```


Figure 4.10: A message with a 29 byte plaintext from the Signal Android application intercepted before transfer.

By placing a breakpoint in the Bramble transport layer protocol `writeFrame` method, both the application data layer data (plaintext) and the encrypted frame for transport can be logged.

Sending a 29 byte message yielded 77 bytes of application data which resulted in a 113 byte message frame. As seen in see Figure 4.11, the encrypted frame includes a frame header (20 bytes), a MAC (16 bytes), padding (none in this case), as well as of ciphertext (77 bytes).

As addressing is handled by lower layers, there is no need for the frames to include recipient information, which significantly lowers the size of these frames compared to the proposed system.

These measurements are excluding additional synchronization packets which add some additional overhead roughly in the order of tens of bytes. See the Bramble synchronization protocol specification for details on these packets[4].

[4]Bramble Synchronization Protocol, https://code.briarproject.org/briar/briar-spec/blob/master/protocols/BSP.md

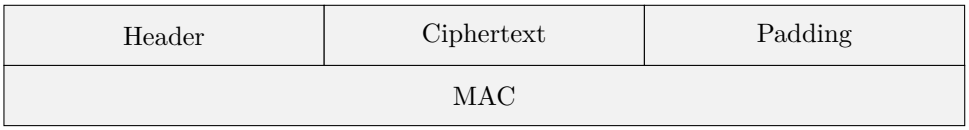


Figure 4.11: Bramble transport protocol frame structure.

Chapter 5
Discussion

The following chapter discusses the practical application of the proposed solution, its weaknesses and strengths.

5.1 Connectivity

Restoring connectivity by using WiFi Direct in the event of an Internet outage does come with significant set up time, but it may still be suitable for asynchronous applications such as IM systems and file transfers. It cannot, however, be used for applications that require uninterrupted, synchronous transfer unless some modifications are done. To achieve this, the WiFi Direct group could be set up as a fallback regardless of current connection status so that it is ready for use immediately if the Internet connection were to fail.

It is worth noting that when an Android device is connected to a WiFi Direct group in legacy mode, the connectivity information shown to the user (see Figure 3.5) might confuse the users, leading to a bad experience using the application.

5.2 Security

Issuing credentials to users in the form of digital certificates for use in asymmetric cryptography enables them to successfully authenticate one another with TLS during out-of-coverage operation.

However, as it requires the user to be online at the time of sign up, so that their digital certificate can be signed by the authentication component, it might (in some cases) be a hindrance that prevent a new user from starting to use the application. It is impossible to invite a new user to sign up to the service when it might be needed the most (when other applications fail due to a lack of Internet connection).

The overhead introduced by using strong cryptography is not insignificant, but this can be mitigated by choosing a cryptosystem with smaller key sizes. From the significant overhead introduced by large RSA keys it is clearly beneficial to choose EC cryptography as it offers the same cryptography strength at smaller overheads.

As demanding cryptographic calculations have to be done mostly by the end-users (and not by the server), scalability should not be a major issue provided that users do not send and receive messages faster than the time it takes to encode/decode them. Assuming strong EC cryptography with a worst case decode time of 30ms, as measured in subsection 4.2.2, one can achieve more than 33 messages per second.

If the TLS connection was not terminated at the GO, but GMs were allowed direct connections to each other, some of the application layer measures designed to protect the client from a dishonest server could have been avoided. However, this would have added to the implementation complexity, as the GO acting as a server makes out-of-coverage operation very similar to online operation. Additionally, as covered in section 2.2.1, intra-GM communication is an optional capability that may not be offered by all devices.

The fact that the same server component code base can be used on both the dedicated server and on the GO is a notable property that makes the proposed system easier to adopt, but requires an application layer chat protocol.

The basic chat protocol that has been proposed provides confidentiality, integrity, authentication and is resilient against messages arriving out-of-order, but lacks more sophisticated properties that further protect users in edge cases such as private key compromise.

The chat protocol protects the content of messages from a malicious GO, but the system does not, in its current state, attempt to detect a Denial of Service (DoS) type of attack, where the GO refuses to forward messages. Detecting this and electing a new GO automatically would make the system more resilient and useable in cases where a current Internet outage is a deliberate act aimed at disturbing communications.

5.3 Overhead

As WiFi Direct supports the same speeds as typical WiFi it is assumed that it provides more than sufficient throughput to carry a significant volume of these messages regardless of the chosen cryptosystem. In online mode, however, a server might have a much larger number of connected users. Keeping the message size low is therefore in the best interest of a developer in order to minimize the costs associated with bandwidth.

Based on the overhead measures from section 4.2, EC cryptography is strongly recommended in most implementations of the proposed system. This could lead to significant cost savings, especially at scale.

Comparing the message size overheads of the proposed solutions to the selected IM applications of subsection 4.2.3, it can be argued that the proposed system does not add an unreasonable overhead compared to other messaging protocols. Choosing EC with strong parameters means that the message size for a 29 byte IM does not exceed 300 bytes. However, sending full public keys of both the sender and recipient with every packet does add significant overhead that other protocols can avoid.

5.4 User Experience

The current application design requires the user to trust both the application and the authentication component. Even after convincing the user that the cryptography is sound, they still have to trust that the application works exactly as it claims to, without any back doors.

Providing users with access to the source code and allowing them to compile the application themselves can help advanced users audit the application operation and ensure them that it functions as it should. The authentication component is still, however, a black box that the user simply has to trust.

No attempts to alleviate this need for blind trust has been made in this project, however, other applications have implemented a simple measure that can be used to manually verify that a user is actually is the person that another user expect them to be.

Assuming that the user trusts the application implementation (for example after having audited its code and compiled it themselves), the application can simply show the user the fingerprint of the keys that are being used to encrypt a conversation. By meeting in person and verifying that the keys being used are shown as being the same on both ends of the encryption, it can be concluded that the encryption is sound.

WhatsApp implements this by showing a hashed concatenation of both users' identity keys [wha17b]. An example of this can be seen in Figure 5.1. The developers have made sure to communicate to the user how to use the security code and has provided a *Learn More* link that takes an interested reader to a knowledge base page where the average user can learn about E2E encryption and the value of the security code[1].

[1]WhatsApp: End-to-end encryption, https://faq.whatsapp.com/en/general/28030015

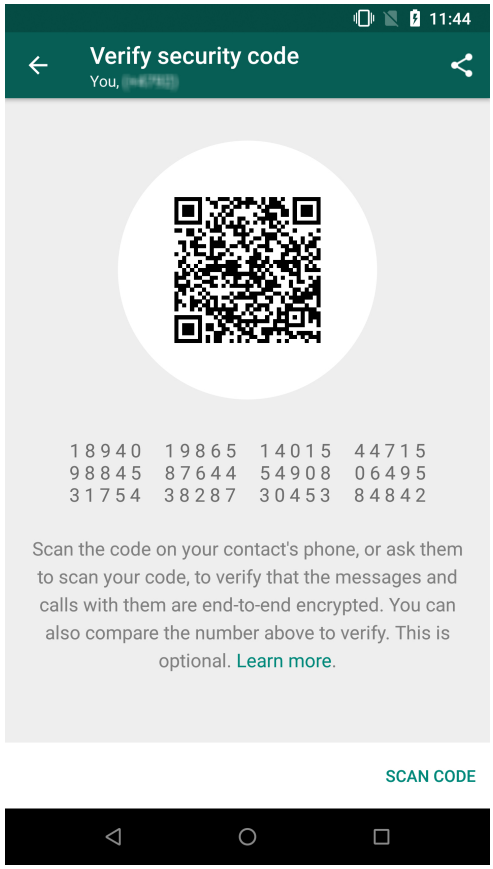


Figure 5.1: WhatsApp security code. Two users meeting in person can verify that the code shown to them is the same as to the other user to verify that their IMs are securely E2E encrypted.

Chapter 6
Conclusion

When smartphone applications are unable to connect to the Internet, many useful services become unavailable. In some cases, these services can be restored by communicating with nearby devices in a P2P fashion. However, services that exploit such technologies need to ensure security in the communication between nearby devices.

This thesis proposes an approach that ensures authenticated communication with confidentiality and integrity protection among peers. The approach enables devices to fetch security credentials in the form of digital certificates for use in asymmetric cryptography during sign up to a service. When unable to reach the centralized server, devices may use WiFi Direct to discover and connect to nearby devices, and mTLS to authenticate them while also setting up a secure data channel. A messaging protocol that may rely on the user's digital certificates must be used to protect message confidentiality and integrity during transfer over this channel.

This proposed design has been validated through implementation of a basic chat application for the Android OS. It makes use of a simple chat protocol that protect message confidentiality and integrity provided that the user's credentials are not compromised at any point in time. This has been validated by examining the transmitted data at various layers of the OSI model.

The chat protocol data overhead has been examined through the sample implementation and compares reasonably well to the open source Signal protocol, but lacks some of its more advanced cryptographic properties such as forward secrecy, raising the question of whether or not current state of the art chat protocols can be adapted to a P2P scenario.

The main drawback to the proposed system is that the user is required to be online at the time of sign up, and that the user must trust the centralized server to be honest and not to issue false credentials. Possible solutions to the latter that are

employed in popular chat applications, such as WhatsApp, is discussed in section 5.4.

6.1 Future Work

This section aims to serve as a basis for projects of similar scales to explore the topic of security in mobile social P2P communication. It introduces both suggested improvements to the messaging protocol and an alternative wireless technology that needs further investigation to determine its viability in solving the issues raised by mobile social applications that lose all functionality when they are unable to reach a central server on the Internet.

6.1.1 Messaging Protocol

This work has shown that access to a centralized server at user sign up is sufficient to facilitate authentication and encryption between users in a P2P scenario where there is no access to the Internet such as in an isolated WiFi Direct group.

The simple chat protocol proposed in this work does, however, lack a number of properties that other more advanced protocols may offer. This raises the question of whether or not a more progressive E2E encryption chat protocol could be implemented in, or adapted for, the context of a group of nearby devices moving in and out of Internet coverage.

Properties such as deniable authentication, forward secrecy and backward secrecy can help safeguard user interests and should be seriously considered for further improvements to improve user privacy.

Deniable authentication refers to a property of a cryptosystem where a recipient can be confident that a message intended for them is authentic, but a third party with access to that message cannot prove it after the fact. In other words it provides message senders plausible deniability from the perspective of third parties [BGB04].

Forward secrecy ensures that messages that have been previously sent cannot be retroactively recovered following the compromise of a users long term key material [UDB+15].

Backward secrecy (also known as future secrecy) is a security property that affects the opposite direction of forward secrecy. It guarantees that future messages are not compromised following the compromise of a users key material [UDB+15].

By combining these two properties, as seen in Figure 6.1, messages from both a reasonably long time before and reasonably long time after a compromise of key material can be considered uncompromised.

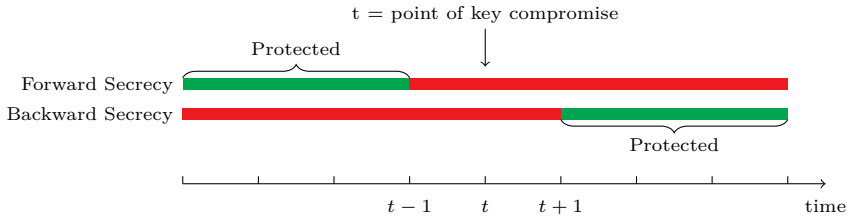


Figure 6.1: Forward- and backward secrecy comparison. Red indicates what messages from which points in time must be considered compromised following/preceding a compromise of key material.

Possible protocols for this future work include the Off-the-Record Messaging (OTR), and the more recent Signal messaging protocol. The addition of the Sesame algorithm to the Signal protocol resolves many of the issues regarding synchronization of multi-device sessions that limited the original implementation, making it a strong candidate [MP17].

A very promising specification that should be investigated for offline P2P communications is the Matrix specification for decentralized communication[1] which supports E2E encryption both between users and within groups based on the cryptographic ratchets popularized by Signal. The current version uses centralized identity servers to facilitate chat rooms that are distributed across decentralized servers. An unresolved issue (as of Thursday 6$^{\text{th}}$ June, 2019) from late 2016 on the Matrix project's official documentation still mentions *a possible evolution of Matrix to be P2P*[2].

6.1.2 Data Transport

WiFi Direct is limited by its set-up time, WPS configuration and the fact that communication relies on a central node (the GO). Exploring alternative transport mechanisms could therefore be a valuable next step.

WiFi Aware (the WiFi Alliance's Neighbor Awareness Networking (NAN) specification) is similar to WiFi Direct, but is aimed at direct device-to-device communication with nearby devices rather than communication through a central node. In WiFi Aware, devices may participate in more than one cluster, meaning that a device can communicate with all nearby devices instead of all devices currently connected to the same GO [Wi-18].

[1] Matrix Specification, https://matrix.org/docs/spec/

[2] GitHub/matrix-org/matrix-doc, Peer-to-peer Matrix (SPEC-455) #710 https://github.com/matrix-org/matrix-doc/issues/710

WiFi Aware does not appear to be widely used, and little literature exists on the topic, but Saloni and Hegde has noted that while its power requirements are currently too high for Internet of Things (IoT) application, mobile to mobile is a suitable use case [SH16]. WiFi Aware has been implemented in Android as of version 8.0 (API level 26), making testing it as simple as following the Android developer documentation guide [Andb].

6.1.3 Denial of Service Resilience

Because the proposed system is vulnerable to malicious users that take over the GO role and then refuse to properly forward messages, a means of automatically detecting and excluding such users would strengthen the security of the offline communication. It would therefore also be a topic worth investigating in the future.

References

[Anda] Save files on device storage | Android Developers. https://developer.android.com/training/data-storage/files. Accessed Thursday 6th June, 2019.

[Andb] Wi-Fi aware overview | Android Developers. https://developer.android.com/guide/topics/connectivity/wifi-aware. Accessed Thursday 6th June, 2019.

[Andc] Wi-Fi peer-to-peer overview | Android Developers. https://developer.android.com/guide/topics/connectivity/wifip2p. Accessed Thursday 6th June, 2019.

[And18] Android Developers. Android source code WifiP2pServiceImpl. https://android.googlesource.com/platform/frameworks/opt/net/wifi/+/pie-release/service/java/com/android/server/wifi/p2p/WifiP2pServiceImpl.java, 2018.

[BCLN16] Joppe W Bos, Craig Costello, Patrick Longa, and Michael Naehrig. Selecting elliptic curves for cryptography: An efficiency and security analysis. *Journal of Cryptographic Engineering*, 6(4):259–286, 2016.

[BGB04] Nikita Borisov, Ian Goldberg, and Eric Brewer. Off-the-record communication, or, why not to use pgp. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, pages 77–84. ACM, 2004.

[BLN16] Daniel J Bernstein, Tanja Lange, and Ruben Niederhagen. Dual ec: A standardized back door. In *The New Codebreakers*, pages 256–281. Springer, 2016.

[Braa] Bramble QR Code Protocol, version 1. https://code.briarproject.org/briar/briar-spec/blob/master/protocols/BQP.md. Accessed Monday 4th March, 2019.

[Brab] Bramble Transport Protocol, version 4. https://code.briarproject.org/briar/briar-spec/blob/master/protocols/BTP.md. Accessed Thursday 6th June, 2019.

[Bria] Briar - How It Works. https://briarproject.org/how-it-works.html. Accessed Thursday 6th June, 2019.

[Brib] Bridgefy Android SDK v 1.1 Quick Start Guide. https://github.com/bridgefy/bridgefy-android-samples/blob/master/README.md. Accessed Thursday 6th June, 2019.

[CCP+15] Claudio Casetti, Carla Fabiana Chiasserini, Luciano Curto Pelle, Carolina Del Valle, Yufeng Duan, and Paolo Giaccone. Content-centric routing in wi-fi direct multi-group networks. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2015 IEEE 16th International Symposium on a*, pages 1–9. IEEE, 2015.

[CMGSS13] Daniel Camps-Mur, Andres Garcia-Saavedra, and Pablo Serrano. Device-to-device communications with wi-fi direct: overview and experimentation. *IEEE wireless communications*, 20(3):96–104, 2013.

[DBL17] Jeanie Decker, Nick Brower, and Brian Lich. How Surface Hub addresses Wi-Fi Direct security issues. 2017. https://docs.microsoft.com/en-us/surface-hub/surface-hub-wifi-direct. Accessed Thursday 6th June, 2019.

[EAA09] Mohamed Elboukhari, Mostafa Azizi, and Abdelmalek Azizi. Integration of quantum key distribution in the tls protocol. *IJCSNS*, 9(12):21–28, 2009.

[FDM+12] Gábor Fodor, Erik Dahlman, Gunnar Mildh, Stefan Parkvall, Norbert Reider, György Miklós, and Zoltán Turányi. Design aspects of network assisted device-to-device communications. *IEEE Communications Magazine*, 50(3), 2012.

[Fir] FireChat - OpenGarden. https://www.opengarden.com/firechat/.

[FTH17] Colin Funai, Cristiano Tapparello, and Wendi Heinzelman. Enabling multi-hop ad hoc networks through wifi direct multi-group networking. In *Computing, Networking and Communications (ICNC), 2017 International Conference on*, pages 491–497. IEEE, 2017.

[Gol11] Steve Gold. Cracking wireless networks. *Network Security*, 2011(11):14–18, 2011.

[GSP11] Paul Gardner-Stephen and Swapna Palaniswamy. Serval mesh software-wifi multi model management. In *Proceedings of the 1st International Conference on Wireless Technologies for Humanitarian Relief*, pages 71–77. ACM, 2011.

[iee12] Ieee standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pages 1–2793, March 2012.

[JMV01] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1(1):36–63, 2001.

[KMKJR16] Ed. K. Moriarty, B. Kaliski, J. Jonsson, and A. Rusch. PKCS #1: RSA Cryptography Specifications Version 2.2. RFC 8017, Internet Engineering Task Force (IETF), 2016.

[Kri18] Paul Krill. Oracle plans to dump risky java serialization. *InfoWorld*, May 2018.

[LB07] Rich Ling and Naomi S Baron. Text messaging and im: Linguistic comparison of american college data. *Journal of language and social psychology*, 26(3):291–298, 2007.

[LSY+16] Kecheng Liu, Wenlong Shen, Bo Yin, Xianghui Cao, Lin X Cai, and Yu Cheng. Development of mobile ad-hoc networks over wi-fi direct with off-the-shelf android phones. In *Communications (ICC), 2016 IEEE International Conference on*, pages 1–6. IEEE, 2016.

[LZLS12] Lei Lei, Zhangdui Zhong, Chuang Lin, and Xuemin Shen. Operator controlled device-to-device communications in lte-advanced networks. *IEEE Wireless Communications*, 19(3), 2012.

[Mac05] John L. MacMichael. Auditing wi-fi protected access (wpa) pre-shared key mode. *Linux J.*, 2005(137):2–, September 2005.

[MEÁ10] Víctor Gayoso Martínez, Luis Hernández Encinas, and Carmen Sánchez Ávila. A survey of the elliptic curve integrated encryption scheme. *Journal of Computer Science and Engineering*, 2:7–13, 01 2010.

[MP16] Moxie Marlinspike and Trevor Perrin. The x3dh key agreement protocol. Technical report, Open Whisper Systems, Nov 2016.

[MP17] Moxie Marlinspike and Trevor Perrin. The sesame algorithm: Session management for asynchronous message encryption (revision 2). Technical report, Open Whisper Systems, Apr 2017.

[PM16] Trevor Perrin and Moxie Marlinspike. The double ratchet algorithm. Technical report, Open Whisper Systems, Nov 2016.

[Res00] Certicom Research. SEC 2: Recommended elliptic curve domain parameters. In *Standards for Efficient Cryptography*, 2000.

[Res18] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, Internet Engineering Task Force (IETF), 2018.

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

[SH16] Shubham Saloni and Achyut Hegde. Wifi-aware as a connectivity solution for iot pairing iot with wifi aware technology: Enabling new proximity based services. In *2016 International Conference on Internet of Things and Applications (IOTA)*, pages 137–142. IEEE, 2016.

[Sha14] Peter Shadbolt. FireChat in Hong Kong: How an app tapped its way into the protests. *CNN*, 2014.

[SHSA15a] Yaron Sheffer, Ralph Holz, and Peter Saint-Andre. Recommendations for secure use of transport layer security (tls) and datagram transport layer security (dtls). RFC 7525, Internet Engineering Task Force (IETF), 2015.

[SHSA15b] Yaron Sheffer, Ralph Holz, and Peter Saint-Andre. Summarizing known attacks on transport layer security (TLS) and datagram TLS (DTLS). RFC 7457, Internet Engineering Task Force (IETF), 2015.

[SY14] Ahmed A Shahin and Mohamed Younis. A framework for p2p networking of smart devices using wi-fi direct. In *Personal, Indoor, and Mobile Radio Communication (PIMRC), 2014 IEEE 25th Annual International Symposium on*, pages 2082–2087. IEEE, 2014.

[TPHH13] Sascha Trifunovic, Andreea Picu, Theus Hossmann, and Karin Anna Hummel. Slicing the battery pie: fair and efficient energy usage in device-to-device communication via role switching. In *Proceedings of the 8th ACM MobiCom workshop on Challenged networks*, pages 31–36. ACM, 2013.

[UDB+15] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. Sok: secure messaging. In *2015 IEEE Symposium on Security and Privacy*, pages 232–249. IEEE, 2015.

[Van97] Scott A Vanstone. Elliptic curve cryptosystem — the answer to strong, fast public-key cryptography for securing constrained environments. *Information security technical report.*, 2(2):78–87, 1997.

[wha17a] WhatsApp. Connecting one billion users every day, Jul 2017. https://blog.whatsapp.com/10000631/Connecting-One-Billion-Users-Every-Day. Accessed Thursday 6th June, 2019.

[wha17b] WhatsApp encryption overview. Technical report, WhatsApp, Dec 2017.

[Wi-16] Wi-Fi Alliance. Wi-Fi Peer-to-Peer (P2P) Technical Specification v1.7. 2016.

[Wi-18] Wi-Fi Alliance. Neighbor Awareness Networking Specification v3.0. 2018.

[WVNA14] Paul Wong, Vijay Varikota, Duong Nguyen, and Ahmed Abukmail. Automatic android-based wireless mesh networks. *Informatica*, 38(4), 2014.

Appendix A
Scientific Paper

A scientific paper which covers the proposed core concept has been produced. It will soon be submitted at WiMob 2019, The 15th International Conference on Wireless and Mobile Computing, Networking and Communications, and has been included on the following pages.

Keeping Connected When the Mobile Social Network Goes Offline

Øystein Sigholt*, Besmir Tola†, and Yuming Jiang†
Department of Information Security and Communication Technology
NTNU-Norwegian University of Science and Technology
Trondheim, Norway
Email: oysteils@stud.ntnu.no*, {besmir.tola,yuming.jiang}@ntnu.no†

***Abstract*—WiFi Direct is an embedded technology in a vast majority of smartphone devices running the Android operating system. As a result, it represents a promising technology that can be exploited in re-establishing connectivity among user devices in case of cellular network outages. A technique that smart devices can use to restore connectivity in situations where they are unable to connect to a cellular tower or access point, but close enough to support device-to-device communication is presented. The proposed technique envisions a combination of security layers that ensure authentication, confidentiality, and integrity of communications among end users. Each device is issued a certificate by a central authentication entity at sign up and when it is unable to connect to the server component, it will attempt to form a group with nearby devices in the same situation over WiFi Direct. Once a WiFi Direct group has been formed, the group owner will temporarily assume the role of the server, and each group member and the group owner will verify each others identity and connect using mutual Transport Layer Security (mTLS), facilitating secure communication. The approach is validated through the implementation of a mobile social application involving several mobile devices, and overheads due to the additional security features are investigated.**

***Index Terms*—Peer-to-peer Communications, WiFi Direct, mTLS, Mobile Social Networks**

I. Introduction

Broadband cellular networks are becoming the most dominant means for mobile data access world-wide. According to a recent mobility report by Ericsson [1], mobile data traffic is expected to undergo an annual growth of around 31% over the coming years. Among the top mobile application categories, video traffic followed by social network applications cover almost 75% of the total monthly data traffic. However, when a mobile user moves out of cellular coverage, or is unable to reach the central server of a service for any reason, connectivity is lost. As a consequence, Internet-based mobile applications will not be able to provide their services and the end-users will become isolated until Internet connectivity is restored, even if the users that are making use of the service, are within a few meters of each other.

Modern smartphones are equipped with a number of radio interfaces that enable wireless communication among devices in close proximity. These capabilities can be used to establish connectivity between neighboring devices even in the most remote out-of-coverage locations and in cases where cellular network outages are experienced.

WiFi Direct [2], a wireless technology allowing WiFi devices to connect directly to each other in a Peer-to-Peer (P2P) fashion, is one of the ways that nearby mobile devices can establish connectivity. It is particularly widely available due to it not requiring any specialized hardware apart from a typical WiFi radio, and easy to use as addressing can be done using the familiar Internet Protocol. In order to establish communication, a common setup involves the formation of a WiFi Direct group of peers where one of the peers acts as a software Access Point (AP) to the remaining devices. This device is referred to as the Group Owner (GO) and the other devices associated to the GO represent the Group Members (GMs). The group formation can be achieved in three different procedures denoted as *standard*, *autonomous*, and *persistent* group formation. During each of these procedures a number of actions are taken by the WiFi Direct capable devices for performing device discovery, GO negotiation, service discovery, security provisioning and address configuration [3].

WiFi Direct connections entail WiFi Protected Setup (WPS) as means to provide a secure connection among members through some manual intervention like inserting a PIN or PushButton Configuration (PBC). This way, users are able to authenticate themselves in the network. However, there are services that require stronger levels of security where data confidentiality, integrity and authenticity are of utmost importance. Mobile Social Networks (MSN) providing Instant Messaging (IM) services necessitate enterprise and automated authentication methods such as Extensible Authentication Protocol - Transport Layer Security (EAP-TLS).

Even though WiFi Direct can be used to reestablish connectivity, a social application usually depends on a central server that users trust. When connected to this server, users have confidence that their messages are delivered to their respective recipients and that no other users of the service can access their private conversations. Without access to this central server, a user in an out-of-coverage P2P context must verify the authenticity of their peers themselves.

The scope of this work is therefore to propose, implement, and experimentally validate a combination of upper layer measures that can be used to securely and easily enable authenticated communication in existing social applications, also in situations with no Internet connectivity.

The reminder of this paper is the following. Section II

presents the related work. In Section III, we illustrate the proposed system architecture, and the relative components for enabling secure and trustworthy communication over WiFi Direct. An implementation on a real testbed, composed of several smart devices running the Android operating system (OS), and how the system components interact with one another are presented in Section IV. Successively, the validation and experimental results analysis of the different security layers adopted in the architecture are illustrated in Section VI. Discussion regarding overheads, incurred due to the additional security levels, in terms of both computation and connection times are presented in Section VII. The potential and limitations of the proposed architecture in regard to connectivity, security, user experience, and overhead are discussed in Section VIII. Finally, Section IX concludes the paper.

II. Related Work

A number of commercial applications for P2P communication, applying various combinations of WiFi and Bluetooth, exist for the Android ecosystem. The most prominent, FireChat, made headlines in 2014 when it accomplished half a million downloads over a period of two weeks as Hong Kong protestors used its P2P functionality to organize efficiently even in areas with heavily congested network traffic [4]. An open source privacy-focused, decentralized, alternative to FireChat, Briar, also uses Bluetooth and WiFi to communicate in addition to the Tor network. It uses public key cryptography to manage identities and secure the communications link, but its decentralized nature and lack of a universally trusted entity makes exchanging identities a difficult problem that in practice requires the two parties to physically meet and manually exchange keys before a connection can be made [5].

A large-scale research effort by the name of the Serval Mesh aims to create an independent network by relying on WiFi devices to form a mesh network based on WiFi ad-hoc mode [6]. Unfortunately, WiFi ad-hoc mode is mostly unavailable on consumer smartphones without modifications, thus making it unsuitable for many use cases.

Shahin and Younis present a framework for P2P networking of Android devices using WiFi Direct that covers discovery, connection establishment, peer management and communication between peers in a single group [7].

Wong et al. noted that connecting Android devices with WiFi Direct uses WiFi Protected Setup (WPS) which requires manual user interaction to accept the connection prompt. They propose using WiFi Direct to create APs that advertise their Service-Set Identifier (SSID) and Pre-Shared Key (PSK) using Network Service Discovery (NSD) instead of leveraging fully fledged WiFi Direct connection establishment. Any WiFi capable client can then connect to the WiFi Direct AP like they would connect to any other WiFi AP (referenced as connecting as a legacy client in WiFi Direct terminology) without the need for users manual verification [8]. They do not however, consider the authentication aspect covered by this work.

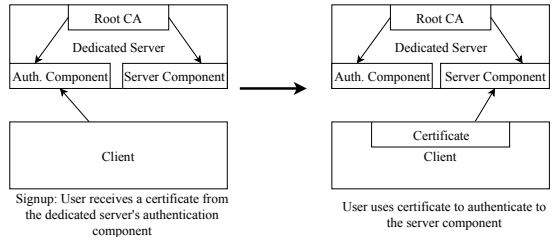


Fig. 1. A client signing up to the service by obtaining a digital certificate from the authentication component and using it to authenticate to the server component.

III. System Architecture

A system consisting of three logical components is proposed. These components can be implemented to facilitate secure communication between users both when connected to the Internet and when out-of-coverage.

A. The Authentication Component

The authentication component is the single mutually Trusted Third Party (TTP) among the members of the social network with the sole responsibility of managing the identities of the users. This component is only available over the Internet, and can therefore not be reached in out-of-coverage operation.

As seen in Figure 1, signing up to the service is done in the same fashion as in a traditional Public Key Infrastructure (PKI) system. The client generates a key pair and creates a Certificate Signing Request (CSR) that is sent to the authentication component for signing. The CSR contains the public key as well as the identity (or Distinguished Name (DN)) that the certificate is for.

The DN should contain some human readable component that users can later use to distinguish between their contacts. This can for example be a username or an email address.

After the authentication component has approved the CSR, the applicant is issued a signed X.509 digital identification certificate that contains the DN, the users public key and a signature that binds the public key to the DN. This means that the authentication component has approved the public key contained by the certificate as belonging to the specific DN also contained in the certificate. The resulting certificate can be used to authenticate to other entities in the system, offline or not as shown in Figure 1.

To revoke credentials after the fact, a Certificate Revocation List (CRL) can be maintained (see [9, Section 3.3]).

B. The Server Component

The server component is responsible for message forwarding to connected clients both over the Internet and during out-of-coverage operation, i.e., WiFi Direct operation mode, and accepts incoming TLS connections on a predefined port.

By exchanging certificates and verifying that they are issued by the authentication component, both the client and the server confirm that the other is a registered user of the service, and

learn the other party's identity. The server component then maintains a forwarding table linking the connected clients identity (public key) to the appropriate socket.

1) Out-of-coverage Operation: The server component can either run on the dedicated server (introduced in more detail in subsection IV-A) and be available over the Internet, or on a user's device in an out-of-coverage situation. When running on a GO it is also the server component's responsibility to set up and manage P2P connectivity to nearby devices. Figure 2 shows how two devices form a P2P group and reestablish connectivity if the dedicated server is unreachable.

If the user device is unable to locate any other nearby P2P devices hosting an instance of the server component, the server is instantiated on the device in out-of-coverage mode. It will autonomously form a P2P device group (middle part of Figure 2), and broadcast the information needed to connect to said group using Wong et al.'s technique [8].

When the group is formed (bottom part of Figure 2), the device will accept incoming connections from nearby devices and the server component will manage message routing just like it would during regular operation on the dedicated server.

If Internet connectivity were to be restored at any time, a device will simply detect the connectivity change and reconnect to the dedicated server, tearing down any open P2P connections.

2) Message Routing: A message packet with the senders public key, the recipients public key, a timestamp, a ciphertext and a signature is transmitted by the clients to the server when they wish to communicate with another user. The recipients public key indicates which client the message should be forwarded to and the signature (protecting the integrity of the other fields) is verified using the attached public key belonging to the sender.

Caching and retransmissions of messages are the responsibility of the client, making the operation of the server rather simple. If signature verification fails, the message is discarded. If not, the server examines its active connections and checks if a client has connected with the identity of the receiver. Successively, the message is forwarded to the appropriate client. If the server does not have an open connection to the correct recipient, the packet is also discarded. Relying on servers to cache messages would cause messages to never make it to their intended recipients if a GO were to disconnect before being able to forward it.

C. The Client Component

The client component is responsible for managing messaging and connecting to and authenticating the server component.

1) Connectivity: The client component is expected to connect to and disconnect from multiple server component instances in a single session as a device might move in and out of range of cellular coverage and P2P groups.

If not connected to a server component the client will first attempt to establish a connection to the dedicated server via the Internet. At the same time, it will start the device discovery process to locate nearby P2P groups. If the dedicated server is not reachable and no group is found, it will set up an instance of the server component and form its own P2P group.

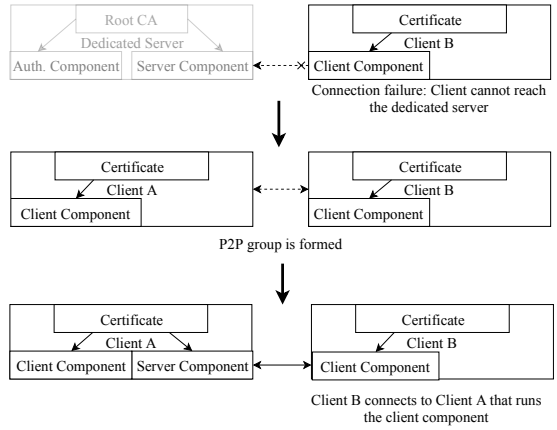


Fig. 2. A client unable to reach the dedicated server establishes a connection with a nearby device. Arrows from the certificates to the client and server indicate that these components use the certificates to authenticate when connecting to each other.

The first device to form a group will advertise its connection information so that nearby devices can discover and join the P2P group. Upon joining a group, the client will attempt to connect to the server component instance running on the GO, instead of the dedicated server.

The client enforces secure connections and will only connect to server components that offer identification certificates issued by the the authentication component over TLS.

2) Messaging: Message packets are transmitted to any server component as soon as possible. If the recipient does not acknowledge the message, the client assumes it was not delivered and caches it for retransmission. If the client connects to another server component it will immediately attempt to transmit all queued messages, otherwise it will periodically attempt to retransmit them with an exponential backoff strategy where the client periodically retransmits the message with increasing delays between attempts.

IV. Implementation

A. The Dedicated Server

The dedicated server is a traditional server reachable using the Internet. It implements both the server component to facilitate messaging and the authentication component to facilitate sign-up.

B. The Client Application

The client application not only implements the client component, but also the server component. It consists of two primary activities and one debug activity to inspect messages being sent and received.

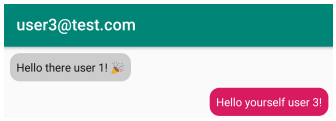


Fig. 3. Two users chatting using the **Chat Activity** as seen by `user1@test.com`.

1) The Main Activity: The main activity contains a list of a user's contacts and the current connection status. The connection status has four distinct values representing different stages of the application connection phase.

- **Setting up:** The application is loading
- **Connecting:** Looking for a server to connect to
- **Connected:** Successfully connected to a server
- **Hosting:** Unable to connect, the server component has been started and is hosting a group

Selecting one of the contacts in the list opens up the chat activity for that particular user. Pressing the info circle in the top right of the screen opens the debug activity.

2) The Chat Activity: The chat activity (seen in in Figure 3) allow users to send basic text messages to their contacts. It shows messages sent/received in chronological order. Messages are only shown if the attached signature can be verified.

3) The Debug Activity: The debug activity (seen in Figure 4) show the messages being sent from/to, or being relayed by a user. It loads the name of the users from the contact list if an entry is found corresponding to the public keys contained in the message.

It displays the state of the attached signature by verifying it using the attached sender public key, and displays a checkmark if the signature can be verified, and a cross if the verification fails. It also decrypts the message ciphertext if it possesses the private key corresponding to the recipient.

V. Cryptography

Elliptic Curve Cryptography (ECC) was selected based on the fact that it requires relatively short keys to provide strong security. It is based on the premise that it is difficult to find the discrete logarithm of an elliptic curve point [10].

Elliptic Curve encryption was done using Elliptic Curve Integrated Encryption Scheme (ECIES). To encrypt a message, a key agreement function is used to create a shared secret value based on a randomly generated ephemeral key pair and the recipients public key. A key derivation function is then used to generate symmetric keys for encrypting and signing the message contents from the shared secret. The actual message encryption is performed using a symmetric encryption algorithm and a digest function is used with the signing key to generate a Message Authentication Code (MAC). The recipient can then obtain symmetric keys to verify the MAC and decrypt the message by using the ephemeral public key and recipient private key, along with the encrypted message [11].

The MAC from ECIES only preserves the integrity of the message itself, so the Elliptic Curve Digital Signature Algorithm (ECDSA) was used to sign the entire message frame including the sender and recipient fields. An ECDSA signature is the elliptic curve variant of the Digital Signature Algorithm (DSA). It is calculated using the senders private key and a random value, and can be verified by knowing the random value and the senders public key [12].

VI. Security Validation

The following section covers an analysis of what can be learned about the operation of the sample implementation and its users by examining select network layers.

A. Data Link Layer Security

As messages are the only data frames transmitted in the WiFi Direct group, it is trivial for an unauthenticated third party monitoring the network to see which devices are transmitting messages. If the aforementioned third party is able to link the physical addresses of each device to an identity, it can keep track of when messages are sent and to/from whom.

Using Wong et al.'s technique of broadcasting the PSK of the formed network using NSD to allow devices to connect to the WiFi network without the need for manual verification prompts makes connection establishment considerably easier for the user [8]. However, this also means that anyone can use service discovery to learn the PSK. A user must therefore not be considered trustworthy based only on their ability to connect to the WiFi network.

Encrypting the PSK with a static secret as done in the original paper would only make the key marginally more difficult to recover, as the secret would need to be distributed onto every users device, from which it could be recovered.

WiFi Direct connections between devices must therefore be considered insecure channels, as an adversary must be considered able to decode transmitted frames [13].

B. Transport Layer Security

As the data link layer security is not sufficient to protect data over the air, the system relies on upper layer security. On the transport layer, the data is transferred over TLS. A number of attacks on TLS have been published that can defeat this protection [14]. It is therefore vital that both the server component and the connecting clients enforce the most recent best practices for TLS [15]. With full control over the implementation of both servers and clients, strict requirements can be enforced without worrying about compatibility.

In addition to confidentiality and integrity protection, the transport layer also provides the access control that is lacking on the lower layers. As both connecting parties require authentication of the other party using mTLS it is impossible for a user to connect without an identity from the authentication component. If attempting to connect without the appropriate credentials, the TLS handshake will fail, and the device will be unable to communicate with other users.

It can therefore be concluded that unauthorized clients are prevented from communicating with users at the transport layer, and that the data being carried is both confidential and

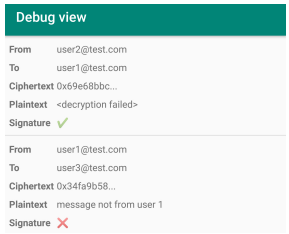


Fig. 4. Message flow as seen by the GO `user3@test.com`. Note how the GO is unable to decipher message contents not intended for itself, and how signature verification fails if the contents (in this case the *from*-field) have been altered in transit.

integrity protected due to the properties of TLS. It is however important to note that the TLS connections are not End-to-End (E2E), as they are terminated at the server component running on the GO. Upper layer measures are therefore required to protect users from a dishonest GO.

C. Application Layer Security

As seen in Figure 4, senders and recipients are visible to the GO, and signatures may be verified, but message contents cannot be deciphered without the appropriate private keys, so the server is unable to learn message contents.

A dishonest server component could potentially attempt to modify the messages it is forwarding. At the bottom of Figure 4, a malicious GO has modified the *from* field of a message packet, but was unable to correctly sign the message as it does not possess the private key of `user1@test.com`. The recipient (`user3@test.com`) therefore discards the message due to the signature verification failure.

In short, the message packet format successfully protects the messages confidentiality and integrity E2E, but does not protect the identity of the sender and recipient.

The GO can, however, choose not to deliver, delay or even deliver messages multiple times. The latter, known as a *replay attack* has an especially undesirable impact if used maliciously. In a high security context, messages that are received long after they were created should be discarded or the user should be notified as a stale message could indicate malicious activity. Duplicate messages should be discarded.

VII. Overhead

The number of steps taken to secure reliable instant messaging in this research add some overhead that must be considered in the evaluation of the proposal.

A. Connection

In the event that the Internet connection fails, a non-negligible amount of time is required to reform connectivity using WiFi Direct. Camps-Mur et al. measured the group formation delay in their 2013 overview paper of WiFi Direct, and noted that the WiFi Direct discovery mechanism introduces some randomness to the time it takes to connect to a group [3].

Timing of this delay was obtained by logging the time it took one Nexus 6P running the sample implementation to connect to another which had already autonomously formed a group and started broadcasting connection credentials using NSD. The process was repeated 500 times by a simple Bash script. It power cycled each device, opened the application on one device, and waited for it to become GO before launching the application on the other device.

	Mean μ	**Median**	**Standard Deviation σ**
Discovery (NSD)	3.42	3.07	1.17
Connection	2.03	1.90	0.46
Total	5.45	5.00	1.54

TABLE I
Seconds used to discover credentials over NSD and connect to a WiFi Direct group as a legacy client.

RSA Key Strength	Message Size	EC Curve	Message Size
2048 bit	1028 bytes	secp224r1	192 bytes
4096 bit	2052 bytes		
		secp384r1	292 bytes

TABLE II
Message packet size in bytes given various key sizes sorted by strength. The named curves are specified in SEC 2 [17].

After a group has been formed, it takes around five seconds (discovery delay plus connection delay) for the first GM to discover the broadcasted credentials and join as seen in Table I. This new client goes through the discovery process which can be expected to take three to four seconds, and uses the credentials discovered to connect to the group.

The test implementation experienced the same issues regarding undesirable NSD behavior mentioned in Wong et al.'s original paper [8]. In some instances, one or more of the test devices did not discover NSD broadcasts from other devices until they had been power cycled. The client then sees the already existing WiFi Direct group, but is unable to identify it as a group offering the chat service and to obtain the PSK required to connect. It then forms and advertises its own group resulting in two isolated groups in the same area competing for members.

B. Messaging

Using a message format with significant additional data results in additional data transfer. Most of the overhead comes from transmitting two full public keys (sender and recipient) with every message. Table II summarizes the size of a single message packet of various key sizes assuming 32 bit time stamps and fixed cryptographic parameters with compressed keys. The transmitted message contains 29 bytes of data, representing a typical instant message [16].

VIII. Discussion

A. Connectivity

Restoring connectivity by using WiFi Direct in the event of an Internet outage does come with set up time, but may still be suitable for asynchronous applications such as IM and file transfers.

B. Security

Issuing credentials to users in the form of digital certificates for use in asymmetric cryptography enables them to successfully authenticate one another with TLS during out-of-coverage operation.

However, as it requires the user to be online at the time of sign up so that their digital certificate can be signed by the authentication component it might be a hindrance that prevent a new user from starting to use the application.

The overhead introduced by using strong cryptography is not insignificant, but this can be mitigated by choosing a cryptosystem with smaller key sizes.

If the TLS connection was not terminated at the GO, but GMs were allowed direct connections to each other, some of the application layer measures designed to protect the client from a dishonest server could have been avoided. However, this would have added to the implementation complexity, as the GO acting as a server makes out-of-coverage operation very similar to online operation. The fact that the server component code base can be used on both the dedicated server and on the GO makes the proposed system easy to adopt, but requires an application layer chat protocol.

The basic chat protocol that has been proposed provides confidentiality, integrity, authentication and is resilient against messages arriving out-of-order, but lacks more sophisticated properties that further protect users in edge cases such as private key compromise.

The chat protocol protects the contents of messages from a malicious GO, but the system does not, in its current state, attempt to detect a Denial of Service (DoS) type attack where the GO refuses to forward messages. Detecting this and electing a new GO automatically would make the system more resilient and useable for example in the case where a current Internet outage is a deliberate act aimed at disturbing communications. We leave the investigation of these edge cases as future work enhancements.

C. Overhead

As WiFi Direct supports the same speeds as typical WiFi it is assumed that it provides more than sufficient throughput to carry a significant volume of these messages. In online mode, however, a server might have a much larger number of connected users. Keeping the message size low is therefore in the best interest of a developer to minimize the costs associated with bandwidth. Sending full public keys of both the sender and recipient with every packet adds significant overhead that could have been avoided by a different chat protocol design.

IX. Conclusion

When smartphone applications are unable to connect to the Internet, many useful services become unavailable. In some cases, these services can be restored by communicating with nearby devices in a P2P fashion. However, services that exploit such technologies need to ensure security in the communication between nearby devices.

This work proposes an approach that ensures authenticated communication with confidentiality and integrity protection among peers. The approach enables devices to fetch security credentials in the form of digital certificates for use in asymmetric cryptography during sign up to a service. When unable to reach the centralized server, devices may use WiFi Direct to discover and connect to nearby devices, and mTLS to authenticate them while also setting up a secure data channel. A messaging protocol that may rely on the user's digital certificates must be used to protect message confidentiality and integrity during transfer over this channel.

This proposed design has been validated through implementation of a basic chat application for the Android OS. It makes use of a simple chat protocol that protect message confidentiality and integrity provided that the user's credentials are not compromised at any point in time. This has been validated by examining the transmitted data at various layers of the OSI model.

The chat protocol lacks some advanced cryptographic properties such as forward secrecy, raising the question of whether or not current state of the art chat protocols can be adapted to a P2P scenario.

The main drawback to the proposed system is that the user is required to be online at the time of sign up, and that the user must trust the centralized server to be honest and not issue false credentials.

References

[1] Ericsson, "Ericsson Mobility Report," november 2018.

[2] Wi-Fi Alliance, "Wi-Fi Simple Configuration Technical Specification v2.0.6," 2018.

[3] D. Camps-Mur, A. Garcia-Saavedra, and P. Serrano, "Device-to-device communications with wi-fi direct: overview and experimentation," *IEEE wireless communications*, vol. 20, no. 3, pp. 96–104, 2013.

[4] P. Shadbolt, "FireChat in Hong Kong: How an app tapped its way into the protests," *CNN*, 2014.

[5] "Briar User Manual," accessed June 6, 2019. [Online]. Available: https://briarproject.org/manual/

[6] P. Gardner-Stephen and S. Palaniswamy, "Serval mesh software-wifi multi model management," in *Proceedings of the 1st International Conference on Wireless Technologies for Humanitarian Relief*. ACM, 2011, pp. 71–77.

[7] A. A. Shahin and M. Younis, "A framework for p2p networking of smart devices using wi-fi direct," in *Personal, Indoor, and Mobile Radio Communication (PIMRC), 2014 IEEE 25th Annual International Symposium on*. IEEE, 2014, pp. 2082–2087.

[8] P. Wong, V. Varikota, D. Nguyen, and A. Abukmail, "Automatic android-based wireless mesh networks," *Informatica*, vol. 38, no. 4, 2014.

[9] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, "Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile," IETF, RFC 5280, May 2008.

[10] S. A. Vanstone, "Elliptic curve cryptosystem — the answer to strong, fast public-key cryptography for securing constrained environments," *Information security technical report.*, vol. 2, no. 2, pp. 78–87, 1997.

[11] V. G. Martínez, L. H. Encinas, and C. S. Ávila, "A survey of the elliptic curve integrated encryption scheme," *Journal of Computer Science and Engineering*, vol. 2, pp. 7–13, 01 2010.

[12] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *International journal of information security*, vol. 1, no. 1, pp. 36–63, 2001.

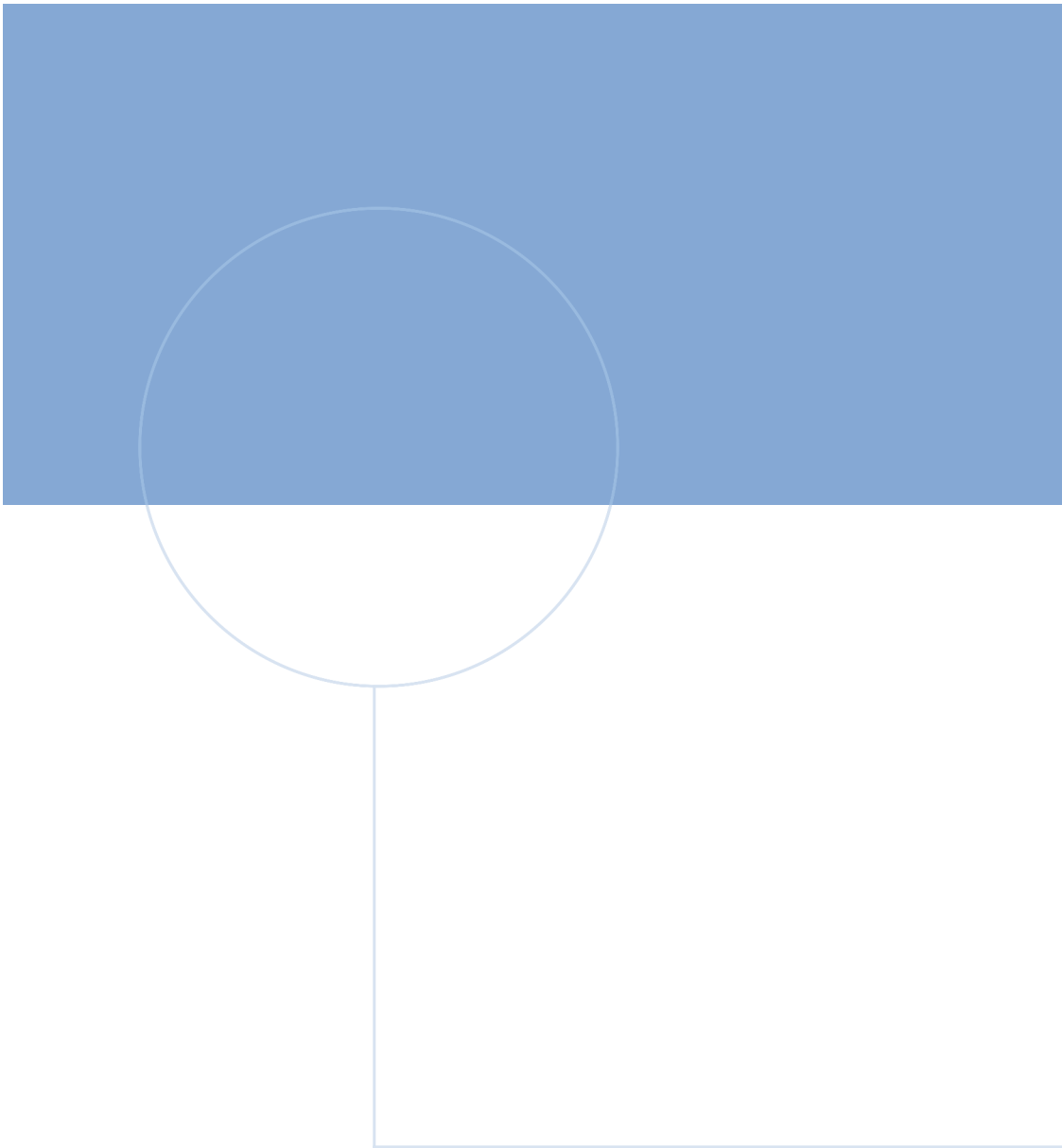
[13] J. L. MacMichael, "Auditing wi-fi protected access (wpa) pre-shared key mode," *Linux J.*, vol. 2005, no. 137, pp. 2–, Sep. 2005.

[14] Y. Sheffer, R. Holz, and P. Saint-Andre, "Summarizing known attacks on transport layer security (TLS) and datagram TLS (DTLS)," IETF, RFC 7457, 2015.

[15] ——, "Recommendations for secure use of transport layer security (tls) and datagram transport layer security (dtls)," IETF, RFC 7525, 2015.

[16] R. Ling and N. S. Baron, "Text messaging and im: Linguistic comparison of american college data," *Journal of language and social psychology*, vol. 26, no. 3, pp. 291–298, 2007.

[17] Certicom Research, "SEC 2: Recommended elliptic curve domain parameters," in *Standards for Efficient Cryptography*, 2000.



NTNU
Norwegian University of
Science and Technology