SECURING MOBILE VOIP PRIVACY WITH TUNNELS

MARTIJN HOOGESTEGER

Master Thesis

Design and Analysis of Communication Systems Group
Faculty of Electrical Engineering, Mathematics and Computer Science
University of Twente
Master Computer Science

October 14, 2016

Martijn Hoogesteger: *Securing Mobile VoIP Privacy with Tunnels,* Master Thesis, © October 2016

SUPERVISORS:
Ricardo de O. Schmidt
Aiko Pras
Mark Prins (*Ministerie van Binnenlandse Zaken en Koninkrijksrelaties*)
Erik Reitsma (*Ministerie van Binnenlandse Zaken en Koninkrijksrelaties*)

"So long, and thanks for all the fish."

Douglas Adams,
The Hitchhiker's Guide to the Galaxy

ACKNOWLEDGMENTS

I have a great many people to thank for their guidance and support during the production of this thesis. Without them, I don't know where this would have ended up!

First and foremost, I would like to thank my supervisors: Ricardo, Aiko, Mark and Erik. Ricardo and Aiko have supervised my work long before I started with a master thesis, and their guidance throughout the years has led me here. I'd like to thank Ricardo in particular, as he was always there for a quick brainstorming session which left me very motivated, even if it was over Skype dealing with 8 hour time differences. I also thank Erik and Mark for their hospitality and advice on my work and direction. Also a thanks to Arnoud, who was not officially my supervisor, but was always up for a discussion. It was great working with all of you.

Finishing your masters, writing your thesis, can be quite challenging at times. I would like to thank everyone that helped me get through these challenges, either directly or indirectly. In particular I have to thank Ralph, who helped me immensely with his hospitality; getting me through one of the larger challenges: Enschede's distance to the world. I owe you one mate! I also thank Kimberly for all of her support during this time, and bearing with me and my strange schedules.

Lastly, I would like to take this chance to thank everyone I've gotten to know during my time as a student at the UT. People from Inter-*Actief*, Euros Zeilen and all the people I met along the way; it wouldn't have been the same without you all, and it was great.

Cheers!
Martijn Hoogesteger

CONTENTS

1 INTRODUCTION

Privacy within technology is increasingly becoming an issue for consumers. For example, Apple has mounted a large effort to protect the privacy of their customers after the FBI demanded them to compromise a suspect's phone to access its data [1]. While this in itself is not a large issue, the underlying principle of a government demanding a company to reduce the security of its product definitely is. It would eventually affect many users, and infringe upon their privacy. This issue has led to many discussions, making customers aware of their privacy and how it is secured. As a result of this, services like WhatsApp have enhanced end-to-end security through encryption [2].

Privacy concerns relate mostly to personal information and (private) communications. Our communications have shifted more and more from fixed line telephony towards Internet networks as bandwidth has increased and mobile (LTE (Long Term Evolution)) networks have become widespread. VoIP (Voice over Internet Protocol) is now slowly taking over our telephony needs, just as email has largely taken over our message-sending needs. Over the past years, more and more VoIP traffic has be seen going through traditional IP networks [3, 4].

With mobile networks becoming more and more capable, the shift towards mobile VoIP has started. More and more mobile VoIP solutions are available and used more frequently by consumers [5]. However, there are some caveats that come in the way of this trend, both technological and economical. At the moment, many mobile networks provide enough bandwidth to support a VoIP connection, but these communications can be unreliable, causing problems in the communication. These problems are, for example, delay and jitter in the connection, quickly reducing the perceived quality of the conversation greatly, making a VoIP call worse than a standard call. Costs and data usage limitations of data on mobile networks can reduce the advantage of VoIP running over the Internet instead of standard telephony lines. Other issues such as keeping SIP (Session Initiation Protocol) channels open, discussed further in Section 2.1, also make mobile VoIP solutions difficult.

With voice communication shifting towards Internet networks, the risk that our privacy can be compromised increases. IP networks are easier to abuse than traditional PSTNs (Public Switched Telephone Network), due to the fact that many more attack vectors already exist for which the technology is easily attained and used and no special hardware is necessary. RTP (Real Time Protocol) and SIP are

traditionally not secured, and extra protocols have to be used for this. Many different reasons exist to hack into VoIP networks, such as theft, corporate espionage and information warfare. These attacks can be performed by a variety of attackers, from unskilled casual hackers to highly skilled foreign intelligence agencies [6].

To guard the privacy of VoIP communication, especially when used in mobile networks, the security of the connection has to be guaranteed. In this work we look into multiple ways to secure VoIP connections and explore alternative of using a secure VPN (Virtual Private Network) tunnel through which VoIP can be set up. We describe how this can be practically used and discuss the viability and caveats of such a solution.

In all, we can see that VoIP is indeed an important technology, and is becoming more and more prevalent in networks. It is used in corporations and by consumers, and is shifting more and more towards mobile phones, replacing the old standard telephony system. We can also see that privacy is becoming a larger concern for consumers, and companies are making changes based on this. This privacy concern is not just towards attackers, but also towards companies handling our data.

1.1 RESEARCH PROBLEM AND QUESTIONS

In many VoIP solutions, it is not clear how secure communications actually are. Using SIP and RTP is definitely not secure, and their secure counterparts might still leak some information. The communications happen over a widely and easily accessible network however; contrary to PSTNs, IP networks are much easier to infringe upon, so the communications are exposed. By using a VPN to protect communications, not only VoIP traffic is protected, but also all other traffic that makes use of the secure tunnel. This can provide an integral solution, if it is feasible.

1.1.1 *Questions*

In this thesis, this problem will be addressed, and the question will be answered:

How to secure VoIP to protect privacy on mobile phones?

To answer this main question, these sub-questions will be addressed:

1. **What are VoIP communication characteristics?**

   To say anything about VoIP, we first look into what VoIP protocols actually look like. Different protocols and technologies will be analyzed and traffic characteristics measured. VoIP infrastructure will be set up as a testbed.

2. **What are the differences between mobile VoIP solutions?**

   Currently, many solutions for VoIP already exist on the mobile app markets. These applications will be reviewed and tested on efficiency and security. Based on this, we can gain some idea of the technologies currently used and how privacy is protected in these applications.

3. **How does a VPN tunnel affect different types of traffic?**

   To see how we can safely tunnel traffic through a VPN, we have to look into what the VPN does with the data. A VPN network will be set up as a testbed. While performance analysis shows small differences between OpenVPN and IPSec, tests can be done to check performance depending on configuration, packet size, throughput and other variables that might occur in traffic. This will provide a basis for setting up VoIP through a VPN.

4. **How can VoIP be secured with a tunnel?**

   To secure VoIP, a tunnel connection can be used. This provides more features and less adaptation to software than other solutions. How this can be done and how secure such solutions are will be analyzed and discussed.

2 BACKGROUND

2.1 VOICE OVER IP

Traditionally, calls have been made over PSTNs. Nowadays, with Internet being widespread, VoIP is becoming the standard to accomplish this. Because the infrastructure is mostly present (Ethernet networks), many businesses implement an internal VoIP network for communication.

VoIP generally works with a signaling protocol such as SIP and a media protocol such as RTP. This separates a control channel and the media channel. With SIP, one person can send an invitation to someone else, and receive signals that a phone is ringing. When the phone is picked up, this is signaled back and a media session is set up through which the voice data is sent back and forth. SIP also establishes the settings that should be used for the media channel, such as encoding options. Signaling traditionally worked with the H.323 protocol, but a shift towards SIP occurred as it is generally more flexible and inter-operable.

VoIP networks can provide many extra features besides making calls and transferring the voice data. This is one of the advantages over a PSTN, besides the lower costs. Beside voice, other multimedia can be sent over the RTP channel as well.

However, the standard way to set up VoIP, using SIP and RTP, is completely unsecured. Improvements upon these protocols exist, and variations on SIP and RTP have been engineered. Below, we discuss the SIP and RTP protocols.

2.1.1 *Signaling*

To indicate that a call is incoming, to receive it and to negotiate settings, a signaling protocol should be used for VoIP. The most common protocol for this is SIP (Signal Initiation Protocol).[7]

SIP works on either TCP (Transmission Control Protocol) or UDP (User Datagram Protocol). The RFC specifies that either is possible, and TCP is only required for large messages. In practice, UDP is used mostly on networks with many clients, and TCP on networks with less, as TCP creates more overhead. SIPS, the secure variant of SIP proposed in the RFC, works by sending the SIP messages encrypted via TLS (Transport Layer Security). This requires SIP to work over TCP, which is not always good for performance.

The SIP Protocol is based on an HTTP-like model where every request results in a reply from the server. Header fields are not fixed and depend on the type of message that is sent. To set up a call for example, Invite packets are sent to the receiving SIP phone, and Ringing and OK packets are sent back. After such a setup phase, a media session is set up, which is mostly an RTP channel. [7]

The payload of a SIP packet can vary depending on the usecase. For example, for exchanging information about a session, the SDP (Session Description Protocol) can be used. SDP defines a standard for conveying information about the media, addresses and other metadata that might be necessary for the channel. [8]

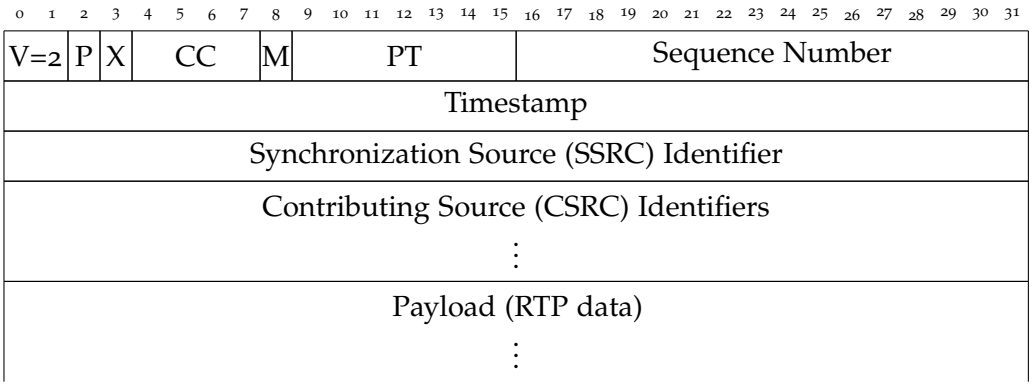
The header of an invite packet would look like this:

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bK776asdhds
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.com
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142
```

In the use-case of VoIP, in general there is a central server called the SIP registration server. Clients register on this server with their identity (a username, fore example). Other clients that register on this server can then invite these clients for a VoIP session. Sometimes, registration is done via a larger infrastructure where payments can also be made for these voice calls. The SIP server can also connect to old POTS (Plain Old Telephone System) networks via relays.

2.1.2 *Media*

To handle the call, the data can be sent in numerous ways. The most standard way to do this is via RTP (Real Time Protocol). RTP is a protocol that can deliver all sorts of media over a channel that is supposed to be streamed and is time-sensitive. RTP also specifies the sub-protocol RTCP, a control protocol for RTP. RTP and RTCP are used in VoIP as the media channel to transfer the voice data. [9]



The characteristics of the RTP channel depend on the type of data that is sent through. In the case of voice data, characteristics also depend on the type of encoding used. There are many different encoding types available. The standard types are defined in an RFC and include voice encoding options such as G722, G723, GSM, L16 (uncompressed) and MPA. For video, encoding such as JPEG, H263 and MPV exist [10]. IANA (Internet Assigned Numbers Authority) specifies other supported encoding options as well, but registering new ones is not possible any more. Media types can be registered as MIME (Multipurpose Internet Mail Extensions) types for RTP now.

While RTP packets simply carry encoded data with timing and sequence information, RTCP packets control higher level information. They can provide synchronization for multiple RTP streams and monitor traffic information such as the QoS (Quality of Service) of the media channel.

2.1.3 *Attacks on SIP and RTP*

While the concept of SIP-based VoIP systems is quite simple (SIP to signal, RTP to send data), this simplicity allows for many security vulnerabilities. SIP does not provide any Confidentiality, Integrity or Availability in its standard form. There are many (basic) attacks on SIP and RTP because of this. [11, 12]. Besides the protocol being unsafe, SIP implementations are not always robust and secure, providing even more vulnerabilities [13].

2.1.3.1 *Hijacking*

As SIP is usually based on UDP, no connection is set up and messages are easily blocked and forged by an attacker. The attacker can then fake a registration request as many times as he likes, for any of the users that can register. If successful, an attacker takes over this user's connection to the VoIP proxy and calls will be routed to him. Authentication attacks against SIP client also exist and result in the same type of hijacking [14].

This can be used to reroute calls to the attacker, but in the case where an account can have multiple registrations per account, an attacker could also misuse accounts to make calls without incurring costs.

2.1.3.2 *Proxy in the middle*

As phones generally connect to a proxy over UDP without any strong encryption and no authentication from the proxy to the user, it is easy to impersonate a proxy and collect data on calls. From this basic attack, many options for further attacks are possible, such as monitoring calls, impersonating users, forging messages, etc.

Such a proxy can be inserted in the network in numerous ways. An attacker could circumvent SIP traffic to the malicious proxy by manipulating the DNS (Domain Name System), routing the traffic based on the configured domain name on the SIP phones. Many MitM (Man in the Middle) attacks work to establish such a proxy, including ARP (Address Resolution Protocol) spoofing or even reconfiguring the phones directly.

2.1.3.3 *Forging and manipulating messages*

Because no strong authentication exist for the SIP messages, many different attacks can be done by forging these messages. A very basic attack using this method is to incur a DoS (Denial of Service) by flooding the network with INVITE messages or forging specific requests that incur a high resource usage in specific SIP implementations, which are sensitive to such attacks[15]. Messages can be manipulated and changed because SIP does not provide any integrity mechanisms by default. Other ways to disrupt services via message manipulating include ending calls prematurely, and changing many SIP configuration options.

2.1.3.4 *Billing attacks*

In commercial VoIP systems, clients pay for the usage of the network and generally pay per minute or through a subscription. The billing information for VoIP is sometimes also sent via SIP messaging. Attacks on these messages are called billing attacks, and can cause overbilling on users. By dropping BYE packets from the client, sessions can last indefinitely long without the user knowing. The same kind of situation happens when the client is sent a BUSY packet (which is not SIP authenticated) when in reality the session was set up legitimately. An other attack involves reusing authenticated INVITE packets to set up connections without the client's knowledge [16].

2.1.3.5 *Final considerations*

Standard SIP and RTP are not secure, and many attack vectors exist against them. Other VoIP systems, using proprietary protocols might therefore be safer. However, even Skype, which is widely used by consumers and uses an entirely different and proprietary protocol, is not entirely secure [17].

2.2 SECURE CHANNELS

In communications it can be necessary to keep certain information private. To keep this information from potential eavesdroppers, we can use a secure channel. A secure channel protects the information in a way that an eavesdropper cannot learn what is communicated. In addition to this, it protects the data from any manipulation, so that the information that is communicated is guaranteed to be genuine.

In addition, the secure channel can try to reduce information that can be leaked via covert channels [18]. These channels use meta-fields such as headers that are sent with the data to carry (confidential) information. More on this in section 2.4.2.2.

With security becoming a large factor in technology, many techniques exist nowadays that provide a form of security on connections such as encryption, authentication, and integrity checks. These techniques can be quite fast and efficient, but typically in return for lesser security guarantees. Tunneling VoIP through a secure channel mitigates many (if not all) of the problems mentioned in section 2.1.3. If implemented correctly it can protect against almost all forms of DoS attacks and can provide strong authentication for clients and servers. A fast and reliable encryption scheme is definitely needed however, as delays in voice communication can quickly cause the connection to become unusable.

There are many encryption schemes that exist, which can generally be categorized into asymmetric and symetric encryption schemes. In asymmetric encryption schemes a pair of keys called the public and private keypair is used. The public key can be freely exchanged and used to encrypt data which is only viewable for the keeper of the private key. While this makes key exchange very easy (the public key can be freely sent over insecure channels), this type of system is almost three orders of magnitude slower than its symmetric counterpart [19]. In symmetric encryption schemes, a shared secret key is used to encrypt and decrypt messages. While these schemes need some way to exchange this key before communications can happen, they are generally much faster [20].

VPNs are one of these solutions to provide a secure channel for VoIP [21] and often provide even more functionality such as authentication. Other ways to secure VoIP connections include the specially designed SRTP (Secure Real-Time Protocol) (with ZRTP) protocol.

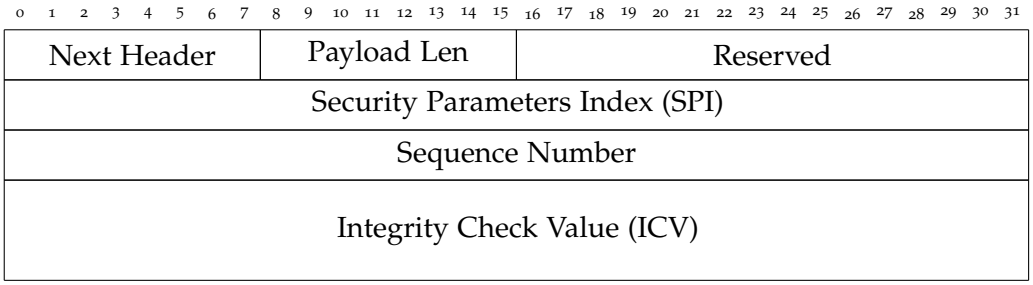
2.2.1 *Virtual Private Network*

A virtual private network connects multiple points (an enduser or subnet) of a network together via an extra layer, which can be thought of as a tunnel, through which the data is sent. This tunnel can provide extra functionality depending on what technique is used such as authentication, encryption, etc. VPNs can work on many different layers of the OSI model, clearly categorizing different VPN solutions.

2.2.1.1 *Internet Protocol Security*

Based in the network layer, the IPSec (Internet Protocol Security) protocol can provide IP packets peer authentication, integrity, encryption and replay protection. It supports two header modes, AH (Authentication Header) and ESP (Encapsulating Security Payload). Where AH provides integrity and origin authentication, ESP also provides confidentiality. ESP does this by encrypting the entire inner IP packet. Because it functions on the IP Layer, it can easily protect all application level data without application-specific adjustments.

AUTHENTICATION HEADER The AH can provide integrity for all IP fields which are not mutated during transit, such as TTL or header checksums. It has IP Protcol number 51. It does this with the following datagram [22]:



The fields in this header all serve different purposes:

NEXT HEADER The IP protocol number of the packet after this header. E.g. 4 for IPv4, 41 for IPv6.

PAYLOAD LEN Length of the AH payload in 32-bit words.

RESERVED Not used, must be all zeros.

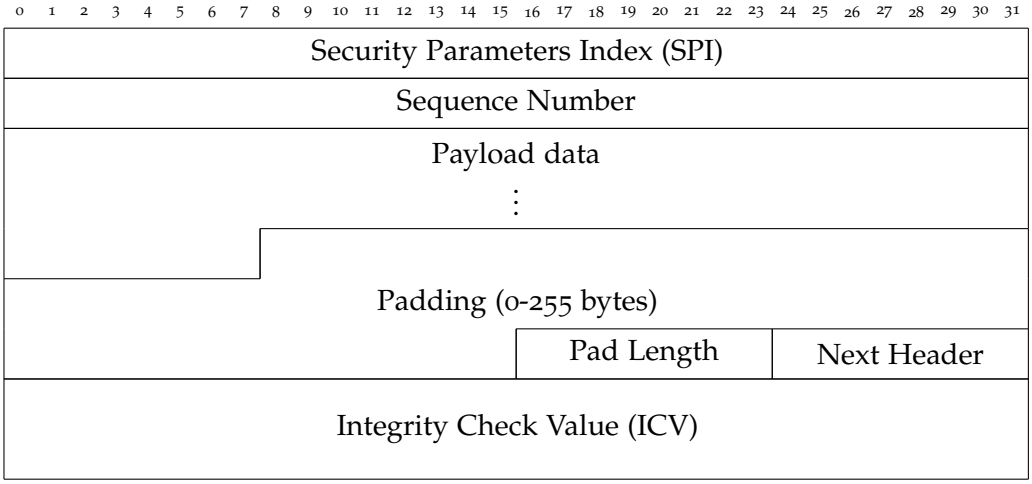
SPI A 32-bit value to identify the SA (Security Association) with this connection

SEQUENCE NUMBER An increasing counter value for every packet. Used against replay attacks.

ICV Result of the Integrity check value calculation, for which multiple algorithms exist. Can be padded so the header is aligned

on 32-bits for IPv4 or 64-bits for IPv6. For authentication and integrity.

ENCAPSULATING SECURITY PAYLOAD ESP also encrypts the entire IP packet that should be sent. By doing this, it also removes the ability to change the mutable fields that are left in AH. This can cause some problems in routing (especially with a NAT (Network Address Translation) device), but there are ways to mitigate this. It has IP Protcol number 50 and encapsulates the data in the following datagram [23]:



The fields in this header all serve different purposes:

SPI A 32-bit value to identify the SA (Security Association) with this connection

SEQUENCE NUMBER An increasing counter value for every packet. Used against replay attacks.

PAYLOAD DATA The contents of the IP packet that are now encapsulated.

PADDING A padding can be added to align the whole plaintext block on 32 or 64 bits.

PAD LENGTH The length of the padding that was added.

NEXT HEADER The IP protocol number of the contents that were encapsulated.

ICV Result of the Integrity check value calculation, for which multiple algorithms exist. Can be padded to the header is aligned on 32-bits for IPv4 or 64-bits for IPv6. For authentication and integrity.

IPSEC ENCRYPTION SCHEMES Different hashing and symmetric encryption schemes are available for IPsec, as defined in RFC7321 [24].

They are: AES-CCM, AES-GCM, AES-CBC, AES-CTR, 3DEC-CBC. For authentication, HMAC-SHA1-96, AES-GMAC and AES-XCBC-MAC-06 can be used.

AES-GCM is recommended for IPSec payloads as it binds Authentication and Encryption well, and provides good performance [25]. It is recommended in RFC6379 [26] in the proposed cryptographic suites. However, IPSec VPNS can still have security issues. For example, the VPN can leak information in covert channels [27].

2.2.1.2 *OpenVPN*

A popular VPN solution working on the application layer is OpenVPN[28]. It is an open-source VPN solution, and as a result, many different variations of it exist. The advantage of the open-source nature of this VPN solution is that it supports a large range of encryption and authentication schemes. The main implementation supports either OpenSSL or mbed TLS (formerly PolarSSL) since version 2.3. As OpenVPN runs on the application layer, it does not require access to the IP-stack and can run in userspace, which is an added ease of use.

Under PolarSSL [29], OpenVPN supports the following encryption schemes: AES, Blowfish, 3DES, DES, ARC4, Camellia and XTEA. These are supported under many different modes of operations: ECB, CBC, CFB, CTR, GCM, CCM. Based on PolarSSL, the OpenVPN-NL version of OpenVPN[30] is especially tailored to guarantee a certain level of security. It is stripped of insecure features and only supports AES-256-CBC for encryption and SHA256 as a message digest. The DH (Diffie-Hellman) group is required to be 2048 bits.

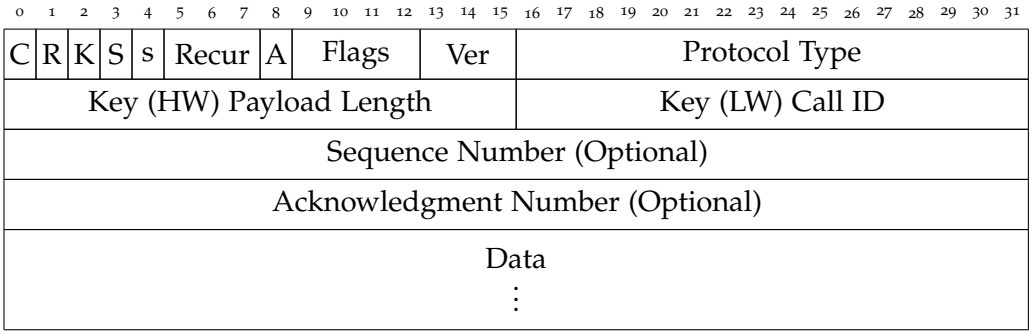
In comparison with IPSec VPNs, OpenVPN does not perform better, but is easier in its use. However, IPSec has been in use much longer, and support in hardware is better, which might explain the small performance advantage it has over OpenVPN[31]. It is not clear how packet size affects this performance.

2.2.1.3 *Point-to-Point Tunneling Protocol*

Defined in an 1999 RFC, PPTP (Point-to-Point Tunneling Protocol) was one of the first widely used VPN protocols. It was implemented with many Windows installations and is still available today. However, it is generally less secure than many alternatives such as IPSec or OpenVPN. [32]

The Microsoft implementation of the PPTP protocol has also been deemed as insecure by several research papers including one from 1998 [33, 34] when it was first analyzed. It was also concluded in analysis and research by [35, 36]

PPTP uses many different control packets. The data is sent via a modified GRE (Generic Routing Encapsulation)[37] header and a PPP (Point-to-Point Protocol) data packet.



Fields in PPTP:

C Checksum flag; GRE Header; set to 0.

R Routing flag; GRE Header; set to 0.

K Key flag; GRE Header; set to 1.

S Flag if sequence number is present. 1 if payload is included.

s Strict source route flag; GRE Header; set to 0.

RECUR Recursion control flag; GRE Header; set to 0.

A Flag if acknowledgement number is present.

FLAGS GRE Header; set to 0.

VER 1 to indicate enhanced GRE.

PROTOCOL TYPE Set to 0x880B.

PAYLOAD LENGTH First half of GRE Key field used for payload length.

CALL ID Second half of GRE Key field used for Peer's Call ID (session identifier).

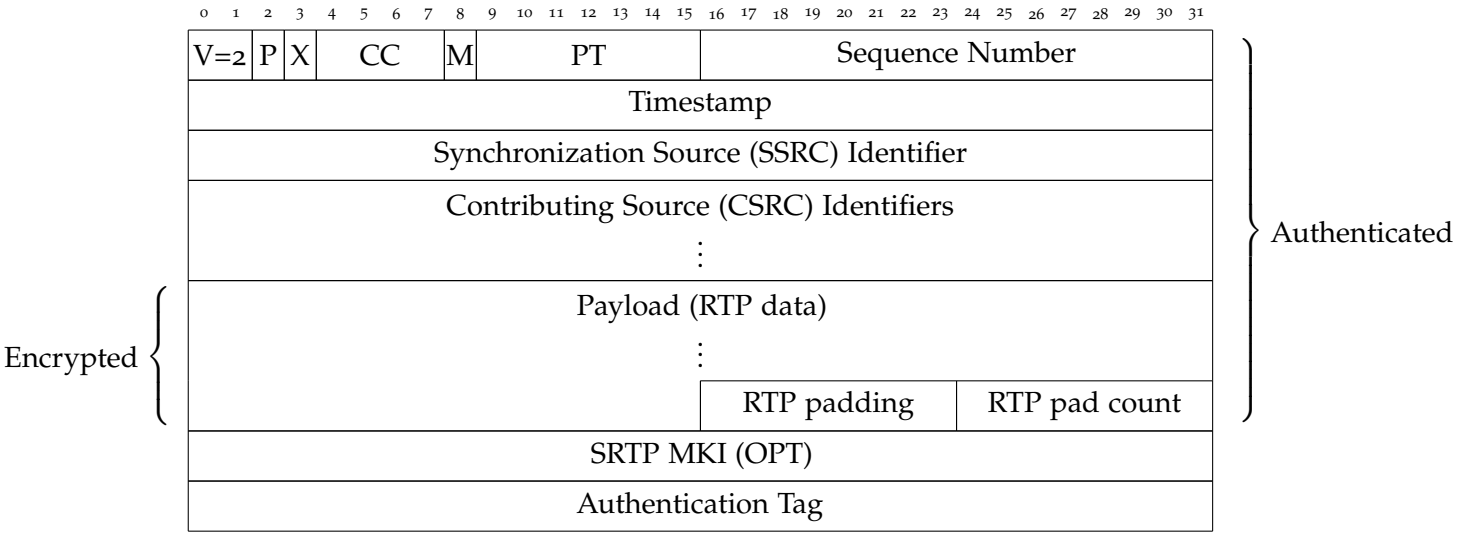
SEQUENCE NUMBER Sequence number for the payload

ACKNOWLEDGMENT NUMBER Sequence number for the highest GRE packet received by the sending peer for this session.

DATA The data contains a PPP data packet.

2.2.2 *Secure Real-time Transport Protocol*

SRTP (Secure Real-time Transport Protocol) is the secure variant of RTP (Real-time Transport Protocol). It provides encryption, authentication, replay protection and integrity to the RTP protocol.[38]



Fields in SRTP:

VERSION The version of the RTP protcol used

PADDING Bit flag to indicate padding is used

EXTENSION Bit flag to indicate extension headers are used

CSRC COUNT Number of CSRC identifiers included

MARKER Bit flag usage depending on profile

PAYLOAD TYPE The type of data that is sent (audio/video/encoding/etc)

SEQUENCE NUMBER Random starting number and increased per packet sent

TIMESTAMP Sampling instant of the first octet of data that is sent

SSRC Identifier for the synchronization source. Random per RTP connection.

CSRCS Other synchronization contributor identifiers.

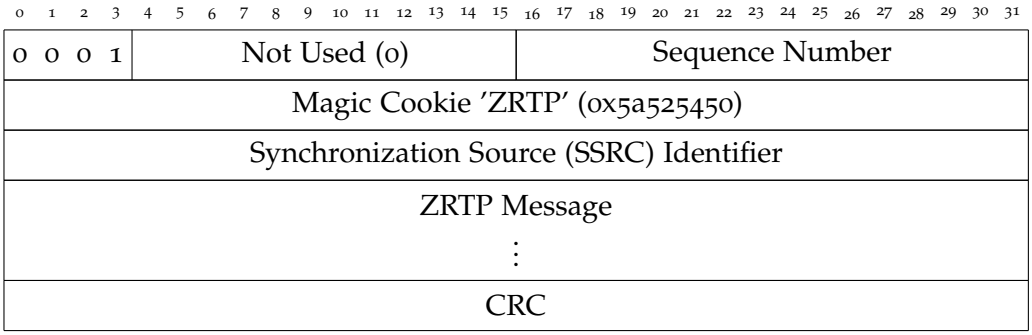
PAYLOAD The data that is sent. Type defined by the Payload Type header field.

PADDING Padding to align the payload for encryption

SRTP MKI Master Key Identifier used by key exchange management to identify master key from which session keys were derived.

AUTHENTICATION TAG Authenticates the data and header with a checksum.

ZRTP As SRTP does not have native key exchange built in, ZRTP was created. It is named after the main creator (Zimmermann) and RTP, the protocol for which it is designed. It provides a Diffie-Hellman key exchange to agree on a session key and other parameters to set up the SRTP session for VoIP specifically.



2.3 MOBILE VOICE OVER IP

In mobile systems, Android has become the largest and most widely used Operating System [39] for smartphones. It is also very easy to develop on and is a very flexible OS, which is why it is facilitating research.

As VoIP is generally a cheaper option than normal phone connectivity, the demand for a mobile VoIP solution is definitely there. Many such solutions exist nowadays on the mobile app markets. These apps generally work very well on reliable WiFi connections, but tend to have degraded call quality when used on mobile networks. These have grown over the past years for better coverage and greater bandwidth, but as VoIP is very sensitive to delays and jitter, some problems still exist there. In time, these problems are likely to be resolved.

2.3.1 *Android*

Android devices range widely in specifications. To have some baseline comparison, we take the top 6 android phone brands and specifications of their high-end phones. These specifications can give an indication of the capability of these phones at the current moment. To compare we take the chipset, the CPUs on the chipset, RAM, GPU and battery size[40]. An overview can be seen in table 1.

Based on the **chipsets**, many of these phones are similar and all use an octacore chipset (except the Moto G). Two out of six use the Snapdragon 810 chipset, which is similar to the Exynos 7 chipset used in the S6 (although the latter is slightly faster). The LG uses a slightly older version of this chipset, and the P8 uses a Kirin chipset which

Table 1: Smartphone specifications

Brands	Samsung	HTC	Huawei	Sony	LG
Type	S6	One M9	P8	Xperia Z5	G4
Chipset	Exynos 7 Octa	Snapdragon 810	Kirin 930/935	Snapdragon 810	Snapdragon 808
CPUs	1.5GHz A53, 2.1GHz A57	1.5GHz A53, 2.0GHz A57	2.0GHz A53, 1.5GHz A53	1.5GHz A53, 2.0GHz A57	1.44GHz A53, 1.82GHz A57
RAM	3GB	3GB	3GB	3GB	3GB
GPU	Mali-T760MP8	Adreno 430	Mali-T628 MP4	Adreno 430	Adreno 418
Battery	2550mAh	2840mAh	2680mAh	2900mAh	3000mAh

is similar, but has the more energy efficient A53 chip instead of the faster A57 chip. The Moto G remains an exception with only a quad core A53 chip. None of the phones described above, or any commercially available phone has an extra co-processor for secure cryptography. In terms of **RAM**, all phones have a similar configuration, with 3GB installed.

Most modern phones nowadays come with an extra processor for video, a GPU. The **GPUs** that are installed are all very similar, but there are differences between the phones. The Mali-T760P8 is at the top of the list, with a slight performance increase (and less energy consumption) than the Adreno 430. After this the Adreno 418 scores best (although quite lower than the 430), then the Mali-628MP4 and the Adreno 306. [41]

Battery capacity in smartphones are notorious for their low capacity relative to their power consumption because of the limited space available. All listed phones have a battery capacity of around 2500 to 3000 mAh.

2.3.2 *Smartphone limitations*

With embedded hardware in smartphones, they are limited in hardware specifications because of size and operating range limits (such as temperature) and power consumption. This severely limits the

possible computational power that is available to do cryptography or complicated protocol negotiations.

An in-depth research into the power consumption of a smartphone was done by Carroll and Heiser. They note that:

> "The GSM module consumes a great deal of both static and dynamic power. Merely maintaining a connection with the network consumes a significant fraction of total power. During a phone call, GSM consumes in excess of 800 mW average, which represents the single largest power drain in any of our benchmarks." ([42])

While this research is a little dated compared to the great jumps that were made in smartphone technology, this still seems to hold true in more recent research from 2013 [43], in which the CPU is also becoming a larger factor. It is now one of the main selling points for CPU cores [44] and power consumption is now an important factor in smartphone design.

In their research, Carroll and Heiser note that the power consumption is very low when the phone in in a suspended state. Normally, a phone can be in a suspended state for a very large part of the day, reducing its power consumption greatly. If the phone is prevented from going into the suspended state however, the battery can last as short as a few hours. This is a very important limitation for mobile systems.

2.3.3 *Mobile VoIP solutions*

Many different applications exist for VoIP on a mobile phone (including Android). Not all of them operate securely, though. Some VoIP applications only provide voice encoding, not encryption, which could be a choice for better performance, but does not give any security guarantees [45].

Nowadays, many voice applications have popped up on mobile app markets. Popular VoIP apps for Android (April 2016) are:

- Viber
- Google Hangouts
- Skype
- Tango
- imo.im
- WhatsApp
- LINE
- Facebook Messenger
- WeChat
- Vonage

Encryption is becoming increasingly more important for these services, especially in light of recent events. WhatsApp has just changed

its service to include end-to-end encryption on all media, including calls [2]. WhatsApp does this with an intricate key and encryption system. Keys are derived from a Root key, using different keys for users and a public key published to the WhatsApp servers. This is based on the double ratcheting algorithm to provide forward secrecy in a messaging system such as WhatsApp. On top of this, connections to servers are also done over an encrypted channel. Voice calls are encrypted using SRTP to encrypt the call. Signaling for calls happens via standard (encrypted) messages over which the SRTP master secret is sent.

2.4 MEASUREMENTS

To define what makes a good mobile VoIP solution, measurements will have to be done to compare different solutions. Important factors for such a solution are the call quality and the impact on the mobile phone. This should all be optimized while keeping the confidential nature of the call intact as well.

The call quality largely depends on the codec that can be used, which is directly tied to the amount of bandwidth that is available for the VoIP call. It also depends on the reliability of this connection, as no delays or jitter should occur.

The impact on the phone depends heavily on the power that is consumed. The power consumption is largely made up of: the WiFi/3G antenna, the CPU, the screen. This power consumption does not only occur during the call, but can also make a big impact because of keep-alive requirements of either the SIP or VPN channels.

Measuring the security of a VoIP solution cannot be done in a discrete, objective manner, and different types of solutions (e.g. SRTP versus a VPN channel) are not directly comparable. The security of solutions will therefore have to be reviewed in a discussion, according to standards and literature.

2.4.1 *Measuring VoIP performance*

To measure how efficient a VoIP solution is, we discuss several metrics that could be used. No methodology is defined however, as that is not the goal of this section.

List of performance metrics

OVERHEAD Byte overhead per data byte

POWER Power consumption for call

CPU TIME Delay imposed by extra (crypto) computations

CONNECTION QUALITY Delay and jitter measurements to determine call quality

The performance can also be influenced by difficult configuration issues which might not be solvable in every network. In some VPN solutions, for example, getting the packets to traverse over a NAT can be quite a problem, especially when they are encrypted. These caveats should also be taken into account when researching these solutions and will be discussed when they occur.

2.4.2 *Discussing Security*

While the performance of a VoIP or VPN solution can be easily measured with discrete measurements, it is harder to define how secure such a solution is. Some attack vectors can be discussed objectively, such as the level of encryption that is used, or the type of hash used to sign a message. As not all attack vectors can be known, and sometimes depend on underlying protocols or systems, it is also a very subjective topic. To mitigate known vulnerabilities, it is best to rely on existing products and software that have been either tested or researched extensively or even especially designed to be secure. This is one of the reasons why basing the security of VoIP on a VPN is a good idea, as it has been designed for robust security and has been extensively tested.

In literature, such security considerations are always very specific towards a product. General frameworks exist to test protocols for basic security features, but these are not relevant to test current VPN technologies as they already suffice these guarantees. To discuss a solution and its security level however, many different sources exist on requirements for VoIP security. In section 2.2.1.2 we have already seen such a requirement for OpenVPN to meet a certain security guarantee. Some of such sources are discussed in this section.

One important indicator for a secure solution is the level of encryption that should be used. The algorithm should be robust and the key that is used should be of sufficient length. If this requirement is met, several secondary performance indicators can be addressed. The solution should minimize resource consumption (CPU and memory usage), for example. The solution should also be fast, especially if it is used to secure time sensitive traffic like VoIP.

Another important factor in how secure a product is, is how subjective it is to misuse as a covert channel. In such a misuse of a protocol, information can be leaked out of a network undetected. This is also discussed further below.

2.4.2.1 *Standards*

Security in (new) technology is especially hard to judge because attack vectors might exist in unknown ways. Some standards do exist for general security but are hard to apply to specific technologies. In the case of VoIP, documents exist that discuss how VoIP should be

set up to be secure. These standards are mostly set up by governing organizations and governments. A few notable documents are:

NIST The computer security division of the NIST has published a "Security considerations for Voice over IP Systems"-document with recommendations on its configuration and discusses security for VoIP [46].

HKSAR The Government of the Hong Kong Special Administrative Region also has a document published on VoIP security considerations [47].

HOMELAND SECURITY As attachement to the Sensitive Systems Handbook, VoIP is also discussed including checklists for security [48].

2.4.2.2 *Covert Channels*

Even if the communication is entirely secure, information can still be leaked, either intentional by a malicious party, or because of the way a protocols works. These information leakages are called covert channels [18] and are an important factor in how secure a protocol is, and if it can be used in a certain setting. For example, if a protocol allows for high cover channel throughput, it might not be suited to be used in a corporate setting where information leakage is a critical security concern.

In some protocols, even if the data is protected, some information can be gained by analyzing the packets that are sent. This is a type of attack where information can be gained via a side-channel such as meta-data or timing. For example by monitoring the network for the occurrence SIP communications, a malicious party could determine whether someone is making a call, and possibly even to whom.

Another way covert channels can be (ab)used is by misusing certain fields or properties of the protocol to hide information. This type of misuse falls under steganography in protocols. A malicious attacker can for example use fields or packet delays to sneak out information from the internal network. This can especially be a problem for protocols with a high throughput, such as VoIP protocols [49]. This is not only true for the data transfer phase of the VoIP call, but also in the signaling phase, even if it is only short. In [50], research was done into SIP steganography, and showed that 2000 bits can be sent in one direction during the call initiation phase. In the data transfer phase, even higher covert channel bitrates can be constructed, such as in [51] sending over 1.3Mbits in a typical VoIP call and in [49]. Depending on what kind of encoding is used, results differ.

One way to counter such leaking of information is by the use of steganalysis [52]. This form of protection can be difficult for a real-time protocol such as VoIP, because any (irregular) delay can have a

large impact on the voice quality. However, some techniques exist to detect steganography in VoIP such as researched in [53] and [54].

3 VOIP CHARACTERISTICS

To analyze VoIP, and to compare different solutions later on, the standard characteristics of VoIP have to be determined. This part of the research answers the first research question: **What are VoIP communication characteristics?**

3.1 METHODS

3.1.1 *Setup*

First, a VoIP setup based on SIP and RTP was configured. This was done using an Asterisk PBX on a virtual Ubuntu server. Two clients were connected to this PBX using Linphone on Ubuntu. See figure 1 for a diagram of the setup.

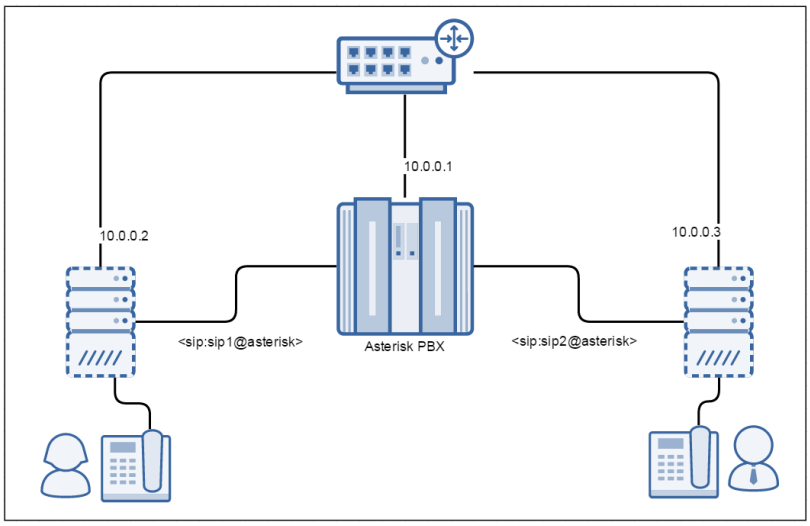


Figure 1: Virtual VoIP setup

A router with NAT was configured in the virtual environment. Within this virtual environment, both clients and the server receive an IP address. When the clients are started and Linphone is executed, they register at the PBX through a SIP `REGISTER` message. An example of this is seen below in figure 2. In this case, the client is `192.168.56.101` connecting to the server at `192.168.56.103` using TCP for SIP signaling.

This message is then rejected by the PBX, as authentication is required. A SIP `401 Unauthorized` is sent back, including authentication settings with a nonce. This is used by the client to send an au-

```
Session Initiation Protocol (SIP as raw text)
    REGISTER sip:192.168.56.103 SIP/2.0\r\n
    Via: SIP/2.0/TCP 10.0.3.15:5060;rport;branch=z9hG4bK1291991810\r\n
    From: <sip:1236@192.168.56.103>;tag=497768841\r\n
    To: <sip:1236@192.168.56.103>\r\n
    Call-ID: 2340511\r\n
    CSeq: 1 REGISTER\r\n
    Contact: <sip:1236@192.168.56.101:60168;transport=tcp;line=a320cafc7acf78e>\r\n
    Max-Forwards: 70\r\n
    User-Agent: Linphone/3.6.1 (eXosip2/4.0.0)\r\n
    Expires: 3600\r\n
    Content-Length: 0\r\n
    \r\n
```

Figure 2: SIP Register message

thenticated SIP message including the authorization. This is replied with an `OK` from the PBX.

3.1.2 *Codec Performance*

To test the VoIP connection, an audio sample was played over the VoIP connection. First, a call is set up between the two clients. Asterisk can route the call in two ways, firstly as in figure 1 with the call routed through Asterisk and secondly as in figure 3, with a direct RTP connection. The SIP messages are always routed through the Asterisk server.

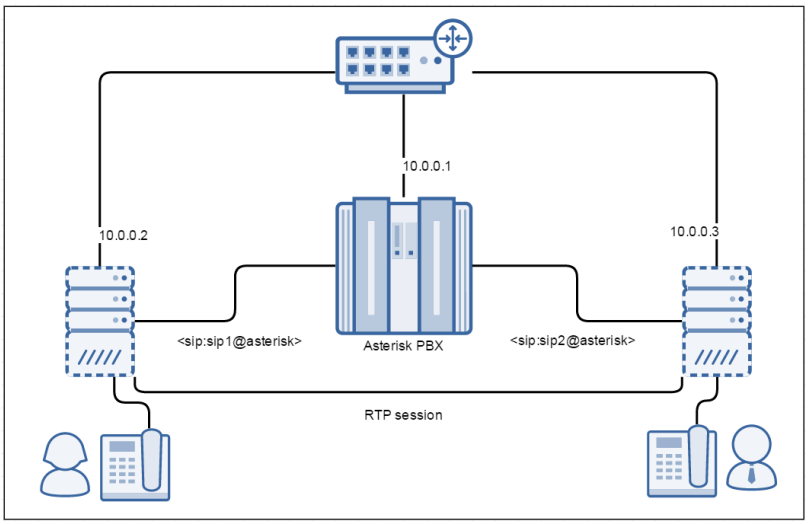


Figure 3: VoIP with direct RTP

For this test, the call was be routed through the asterisk PBX. This setup can best be used in a scenario where one (or both) of the callers has to use a secure connection. This connection can then be specially configured and secured by the PBX, without requiring anything from the machine at the other end of the call.

Because of availability in Asterisk and on the VoIP client Linphone, the following codecs were tested:

SPEEX A common very flexible codec designed for VoIP, tested in 8kHz, 16kHz and 32khz sampling rates.

μ/A-LAW Two versions of the G.711 codec, with slight algorithm changes.

G.722 Standard wideband codec as improvement on the G.711 standard for better quality.

G.726 Bandwidth saving codec in comparison to G.711 codecs.

GSM A very (old) standard and widely adopted codec.

Some of these codecs support VBR (Variable Bit Rate), which can reduce the bitrate needed depending on the voice data that is sent. While this reduces bandwidth requirements, it can be unsafe, as shown in a paper by Wright et al.[55]. By analyzing the VoIP packet lengths, information about spoken sentences can be derived.

The procedure to test each codec consisted of numerous steps. First, the VMs were loaded and configured. This consisted of the Server VM, on which Asterisk was started. Then, two client VMs were started with Linphone. Both Linphone applications were then configured for the Asterisk server and registered. On the server VM, `/etc/asterisk/sip.conf` was changed to only allow the specifc codec that would be tested. The following lines would be added: (note that ';' is a comment in this configuration file)

```
disallow=all
allow= <codec> ; (e.g. g722, speex32, etc.)
```

Then, tcpdump would be started with the following command:

```
$ tcpdump -s 0 -i eth0 -w <filename>.pcap
```

Following this, the call was set up between the two VoIP clients. A one minute audio sample was then played from one client to the other. The call was then ended and tcpdump stopped. The resulting pcap was then retrieved from the client VM to the host machine. This was repeated for each codec.

3.1.3 *TCP Test*

Asterisk can be configured to run over TCP. For this test, this configuration option was set. A few of the different codecs were enabled and the voice sample was played over the channel.

For Asterisk, this means the SIP traffic goes over TCP, but the RTP channel is still transported over UDP. The difference in SIP traffic size can then be measured.

For this test, the same setup as depicted in figure 1 was used. The SIP traffic, which is relevant to this test, was routed through the Asterisk PBX.

3.1.4 *Frame size*

An important factor in how VoIP traffic is shaped is how the voice data is packetized. The RTP packets can be as small as 10ms of voice data, or as large as 300ms. The range of packetization that is possible depends on the codec that is used. The range of values that are possible are shown in table 2.

Table 2: Asterisk packetization options [56]

Name	Minimum (ms)	Maximum (ms)	Default (ms)	Increment (ms)
g723	30	300	30	30
gsm	20	300	20	20
ulaw	10	150	20	10
alaw	10	150	20	10
g726	10	300	20	10
ADPCM	10	300	20	10
SLIN	10	70	20	10
lpc10	20	20	20	20
g729	10	230	20	10
speex	10	60	20	10
ilbc	30	30	30	30
g726_aal2	10	300	20	10

The packetization options for G.722 were unknown. This was experimentally defined and discussed in results section 3.2.3. G.722 only supports a frame size of 10ms or 20ms in Asterisk.

One codec was analyzed for several different packetization values. To facilitate a broad range of possible values, a codec with a large maximum frame size was selected out of the codecs that are available to us. This leaves G.726 with a range between 10ms and 300ms, G.711 codecs with a range between 10ms and 150ms, and GSM with a range between 20ms and 300ms. The choice was made to perform the test on the G.711 codec (μ-law or A-law) as it is commonly used as a high-quality codec in VoIP systems. This does not allow for framing sizes from 150ms to 300ms, but such values are almost never an option anyway since they would incur too much delay and possible loss of large chunks of voice data. The A-law codec was chosen as it has a broader usage.

3.2 RESULTS

The results from both tests described in the methods section are detailed below.

3.2.1 *Codec Performance*

Results from testing the different codecs in the virtual environment are summarized in table 3. The raw results are shown in appendix A. These results relate to the RTP connection that was made for the call. Per codec, the average jitter, packet count, RTP stream duration and PPS (Packets per second) value is shown. The maximum jitter that occurred is also shown. No RTP packets were lost in the simulation.

Table 3: Average RTP statistics and max jitter per codec

Codec	Jitter	Max. Jitter	Packets	Duration	PPS
Speex32	17,91	130,18	3317	66,30	50,03
Speex16	17,70	61,62	3264	65,32	49,97
Speex8	15,53	35,10	3456	69,30	49,87
µ-law	17,41	54,85	3368	67,44	49,94
A-law	15,68	27,51	3608	72,13	49,97
G.722	15,85	41,24	3303	66,06	50,00
G.726	16,53	70,34	3369	67,53	49,89
GSM	16,31	26,37	3271	65,53	49,92

Note that the average PPS is almost always exactly around 50. All listed protocols have a default framing size of 20ms. This means that every 20ms, a frame is sent in a packet, resulting in 50 packets for every second of voice data.

Results for the entire connection (SIP & RTP) are detailed in table 4. These results include the average amount of bytes that were transferred, and the average rate in KiloBytes per second that the connection had. Figure 4 shows the total amount of transferred bytes per codec in a bar chart, clearly showing the large differences between codecs.

Table 4: Average total payload and throughput per codec

Codec	KBytes	Rate KB/s
Speex32	1657,38	22,33
Speex16	1638,32	23,67
Speex8	1019,04	14,00
µ-law	2834,39	41,00
A-law	3035,33	41,33
G.722	2779,68	39,67
G.726	1129,56	25,00
GSM	1780,71	16,33

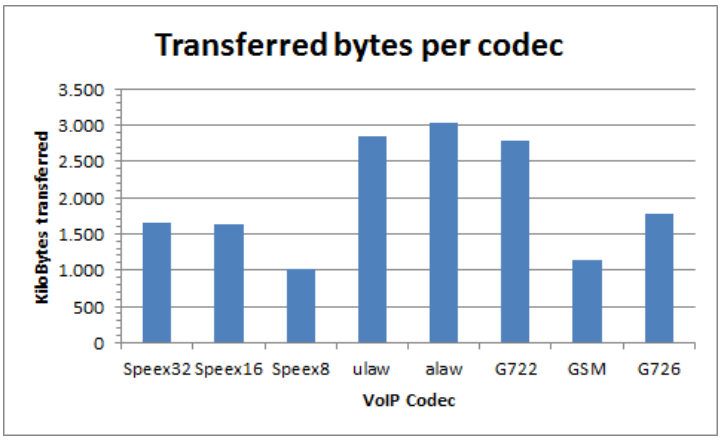


Figure 4: Total bytes transferred per codec

3.2.2 *TCP Test*

In a standard call, with SIP configured to run over UDP, only 10 SIP packets are interchanged between a caller and the PBX. An example of this traffic, captured in the setup described in section 3.1.1, is shown in figure 5.

Source	Destination	Protocol	Length	Info
192.168.56.104	192.168.56.103	SIP/SDP	998	Request: INVITE sip:1236@192.168.56.103 |
192.168.56.103	192.168.56.104	SIP	522	Status: 401 Unauthorized |
192.168.56.104	192.168.56.103	SIP	291	Request: ACK sip:1236@192.168.56.103 |
192.168.56.104	192.168.56.103	SIP/SDP	1164	Request: INVITE sip:1236@192.168.56.103 |
192.168.56.103	192.168.56.104	SIP	466	Status: 100 Trying |
192.168.56.103	192.168.56.104	SIP	482	Status: 180 Ringing |
192.168.56.103	192.168.56.104	SIP/SDP	878	Status: 200 OK |
192.168.56.104	192.168.56.103	SIP	395	Request: ACK sip:1236@192.168.56.103:5060 |
192.168.56.104	192.168.56.103	SIP	572	Request: BYE sip:1236@192.168.56.103:5060 |
192.168.56.103	192.168.56.104	SIP	434	Status: 200 OK |

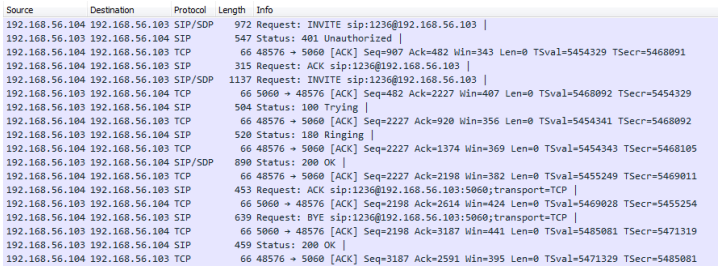
Figure 5: SIP traffic over UDP

A notable feature in this traffic is that the `INVITE` packet is sent twice. The first time, it does not contain any authorization data and the response is a `401 Unauthorized`. The response also contains the authorization information with a nonce, which can be used to send a valid authorized `INVITE` packet. In this example, only one `Trying` and `Ringing` packet is returned, but these could be multiple packets.

When the same process is done, but with SIP configured to run over TCP, 18 packets are exchanged. An example of this traffic, comparable to the UDP example, is shown in figure 6.

Interesting to note here, is that not all packets are replied to with an explicit `ACK` packet. Specifically, the first `INVITE` message and the SIP `ACK` message do not need a specific TCP `ACK` message.

We define three phases in the SIP traffic. First, the caller authorizes himself and sends an `INVITE` packet. Next, the PBX sets up the call and the caller sends an `ACK` message to acknowledge the call. Finally, the caller ends the call with a `BYE` message which is confirmed.



Source	Destination	Protocol	Length	Info
192.168.56.104	192.168.56.103	SIP/SDP	972	Request: INVITE sip:1236@192.168.56.103 |
192.168.56.103	192.168.56.104	SIP	547	Status: 401 Unauthorized |
192.168.56.104	192.168.56.103	TCP	66	48576 → 5060 [ACK] Seq=907 Ack=482 Win=343 Len=0 TSval=5454329 TSecr=5468091
192.168.56.104	192.168.56.103	SIP	315	Request: ACK sip:1236@192.168.56.103 |
192.168.56.104	192.168.56.103	SIP/SDP	1137	Request: INVITE sip:1236@192.168.56.103 |
192.168.56.103	192.168.56.104	TCP	66	5060 → 48576 [ACK] Seq=482 Ack=2227 Win=407 Len=0 TSval=5468092 TSecr=5454329
192.168.56.103	192.168.56.104	SIP	504	Status: 100 Trying |
192.168.56.104	192.168.56.103	TCP	66	48576 → 5060 [ACK] Seq=2227 Ack=920 Win=356 Len=0 TSval=5454341 TSecr=5468092
192.168.56.103	192.168.56.104	SIP	520	Status: 180 Ringing |
192.168.56.104	192.168.56.103	TCP	66	48576 → 5060 [ACK] Seq=2227 Ack=1374 Win=369 Len=0 TSval=5454343 TSecr=5468105
192.168.56.103	192.168.56.104	SIP/SDP	890	Status: 200 OK |
192.168.56.104	192.168.56.103	TCP	66	48576 → 5060 [ACK] Seq=2227 Ack=2198 Win=382 Len=0 TSval=5455249 TSecr=5469011
192.168.56.104	192.168.56.103	SIP	453	Request: ACK sip:1236@192.168.56.103:5060;transport=TCP |
192.168.56.103	192.168.56.104	TCP	66	5060 → 48576 [ACK] Seq=2198 Ack=2614 Win=424 Len=0 TSval=5469028 TSecr=5455254
192.168.56.104	192.168.56.103	SIP	639	Request: BYE sip:1236@192.168.56.103:5060;transport=TCP |
192.168.56.103	192.168.56.104	TCP	66	5060 → 48576 [ACK] Seq=2198 Ack=3187 Win=441 Len=0 TSval=5485081 TSecr=5471319
192.168.56.103	192.168.56.104	SIP	459	Status: 200 OK |
192.168.56.104	192.168.56.103	TCP	66	48576 → 5060 [ACK] Seq=3187 Ack=2591 Win=395 Len=0 TSval=5471329 TSecr=5485081

Figure 6: SIP traffic over TCP

In table 5 the different phases are analyzed, comparing SIP over UDP and TCP. Interesting to note is that, while the packet overhead is large, the added bytes by TCP are not that large. As the configuration was not entirely identical between the two tests, a correction was applied to the UDP data. 62 bytes were reduced from both `INVITE` packets and 22 bytes were reduced from the `200 OK` packet to correct for extra enabled codecs.

Table 5: SIP comparison between UDP and TCP

	Packets			**Bytes**		
Phase	**UDP**	**TCP**	**Factor**	**UDP**	**TCP**	**Factor**
Invite	4	6	1.50	2851	3103	1.088
Setup	4	8	2.00	2199	2631	1.196
End	2	4	2.00	1006	1230	1.223
Total	10	18	1.80	6056	6964	1.150

3.2.3 *Framing*

3.2.3.1 *Framing G.711*

G.711 has a range of framing options between 10ms and 150ms. The framing options increase with 10ms. In standard G.711, each 10ms of data contains 80 bytes of payload. This means that the payload will increase linearly as the framing size increases.

Each packet has an overhead, it consists of the following parts:

- **Ethernet** The ethernet encapsulation creates an overhead of 14 bytes
- **IP** In the testcase, IPv4 was used. This creates an overhead of 20 bytes. In the case of IPv6, this would be 40 bytes.
- **UDP** The UDP header has an overhead of 8 bytes.

As the packet overhead stays consistent at 42 bytes, the RTP conversation benefits if the amount of packets is reduced. The amount of

packets depends solely on the size of the framing settings. Depending on which codec is used, the amount of bytes that is contained in the packet is either larger or smaller than G.711. The results of the amount of packets is displayed in figure 7 and for the data in figure 8. The raw results are displayed in appendix B. The data is corrected to correspond to a 60 second conversation.

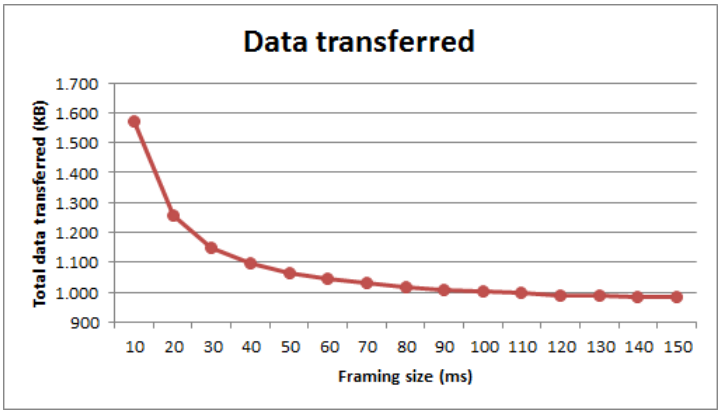


Figure 7: Data transferred results for framing test

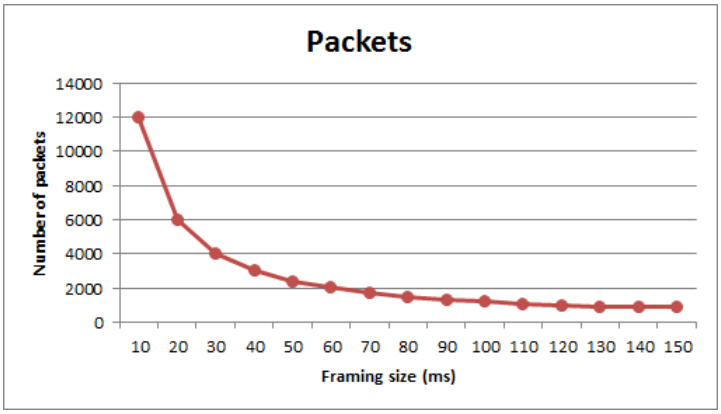


Figure 8: Packet count results for framing test

While large framing sizes cause overhead to be reduced, they also increase delay. The maximum one-way delay should be kept as small as possible to guarantee good voice quality. This makes high framing sizes not feasible in a real setting. It is clear from the data that large overhead issues can be seen in lower framing sizes. After around 40ms, the difference of increasing the framing size is quite small, making this a suitable choice. If an even larger gain is required, 80ms could also be chosen as the framing size. This might incur some delay however.

3.2.3.2 *G.722 options*

As the packetization options for G.722 were unknown, this was experimentally confirmed to be 20ms by default, and with a minimum

of 10ms and maximum of 100ms. This was first attempted by over- and underconfiguring the packetization values for G.722 (with 5ms and 400ms) and measuring the resulting packets per second output. However, this resulted in Asterisk to select the default 20ms value, contrary to the specification. Options from 10 to 100ms were possible, with 10ms steps. The direct results of over-configuring are presented in figure 9. The pps values are $\frac{6603}{66,00} = 100,05$ and $\frac{3168}{63,30} = 50,05$ or framing sizes of 10 and 20ms respectively.

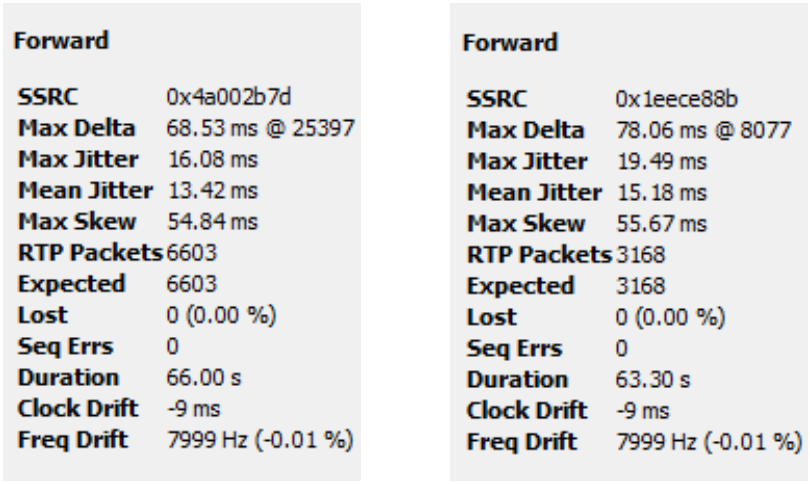


Figure 9: G722 framing size results

3.2.3.3 *Framing with compression*

When changing framing options with a codec that uses compression, compression might become more effective on a larger sample. In a preliminary analysis, the following results were found for G.722, a wideband codec that does compression effectively:

Table 6: Framing size with G.722

Frame size	Payload	Factor	Normalized factor
10	92	1	1
20	172	1,87	0,935
60	492	5,35	0,892
80	652	7,09	0,886
100	812	8,83	0,883

The payload seems to increase in a non-linear fashion. However, on closer inspection, there seems to be a 12 byte fixed size embedded in the payload, with a further 80 bytes of voice data per 10ms of framing. It does not seem to compress more efficiently on larger samples.

3.3 FINAL CONSIDERATIONS

Results from the codec tests do not show large differences in performance in regard to jitter and number of packets. There are however (large) differences in the amount of data that is transferred. While some codecs even use about half as much data as other codecs, such as GSM compared to A-law, their quality is also generally perceived as less.

Using SIP over TCP has a larger overhead than over UDP, but it is not twice as large as the UDP variant of SIP. This is because some TCP ACK messages are not sent if the SIP message is an ACK. There are also not many SIP packets relative to the amount of RTP packets in a call. This makes TCP a very viable option for SIP if it has benefits, such as NAT port forwarding timeouts.

G.711 was chosen to do packetization tests with, as it is commonly used and has a good MOS score. The specific A-law version of G.711 was chosen because of its broader usage, as μ-law is used only in North America.

Packetization has a very large impact on the amount of overhead produced in RTP packets. While larger framing sizes cause less overhead, one-way delay should still be kept as small as possible. A framing size of 40ms seems appropriate for these requirements.

4 MOBILE VOIP SOLUTIONS

To identify current popular ways of securing VoIP, a few important mobile VoIP applications for Android were looked at. An analysis of these applications answers the second research question: **What are the differences between mobile VoIP solutions?**

Three large voice communication applications are analyzed in this chapter. These are: Hangouts, Skype and Whatsapp. They correspond to some of the largest Internet companies, respectively Google, Microsoft and Facebook. These companies have different stances on privacy and security, and their implementations in these applications might be very different.

4.1 METHODS

The current trend is determined in three different ways. Firstly, the three different companies will be quickly reviewed to discover their stance towards privacy and security. Secondly, technical documentation will be reviewed for the specific applications. Finally, packet captures will be analyzed for the applications and analyzed.

For the second part, reviewing technical documentations, a literature study is done. Not only literature is regarded, but also white papers and other documentation available from these companies.

The packet capture will be made using an Android phone. The phone will be connected via WiFi to a wireless access point. Packets will be captured at this access point using Wireshark. To accomplish this, a laptop is used as the access point, with the wired connection shared on the wireless interface. The network setup is depicted in figure 10. The packet capture is then filtered for other traffic beside the VoIP traffic.

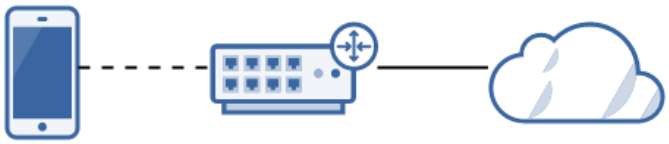


Figure 10: Packet capture for apps

An analysis technique similar to the one used in a study by Azfar et al.[45] will be used. Histogram analyses will be done for these traces

to determine traffic security characteristics. The same voice sample as used in section 3.1.1 will be used for these traffic captures.

In a histogram analyses of the data, the distribution of byte values can be seen. If the data is encrypted, such a distribution is uniform. If the data is encoded, chunking can be seen (groups of bytes with a higher occurrence). In plaintext data, such chunking can also be seen, and will be predictable depending on the data that is sent.

4.2 RESULTS

4.2.1 *Information gathering*

Specifications for the three different apps are looked up and inspected. From these documents, some conclusions can be drawn on how secure these applications are and thus how privacy is protected.

4.2.1.1 *Hangouts*

Google Hangouts encrypts all signals and audio/video. All signals are encrypted over an HTTPS connection with authentication. Messages are sent with these signals, securing them as well. 128-bit AES is used with ECDHE-ECDSA key exchange, also guaranteeing perfect forward secrecy on the transmission. [57] Signals are however not encrypted end-to-end and Google has access to the unencrypted data on their receiving servers. Audio and Video are encrypted using SRTP with AES ciphers and an HMAC using SHA1 for authentication. The Hangouts page added towards the end of 2015 that "To improve audio and video quality, Hangouts calls use a direct peer-to-peer connection when possible, instead of routing through a server."[1] This implies that this was not true before this time, and not guaranteed to be end-to-end.

4.2.1.2 *Skype*

One of the oldest and most known Voice over IP applications, Skype has been actively used since 2003. Skype once started as a peer-to-peer voice application, but after it was taken over by Microsoft in 2011, it replaced all peer-operated supernodes by Microsoft servers.

Skype states on its website that all communication is encrypted, but this only seems to imply the connection to the server, as only TLS is used for messages. When peers connect directly, AES is used. However, Skype also states that "in the future it will only be sent via our cloud to provide the optimal user experience." [58]

Much criticism has been expressed against Skype as it was revealed that many if not all of Skype communications were shared with gov-

1 Old version available at the Web Archive http://web.archive.org/web/20150914224359/https://support.google.com/hangouts/answer/6046115?hl=en

ernments by Microsoft. This shows that by routing many of the traffic through their own supernode-servers, Microsoft gains access to much of the unencrypted data. When once Skype was peer-to-peer encrypted, much of this is now compromised. [59]

4.2.1.3 *Whatsapp*

Whatsapp has just recently updated their entire security backend with the integration of the Open Whisper Systems algorithms. Before this, only a TLS channel was used to secure the communication channel.

Their new encryption scheme was even published in a white paper, making their security schemes publicly available for review. This is rare for large popular apps, and great for public knowledge.

4.2.2 *Trace analysis*

For each application, three sample measurements were taken. The measurements entailed a call of approximately 30 seconds, with calling phase and hanging up included. Each application is discussed below, detailing basic results from the samples, histogram analysis and other remarks.

4.2.2.1 *Hangouts*

Hangouts connects directly to a Google server, though which the connection is routed. STUN (Session Traversal Utilities for NAT) is used for NAT traversal. The basic trace results are shown in table 7. Signaling seems to be done via the QUIC (Quick UDP Internet Connections) protocol. Voice data is sent in UDP payloads. No direct connection was set up between clients, although this seems to be quite feasible, as other applications did manage to do this.

Table 7: Hangouts basic trace results

	Hangouts			
	1	2	3	Average
Packets	4901	3804	4234	4313
Bytes	733555	568342	624325	642074
Duration	51,4	38,7	47,7	45,93
PPS	95,35	98,29	88,76	94,14
kBPS	7,51	5,65	6,87	6,68

The histogram data for Hangouts, as shown in figure 11, does not show any strange particularities. The data seems to be spread out evenly with minimal clustering. This supports the information given on the Hangouts website.

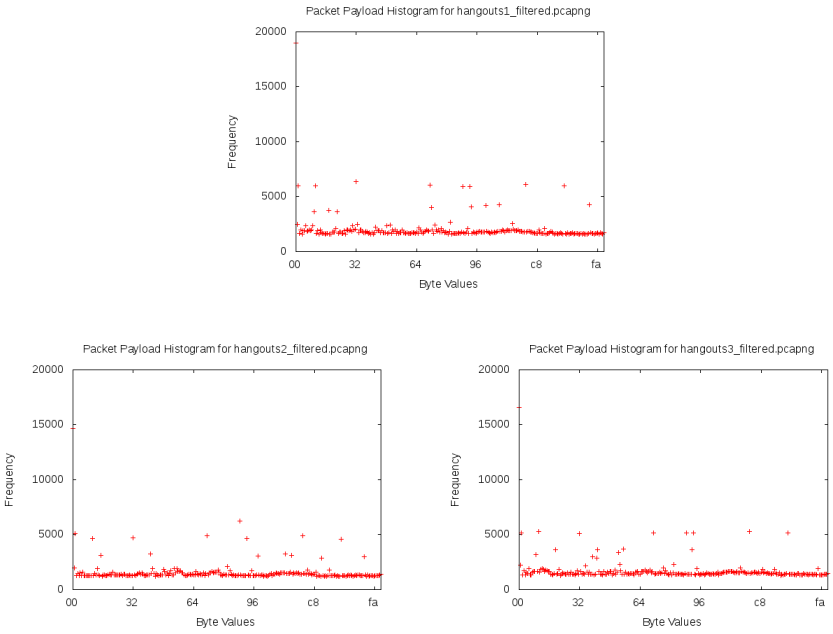


Figure 11: Hangouts Histogram results

4.2.2.2 *Skype*

Skype directly connects the two clients to form the voice connection initially. STUN seems to be used for NAT traversal in this phase. Very quickly however, a connection via a Skype server is set up. This connection is setup using TURN (Traversal Using Relays around NAT). After a few seconds, all data is sent over this connection. It is strange that a TURN relay is set up for the connection, where only STUN was sufficient for hangouts to traverse the NAT. This occurred for all three samples.

Compared to the other apps, Skype used the least amount of bytes per second. The results can be seen in table 8.

Table 8: Skype basic trace results

	Skype			
	1	2	3	Average
Packets	3546	4939	3525	4003
Bytes	436843	617072	438392	497436
Duration	35,8	48,6	34,5	39,63
PPS	99,05	101,63	102,17	100,95
kBPS	4,31	5,93	4,19	4,81

The histogram data for Skype is a little less consistent than the one for Hangouts. This is most likely caused by the TURN encapsulation of the data, causing some small bumps that are not uniform in the

data. The rest of the data does seem to be spread uniformly, supporting the statements about Skype's encryption.

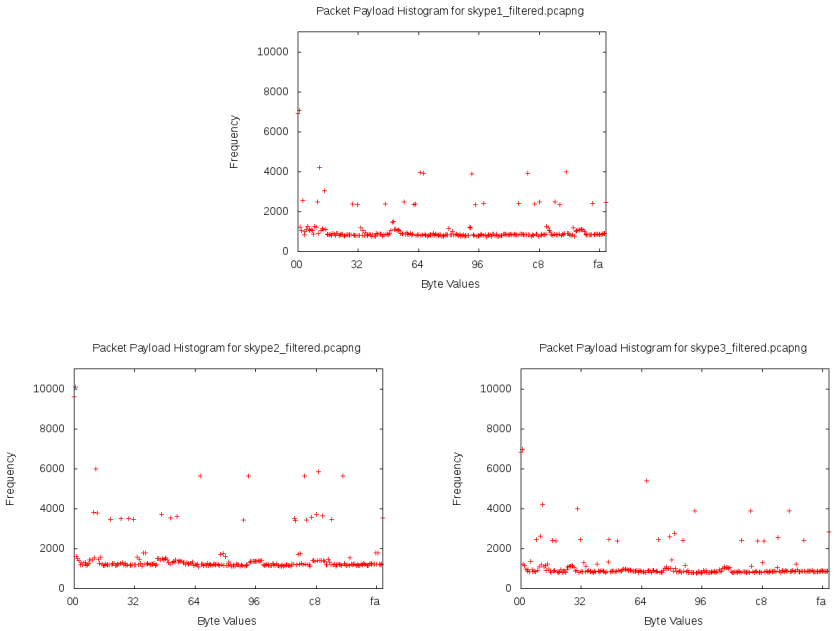


Figure 12: Skype Histogram results

4.2.2.3 *Whatsapp*

Whatsapp does not connect two clients to each other, but a connection to a Facebook server is set up instead. All traffic is routed through this connection. Much like the connection with Hangouts, STUN is used for NAT traversal. Contrary to Hangouts, the STUN messages are used very sparingly, and only when it is needed, which heavily reduced the amount of packets for this connection. Results are listed in table 9.

While Whatsapp used a very limited number of packets compared to the other applications, it did carry the most bytes per second. This can potentially be more efficient as less overhead is used. It most likely means that the compression for Whatsapp is worse than the other apps.

Out of all three applications, the histogram for Whatsapp is most consistently spread. This supports the fact that Whatsapp is properly encrypted.

4.3 FINAL CONSIDERATIONS

Google Hangouts, Skype and Whatsapp all have different encryption schemes in use. They all employ different techniques of communication though, which results in different characteristics in their trace

Table 9: Whatsapp basic trace results

	Whatsapp			
	1	2	3	Average
Packets	1088	968	925	994
Bytes	210683	194669	181370	195574
Duration	44,1	37,7	40,5	40,77
PPS	24,67	25,68	22,84	24,40
kBPS	8,34	7,40	7,75	7,83

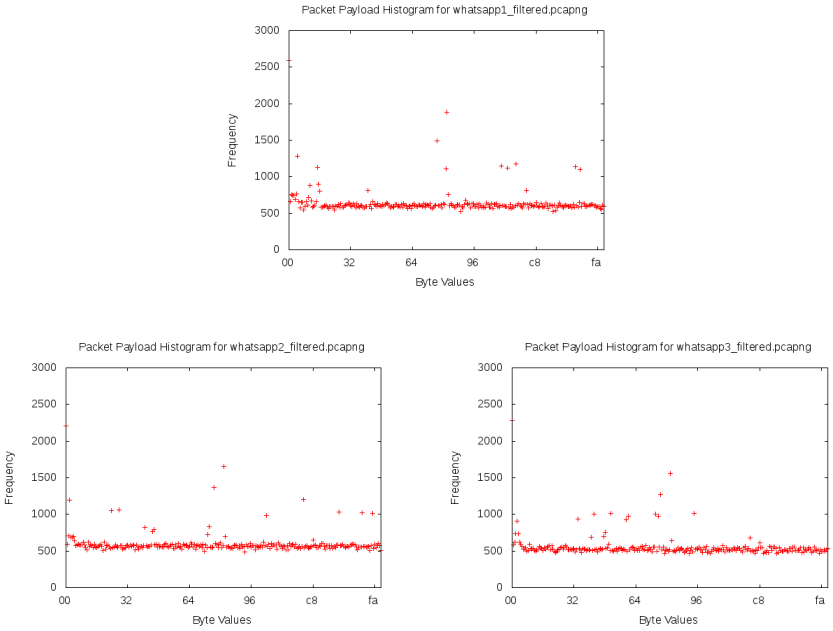


Figure 13: Whatsapp Histogram results

samples. Skype seems to employ the best compression, and uses TURN for NAT Traversal. Whatsapp and Hangouts turn out to be very similar, using slightly more bytes per second than Skype.

A trend can be noticed for apps to go towards end-to-end encryption. This can be seen in apps such as Whatsapp changing their protocol to employ OpenWhisper encryption algorithms, but it is also not the best for every use-case. This trend currently exist mainly because of the heightened demand for end-to-end encryption. While this was not an issue or an important feature of applications for consumers before, now applications are criticized for their encryption choices. However, on the other side of this discussion exists a darker argument: When everything is entirely secure (and maybe even anonymous), no legal surveillance can be done by governments or companies monitoring internal traffic. This makes these techniques very

suitable for malicious usage such as exfiltrating information from a network, or having illegal communication.

5 VPN TUNNELS

In this chapter, VPN tunnels are discussed. Two different types of VPN are tested, OpenVPN, which works on the application layer, and IPSec VPN, which works on the IP layer. As VoIP traffic can be very vulnerable to delays and unreliable bandwidth, it is important to know how the VPNs affect the data that is transferred. The third research question **How does a VPN tunnel affect different types of traffic?** will be discussed in this chapter.

VPNs are generally used to connect different networks or machines together. A typical example is securely connecting to a corporate network from home. All traffic sent to the connected network is then secured through the tunnel.

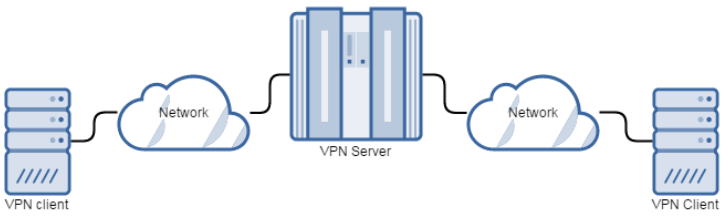


Figure 14: Network setup for VPN Tunnels

5.1 METHODS

Both VPNs were setup and configured in the virtual environment. As they work in very different layers of the OSI stack, they have a very different technique to encapsulate the data. These ways, and their configuration in the setup, are discussed.

Both VPNs have a network configuration as depicted in figure 14. Two client VMs are connected to the main server VM. A network can exist between the server and the client, as the data is fully encapsulated and encrypted.

5.1.1 *OpenVPN*

OpenVPN creates a second interface device through which a connection can be made by an application. This can either be a TAP or TUN interface, corresponding to OSI level 2 and OSI level 3 networking respectively. All packets sent through this interface are encrypted with

the configured encryption scheme. As we make use of the OpenVPN-NL package, this encryption is AES-256-CBC, signed with SHA256. Additionally, the packet contents can be compressed using LZO (Lempel–Ziv–Oberhumer) compression.[60]

Serverside, OpenVPN is configured with a TUN interface. It uses the private `10.8.0.0/24` subnet for VPN routing. It is configured to let clients see each other on the network. A root CA was setup, generating certificates for the clients and the server. DH-2048 parameters were also generated for the server. The configuration can be seen in appendix D.1.1.

Two client VMs were configured with OpenVPN, using generated client settings and certificates from the server. The clients connect with the same certificate, but are given separate IP addresses. As the client-to-client setting is enabled, traffic can flow from one client, through the VPN server, to the other client. The client configuration can be seen in appendix D.1.2.

5.1.2 *IPSec*

IPSec works as part of the Internet Protocol. To secure the connection, ESP (Encapsulating Security Payload) is used. This is done to use IPSec as OpenVPN was used: to encrypt the payload. The encryption is done with AES-256 as was used in OpenVPN. This makes both protocols comparable in terms of speed and overhead efficiency.

On the server, IPSec is configured with a receiving PPP (Point-to-Point) tunnel interface using xl2tpd. The Openswan IPSec distribution is used. The tunnel uses the IP address range `172.16.1.30-100`. As authentication scheme, MS-CHAP v2 was configured. In contrast to the OpenVPN setup, no certificates were generated for IPSec, but a PSK (Pre-Shared Key) was configured. For CHAP authentication, users 'alice' and 'bob' were created with a simple password. The configuration can be seen in appendix D.2.1.

Two client VMs were configured to connect with the IPSec server. IPSec using xl2tpd was set up for the users 'alice' and 'bob' on these clients. Their configuration is listed in appendix D.2.2.

5.1.3 *Tests*

For both VPNs, several tests are performed. Because OpenVPN and IPSec function very differently, they are not subjected to the same tests entirely.

5.1.3.1 *UDP packet range test*

A range of different UDP packets are tested to see the effect of the VPN solution. This shows the amount of overhead the VPN produces

on each data packet. UDP packets with sizes of 100, 200, 400, 800 and 1600 bytes are tested.

To measure the VPN difference, packets are captured on the ethernet interface (etho), where the packets are encapsulated, and on the VPN interface (tuno for OpenVPN, pppo for IPSec), on which the original packets are captured. By comparing these results, the encapsulation characteristics can be determined.

5.1.3.2 *Compression*

OpenVPN and IPSec can additionally compress packets if enabled in the configuration. To test this compression, it was enabled in the server configuration, and the range of UDP packets was sent again. Both traces are compared against each other again to determine the scope of the compression. Packets are sent with a random and semi-random payload to test the compression.

5.1.3.3 *IPSec Encapsulation*

To traverse a NAT, IPSec can be encapsulated in UDP packets. This option was forced in the IPSec configuration to test the overhead created by the encapsulation. It is enabled with the `forceencaps=yes` option. Packets were again captured on both interfaces, and compared to define the scope.

5.2 RESULTS

5.2.1 *UDP packet range test*

The UDP range test was performed for both OpenVPN and IPSec.

5.2.1.1 *OpenVPN*

The results for OpenVPN can be seen in table 10. Captures on the `ppp0` interface do not contain a link-layer overhead, which the `eth0` captures do. This overhead of 14 bytes is corrected in the tables below. A clear difference can be seen in the encapsulated and unencapsulated packets. The results show that the OpenVPN encapsulation adds between 89 and 97 bytes in overhead for unfragmented packets. An example can be seen in figure 15 without the encapsulation, and the same packets after the TUN interface encapsulation in figure 16.

Results for all packet sizes are shown in table 10. The overhead differs a bit for the different sizes. If the packet is fragmented, OpenVPN creates even more overhead, as can be seen at the 1600 and 3200 byte values.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.8.0.10	10.8.0.8	UDP	828	55507 → 5001 Len=800
2	0.006452	10.8.0.10	10.8.0.8	UDP	828	55507 → 5001 Len=800
3	0.012853	10.8.0.10	10.8.0.8	UDP	828	55507 → 5001 Len=800
4	0.018738	10.8.0.10	10.8.0.8	UDP	828	55507 → 5001 Len=800
5	0.025228	10.8.0.10	10.8.0.8	UDP	828	55507 → 5001 Len=800
6	0.031751	10.8.0.10	10.8.0.8	UDP	828	55507 → 5001 Len=800
7	0.037655	10.8.0.10	10.8.0.8	UDP	828	55507 → 5001 Len=800
8	0.042894	10.8.0.10	10.8.0.8	UDP	828	55507 → 5001 Len=800
9	0.049256	10.8.0.10	10.8.0.8	UDP	828	55507 → 5001 Len=800
10	0.055043	10.8.0.10	10.8.0.8	UDP	828	55507 → 5001 Len=800

Figure 15: OpenVPN Unencapsulated packets for 800 bytes UDP test

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.56.102	192.168.56.101	OpenVPN	939	MessageType: P_DATA_V1
2	0.000938	192.168.56.101	192.168.56.103	OpenVPN	939	MessageType: P_DATA_V1
3	0.006437	192.168.56.102	192.168.56.101	OpenVPN	939	MessageType: P_DATA_V1
4	0.007195	192.168.56.101	192.168.56.103	OpenVPN	939	MessageType: P_DATA_V1
5	0.013510	192.168.56.102	192.168.56.101	OpenVPN	939	MessageType: P_DATA_V1
6	0.015983	192.168.56.101	192.168.56.103	OpenVPN	939	MessageType: P_DATA_V1
7	0.018708	192.168.56.102	192.168.56.101	OpenVPN	939	MessageType: P_DATA_V1
8	0.025117	192.168.56.102	192.168.56.101	OpenVPN	939	MessageType: P_DATA_V1
9	0.028410	192.168.56.101	192.168.56.103	OpenVPN	939	MessageType: P_DATA_V1
10	0.028416	192.168.56.101	192.168.56.103	OpenVPN	939	MessageType: P_DATA_V1

Figure 16: OpenVPN Encapsulated packets for 800 bytes UDP test

5.2.1.2 *IPSec*

The results for IPSec can be seen in table 11. Captures on the `ppp0` interface do not contain a link-layer overhead but a Linux cooked capture header of 16 bytes. The `eth0` captures do have a link layer header of 14 bytes. These overheads are corrected in the tables below. The IPSec results show an overhead between 88 and 76 for all different packet sizes. Notable is the consistency even for fragmented packets. An example of a trace can be seen in figure 17 without the encapsulation, and the same packets after the IPSec encryption in figure 18.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.16.1.30	172.16.1.1	UDP	844	Source port: 48299 Destination port: 5001
2	0.006666	172.16.1.30	172.16.1.1	UDP	844	Source port: 48299 Destination port: 5001
3	0.012579	172.16.1.30	172.16.1.1	UDP	844	Source port: 48299 Destination port: 5001
4	0.018484	172.16.1.30	172.16.1.1	UDP	844	Source port: 48299 Destination port: 5001
5	0.024624	172.16.1.30	172.16.1.1	UDP	844	Source port: 48299 Destination port: 5001
6	0.030813	172.16.1.30	172.16.1.1	UDP	844	Source port: 48299 Destination port: 5001
7	0.037068	172.16.1.30	172.16.1.1	UDP	844	Source port: 48299 Destination port: 5001
8	0.043135	172.16.1.30	172.16.1.1	UDP	844	Source port: 48299 Destination port: 5001
9	0.049124	172.16.1.30	172.16.1.1	UDP	844	Source port: 48299 Destination port: 5001
10	0.055153	172.16.1.30	172.16.1.1	UDP	844	Source port: 48299 Destination port: 5001

Figure 17: IPSEC Unencapsulated packets for 800 bytes UDP test

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0xb1824e2f)
2	0.006474	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0xb1824e2f)
3	0.012350	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0xb1824e2f)
4	0.018261	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0xb1824e2f)
5	0.024391	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0xb1824e2f)
6	0.030647	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0xb1824e2f)
7	0.036846	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0xb1824e2f)
8	0.042993	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0xb1824e2f)
9	0.048916	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0xb1824e2f)
10	0.055006	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0xb1824e2f)

Figure 18: IPSEC Encapsulated packets for 800 bytes UDP test

Results for all packet sizes are shown in table 11. The fixed overhead of 111 bytes is true for all packets that have a size less than the MTU. If the packet is fragmented, openVPN creates even more overhead, as can be seen at the 1600 and 3200 byte values.

Table 10: OpenVPN Encapsulation data result

	Encapsulated		Unencapsulated		
Packet Size	**Bytes**	**Packets**	**Bytes**	**Packets**	**Overhead per sent packet**
100	221	1	128	1	93
200	317	1	228	1	89
400	525	1	428	1	97
800	925	1	828	1	97
1600	1854	3	1648	2	103
3200	3599	5	3268	3	110

Table 11: IPSec Encapsulation data result

	Encapsulated		Unencapsulated		
Packet Size	**Bytes**	**Packets**	**Bytes**	**Packets**	**Overhead per sent packet**
100	216	1	128	1	88
200	312	1	228	1	84
400	504	1	428	1	76
800	904	1	828	1	76
1600	1808	3	1648	2	80
3200	3496	5	3268	3	76

5.2.2 *Compression*

5.2.2.1 *OpenVPN*

A large difference can be made by enabling the LZO compression in OpenVPN. An example can be seen in figure 19 with uncompressed packet with a size of 939 bytes. The compression is enabled in figure 20, where packets only have a size of 203 bytes. This however only occurs for data that is highly compressible.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.56.102	192.168.56.101	OpenVPN	939	MessageType: P_DATA_V1
2	0.000938	192.168.56.101	192.168.56.103	OpenVPN	939	MessageType: P_DATA_V1
3	0.006437	192.168.56.102	192.168.56.101	OpenVPN	939	MessageType: P_DATA_V1
4	0.007195	192.168.56.101	192.168.56.103	OpenVPN	939	MessageType: P_DATA_V1
5	0.013510	192.168.56.102	192.168.56.101	OpenVPN	939	MessageType: P_DATA_V1
6	0.015983	192.168.56.101	192.168.56.103	OpenVPN	939	MessageType: P_DATA_V1
7	0.018708	192.168.56.102	192.168.56.101	OpenVPN	939	MessageType: P_DATA_V1
8	0.025117	192.168.56.102	192.168.56.101	OpenVPN	939	MessageType: P_DATA_V1
9	0.028410	192.168.56.101	192.168.56.103	OpenVPN	939	MessageType: P_DATA_V1
10	0.028416	192.168.56.101	192.168.56.103	OpenVPN	939	MessageType: P_DATA_V1

Figure 19: Uncompressed encapsulated packets for 800 bytes UDP packets in OpenVPN

In a test with completely random packet payloads, this does not occur. An example of the encapsulated results is shown in figure 21.

No.	Time	Source	Destination	Protocol	Length	Info
13	2.987922	192.168.56.101	192.168.56.103	OpenVPN	203	MessageType: P_DATA_V1
14	2.987929	192.168.56.101	192.168.56.103	OpenVPN	203	MessageType: P_DATA_V1
15	2.988050	192.168.56.102	192.168.56.101	OpenVPN	203	MessageType: P_DATA_V1
16	2.988122	192.168.56.102	192.168.56.101	OpenVPN	203	MessageType: P_DATA_V1
17	2.989698	192.168.56.101	192.168.56.103	OpenVPN	203	MessageType: P_DATA_V1
18	2.989703	192.168.56.101	192.168.56.103	OpenVPN	203	MessageType: P_DATA_V1
19	2.993198	192.168.56.102	192.168.56.101	OpenVPN	203	MessageType: P_DATA_V1
20	2.995633	192.168.56.101	192.168.56.103	OpenVPN	203	MessageType: P_DATA_V1
21	2.999746	192.168.56.102	192.168.56.101	OpenVPN	203	MessageType: P_DATA_V1
22	3.010834	192.168.56.101	192.168.56.103	OpenVPN	203	MessageType: P_DATA_V1

Figure 20: Compressed encapsulated packets for 800 bytes UDP packets in OpenVPN

This gives no advantage to using LZO compression when the data is not further compressible.

No.	Time	Source	Destination	Protocol	Length	Info
4	0.019295	192.168.56.103	192.168.56.101	OpenVPN	939	MessageType: P_DATA_V1
5	0.022486	192.168.56.103	192.168.56.101	OpenVPN	939	MessageType: P_DATA_V1
6	0.029750	192.168.56.103	192.168.56.101	OpenVPN	939	MessageType: P_DATA_V1
7	0.034689	192.168.56.103	192.168.56.101	OpenVPN	939	MessageType: P_DATA_V1
8	0.040810	192.168.56.103	192.168.56.101	OpenVPN	939	MessageType: P_DATA_V1
9	0.047160	192.168.56.103	192.168.56.101	OpenVPN	939	MessageType: P_DATA_V1
10	0.053232	192.168.56.103	192.168.56.101	OpenVPN	939	MessageType: P_DATA_V1
11	0.059285	192.168.56.103	192.168.56.101	OpenVPN	939	MessageType: P_DATA_V1
12	0.065767	192.168.56.103	192.168.56.101	OpenVPN	939	MessageType: P_DATA_V1
13	0.071420	192.168.56.103	192.168.56.101	OpenVPN	939	MessageType: P_DATA_V1

Figure 21: Compressed encapsulated packets for 800 bytes random payload in OpenVPN

5.2.2.2 *IPSec*

Compression was enabled in IPSec by enabling the `compress=yes` option in the configuration for the client and the server. The results are similar to the OpenVPN results. If the data is compressible, a large difference can be seen. In the example below, we can see a that packets with a payload of 800 bytes result in a 918 byte packet if no compression is used, and a 182 byte packet where compression is used. This is an even larger decrease than was seen with OpenVPN.

Contrary to the compression used in OpenVPN, IPSec compression does not always result in packets of the same size. For packets with a payload size of 200 bytes, compressed packets of size 166 and 182 were found. This can be a result of some packets being better compressible than others; but implies a block size of 16 bytes for the compression.

If the payload is not compressible but entirely random, the results are the same as for OpenVPN. An example of packets with random payload with compression turned on can be seen in figure 24. No gain is made by compressing headers or anything else; the packets have the same size as their counterparts where compression is turned off.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0xb1824e2f)
2	0.006474	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0xb1824e2f)
3	0.012350	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0xb1824e2f)
4	0.018261	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0xb1824e2f)
5	0.024391	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0xb1824e2f)
6	0.030647	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0xb1824e2f)
7	0.036846	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0xb1824e2f)
8	0.042993	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0xb1824e2f)
9	0.048916	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0xb1824e2f)
10	0.055006	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0xb1824e2f)

Figure 22: Uncompressed encapsulated packets for 800 bytes UDP packets in IPSec

No.	Time	Source	Destination	Protocol	Length	Info
2	0.604318	192.168.56.102	192.168.56.101	ESP	182	ESP (SPI=0x90674c0d)
3	0.610720	192.168.56.102	192.168.56.101	ESP	182	ESP (SPI=0x90674c0d)
4	0.616809	192.168.56.102	192.168.56.101	ESP	182	ESP (SPI=0x90674c0d)
5	0.622756	192.168.56.102	192.168.56.101	ESP	182	ESP (SPI=0x90674c0d)
6	0.628855	192.168.56.102	192.168.56.101	ESP	182	ESP (SPI=0x90674c0d)
7	0.635179	192.168.56.102	192.168.56.101	ESP	182	ESP (SPI=0x90674c0d)
8	0.641144	192.168.56.102	192.168.56.101	ESP	182	ESP (SPI=0x90674c0d)
9	0.647354	192.168.56.102	192.168.56.101	ESP	182	ESP (SPI=0x90674c0d)
10	0.653655	192.168.56.102	192.168.56.101	ESP	182	ESP (SPI=0x90674c0d)

Figure 23: Compressed encapsulated packets for 800 bytes UDP packets in IPSec

5.2.3 *IPSEc Encapsulation*

On both the server and the client, the `forceencaps=yes` option was enabled. This causes IPSec to fake the NAT detection payloads. The NAT is thus detected and UDP encapsulation is enabled:

```
003 "L2TP-PSK" #1: NAT-Traversal: Result using draft-ietf-ipsec-
    nat-t-ike (MacOS X): both are NATed
```

The UDP encapsulation adds another UDP header of 8 bytes in overhead. An example of these packets can be seen in figure 25. With this option enabled, this makes the overhead of IPSec encapsulation rise from between 88 and 76 bytes to between 96 and 84 bytes. This makes it very comparable to OpenVPN with the overhead between 97 and 89 bytes

5.3 FINAL CONSIDERATIONS

OpenVPN and IPSec seems to add quite some overhead, but as the overhead is not proportional to the payload size, this can be limited by making packets as large as possible. This way, the overhead is relatively small in relation to the payload. In the chapter about VoIP, a method was shown to accomplish this through packetization options. However, when fragmentation occurs, the overhead becomes even more significant, so this should be avoided. Compression does not seem to provide much relief to the overhead if the payload is not compressible, but has large benefits for other payloads.

Both VPN solutions provide a way to compress the data before it is encrypted. From the results it can be seen that large gains can be made by enabling compression, but it only works on their payloads. It is therefore very dependant on how compressible the payload is. If the payloads are not compressible, this option can better be turned

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0x90674c0d)
2	0.006210	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0x90674c0d)
3	0.006464	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0x90674c0d)
4	0.012498	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0x90674c0d)
5	0.018552	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0x90674c0d)
6	0.024625	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0x90674c0d)
7	0.031006	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0x90674c0d)
8	0.036854	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0x90674c0d)
9	0.043445	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0x90674c0d)
10	0.049209	192.168.56.102	192.168.56.101	ESP	918	ESP (SPI=0x90674c0d)

Figure 24: Compressed encapsulated packets for 800 bytes random payload in IPSec

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.56.102	192.168.56.101	ESP	926	ESP (SPI=0x5e78e8c8)
2	0.005647	192.168.56.102	192.168.56.101	ESP	926	ESP (SPI=0x5e78e8c8)
3	0.012052	192.168.56.102	192.168.56.101	ESP	926	ESP (SPI=0x5e78e8c8)
4	0.019198	192.168.56.102	192.168.56.101	ESP	926	ESP (SPI=0x5e78e8c8)
5	0.023964	192.168.56.102	192.168.56.101	ESP	926	ESP (SPI=0x5e78e8c8)
6	0.030232	192.168.56.102	192.168.56.101	ESP	926	ESP (SPI=0x5e78e8c8)
7	0.036310	192.168.56.102	192.168.56.101	ESP	926	ESP (SPI=0x5e78e8c8)
8	0.042915	192.168.56.102	192.168.56.101	ESP	926	ESP (SPI=0x5e78e8c8)
9	0.049170	192.168.56.102	192.168.56.101	ESP	926	ESP (SPI=0x5e78e8c8)
10	0.054617	192.168.56.102	192.168.56.101	ESP	926	ESP (SPI=0x5e78e8c8)

Figure 25: UDP encapsulated ESP packets for 800 bytes in IPSec

off to save on CPU resource consumption, which will cost battery power. VoIP encoding can cause the payloads to be compressed quite a bit already, making this an option that might not prove very useful. It should however be tested under VoIP traffic, as the packetization of VoIP packets might not be entirely effective in compressing and encoding the voice data.

IPSec has the option to be encapsulated in UDP packets for NAT Traversal. This option should be enabled for a mobile phone as it might encounter NATs very frequently. This also makes IPSec even more comparable to OpenVPN, as it has similar NAT traversing properties.

To compare both solutions, we look at more than just the raw numbers of overhead. IPSec and OpenVPN have shown to be very comparable in this aspect, especially if IPSec has UDP encapsulation enabled. OpenVPN has shown to be much more flexible to install and set up. OpenVPN also has the benefit of working on the application layer, which does not require access further in the stack, which can be a problem on some devices such as mobile phones. For our use case, OpenVPN is chosen because of this.

6 TUNNELED VOIP

Combining all of the research from previous chapters, now the focus lies on tunneling VoIP over VPN tunnels. Much of the previous research provides an indication as to what parameters are optimal.

The research into VoIP characteristics provided important results in packetization options. The research into VPNs showed that compression can be very important if no large delays are incurred.

In this chapter, the best way to tunnel VoIP over VPN is determined. The question **How can VoIP be secured with a tunnel?** is answered. Additionally, the impact of this solution for a mobile device and solutions to problems are discussed.

6.1 METHODS

The VoIP setup from section 3.1.1 is used again. This time, the connection to the clients is tunneled over a VPN. The setup is shown in figure 26, with the transparent green overlay depicting the VPN tunnel.

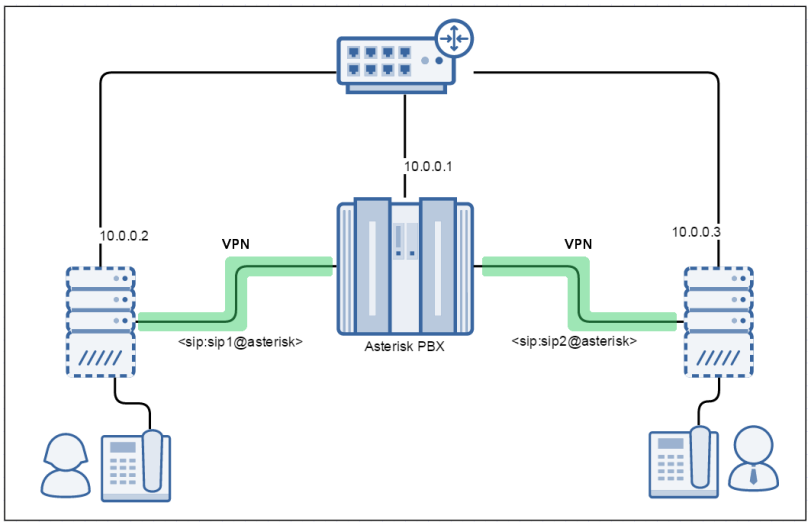


Figure 26: VoIP over VPN setup

In chapter 5, IPSec and OpenVPN were compared. As they are very similar in terms over overhead performance, OpenVPN was chosen as a suitable VPN solution to test with. In this test, OpenVPN will be configured from the clients to the server running Asterisk.

The voice sample as used in chapter 3 will be used again. After the VoIP server is setup, a client will connect and the voice sampled played through again. The call will once again be packet captured

and analyzed. The capture will be done on both the `eth` and `tun` interface to see both the encapsulated and unencapsulated packets. G.722 will again be used as the media codec.

To confirm the hypothesis from the results of chapter 4, where compression was used, the sample will also be played with compression turned on. The hypothesis is that, since VoIP is already encoded efficiently, compression will not provide significant improvement.

The conclusion from chapter 3 about packetization was that 40ms of framing size would be a good setting. To test this out, we tested the VoIP call for framing sizes of 20, 40, 60 and 80ms. Larger values would incur a large delay on the connection, making the voice quality degrade too much.

6.2 RESULTS

6.2.1 *VoIP Baseline*

As a baseline, the VoIP call was done over the connection without the VPN. This was done for framing sizes of 20, 40, 60 and 80ms. Several characteristics are presented to be compared: total bytes sent, total packets sent and packet size. The call length is 30 seconds for these measurements. The baseline results are presented in table 12 and in a graph in figure 27.

Table 12: VoIP baseline without VPN

Frame size	Total bytes	Total packets	Packet size
20	183171	853	214
40	167845	454	374
60	135383	263	534
80	165843	251	694

It can be seen that while the packet size increases linearly, the total packet count does not decrease in the same manner. This means that there is no direct trade-off between increasing the framing size and the total amount of packets. These results provide a reference for the measurements over OpenVPN, with and without compression.

6.2.2 *VoIP over OpenVPN*

OpenVPN was enabled on both clients and their SIP clients reconfigured to connect to the server over the `tun` interface. The SIP registration was successful and calls could be established. A problem occurred with establishing the RTP media session however. While both clients seemed to think they were connected, no data could be transferred over RTP from client to client. After some deliberation

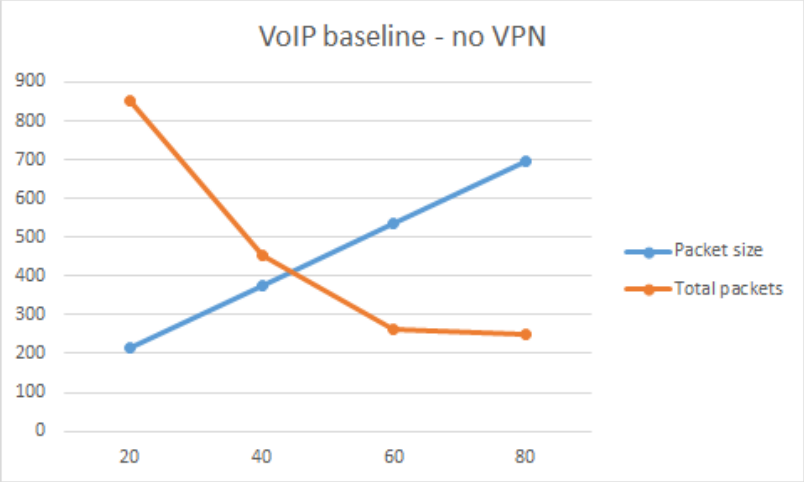


Figure 27: VoIP baseline without VPN

and packet analyses, the media channel settings of Asterisk were reconfigured to `directmedia=no`. This fixed the problem and the clients connected over RTP as well.

The results from this analysis are shown in table 13 and in figure 28.

Table 13: VoIP with VPN tunnel

Frame size	Total bytes	Total packets	Packet size
20	285988	954	299
40	209728	469	459
60	185547	351	619
80	169691	245	779

6.2.3 *VoIP over OpenVPN with Compression*

Compression using LZO was then configured for the OpenVPN server and clients. The measurements were performed again, and the results are shown in table 14. For the compression test, only the packet size is listed, to see if any gains can be made. For each frame, all RTP packets in the stream seem to have the same packet size (excluding control packets).

Absolutely no gain is seen for the packets that are encoded with G.277. The hypothesis was that a gain could maybe be made because framing might concatenate 10ms chunks of voice data. This has proven to be wrong, as no compression gains could be made. However, other stray packets in the stream such as RTCP packets do benefit from the compression.

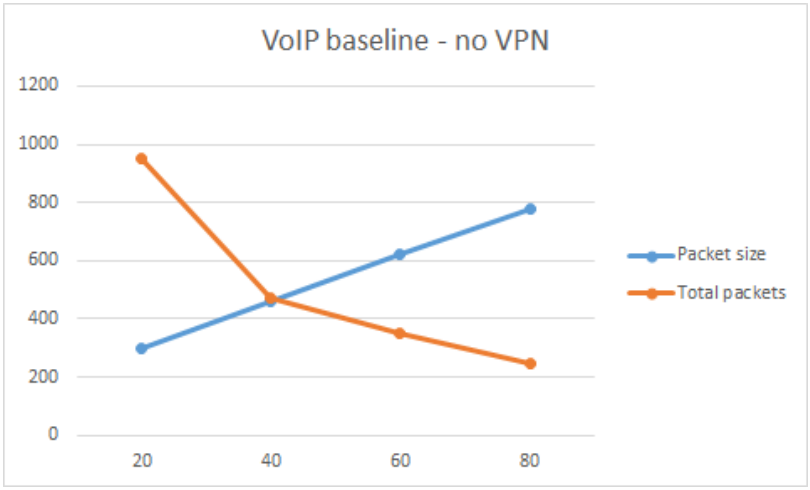


Figure 28: VoIP with VPN tunnel

Table 14: VoIP with VPN tunnel using compression

Frame size	Packet size
20	299
40	459
60	619
80	779

6.3 MOBILE ISSUES

6.3.1 *NAT Traversal*

As mobile network connections are generally not reliable and stable, they can produce a lot of jitter on a VoIP connection. WiFi networks are therefore generally a good alternative. These are the two network types on which this research is focused. These networks can employ many different NAT types. A mobile phone that can reside behind different NATs should use an application that can traverse them for a good connection.

IPSec generally does not lend itself to NAT Traversal very well, as in ESP mode, the port number is encrypted and cannot be read or changed by the NAT device. While NAT traversal options exist in IPSec extensions, it is not a perfect solution. The NAT traversal in IPSsec will encapsulate the packets in UDP packets. The packets are then routed over port 4500. This creates the extra overhead of the UDP packet. OpenVPN, working in the application layer, does not have this problem of NAT traversal. By default, OpenVPN works over UDP, which can be translated in almost every NAT.

Another issue that arises from NAT Traversal is the timeout most NAT devices employ. In general, UDP NAT translations have a very

short timeout, ranging from a few minutes to a few hours. TCP timeouts are generally much longer, sometimes even 24 hours. To keep such a translation active, a keep-alive packet must be sent. To do this on a mobile phone would reduce the battery time greatly, as the phone would have to wake up for every packet that is sent.

6.3.2 *Battery consumption*

We have identified two factors that might be important in mobile battery consumption: reducing wake frequency and resource consumption. We have seen that most mobile VoIP solutions employ heavy encryption schemes, such as the one seen in WhatsApp. This has not shown to be a great burden for mobile phones. In the background research, it was also shown that many phones currently on the market have very powerful CPU's. In almost all models, a set of energy efficient CPU's is paired with a more powerful set. The result of this is that the encryption used by a VPN is almost negligible.

A very important factor however, is the waking frequency. For any solution, the sleep periods should be maximized. This means that the phone should keep keep-alive packets to a minimum. In the proposed solution we have three elements that should be optimized for keep-alive packets: the VPN, the SIP registration and the NAT translation.

For OpenVPN, a keepalive can be configured in the settings, as can be seen in appendix D.1.1. The `keepalive` setting has two parameters, the first requiring clients to send a keepalive packet every x seconds. The second parameter specifies the amount of time after which OpenVPN will detect a client as down. This can be configured to quite a large value, such as 5 or 10 minutes for the keepalive, and 30 minutes for the down detection. This improves the length of the sleep cycle greatly. The disadvantage of this is that the OpenVPN server might still believe the client is up, when the phone has actually been disconnected. With a mobile phone, this can happen a lot, and this situation should therefore be handled correctly. Under the usecase of VoIP for VPN, this is no problem. If the PBX does not receive a reply to the `INVITE` packets, it will conclude that the host is down and the call will be declined properly.

Asterisk keepalive can also be configured in its settings. This is a server-side option to send keepalive packets to clients, which is not standard. This option can be very useful if a number of different VoIP clients are used, as they will not have to be configured individually for keep-alive functionality. Asterisk does not require SIP registrations to be updated very frequently.

NAT timeouts can be a big problem for mobile phones, as they can be very short, causing the phone to wake up very frequently. This is generally a problem with protocols running on top of UDP,

as timeouts for UDP can be very short. A simple solution might be to use TCP as the transport protocol for OpenVPN when it is not in use. While switching transport protocol on the fly is not an option currently, mobile phones might benefit from it greatly. TCP should not be used for OpenVPN when more data is sent, as it will result in a large overhead and throttling.

6.4 FINAL CONSIDERATIONS

We have seen that VoIP over OpenVPN is a very feasible solution to secure the channel. While the VPN tunneling does result in some overhead, the effect of this can be greatly reduced by leaving the framing size as large as the connection allows. Only a range of framing sizes should be used however. Low framing size values result in a very large number of packets that have to be sent, which can cause other delay problems such as congestion. Large framing sizes should also be avoided, as they incur a delay of their own, which can be noticeable. The accepted values proposed in this research are 20 to 80ms. A round trip time measurement could be made at the beginning of a conversation, and during a call, adapting the framing size to fit the connection. An accepted maximum delay value should be set (e.g. 100ms). The connection will incur a certain delay, which can be subtracted from this accepted maximum (e.g. 40ms). A certain amount of extra delay might still be acceptable on this connection (e.g. 60ms). Within the range of this extra delay, framing sizes could be set to larger values.

7 DISCUSSION

7.1 LIMITATIONS

This research was done mostly in a lab environment, running VoIP and VPN servers in VMs in a simulated network. While much of the theories discussed also apply to real-world scenario's, they have not been tested in such an environment. An exception to this is the research toward current mobile VoIP solutions, which was performed in an entirely realistic environment. The results would have benefited from a real-world implementation and testing it against different types of networks (mobile, WiFi) and different NAT configurations.

7.2 CONCLUSION

Four different subjects were addressed in the previous chapters, corresponding to the four main research questions. They have shed light on four different subjects: VoIP characteristics, current mobile VoIP solutions, VPN tunnel characteristics and VoIP over VPN. Beside this, the background of these subjects was also discussed in a literature study.

In chapter 3, the first research question was addressed. An Asterisk PBX server was set up with two clients. Different codecs were tested and analysed, and packetization was discovered as an important factor for VoIP performance. In the chapter following this, chapter 4, we analysed how current leading mobile VoIP applications secured their calls. It was shown that while their connection encryption is excellent, end-to-end encryption is rarely actually established. In chapter 5 two different VPNs were compared: OpenVPN and IPSec. The results showed that their can be very similar in performance, especially regarding overhead. IPSec showed to be less flexible and quite difficult to set up for mobile phones however, and OpenVPN was chosen to be an effective VPN solution. Finally, in chapter 6, the research came together to set up a primary goal of this research: VoIP over VPN. From chapter 3, VoIP codecs and packetization options were used. From chapter 5, OpenVPN settings were configured to be optimal for the VoIP usecase. Compression was also tested for VoIP packetization, but proved to result in no improvements.

7.3 OPEN CHALLENGES AND FUTURE WORK

One of the interesting additions to OpenVPN would be the option to switch transport protocol when necessary. TCP could be used in sleep states, and UDP when the network is being actively used. This could be implemented and researched, as it might provide great improvements for mobile phones.

Another point of research would be to bring the proposed solutions into real-world scenario's, as discussed in the limitations section. It would provide even more insight into the problems that can arise from different NAT devices and networks.

A

RTP RESULTS

The results for the RTP tests are listed below. For many different types of codes, results are listed.

192.168.56.105:7078 ↔
192.168.56.103:13640

Forward

SSRC	0x2b5a3f8a
Max Delta	60.47 ms @ 8378
Max Jitter	17.41 ms
Mean Jitter	14.65 ms
Max Skew	-32.77 ms
RTP Packets	3332
Expected	3332
Lost	0 (0.00 %)
Seq Errs	0
Duration	66.61 s
Clock Drift	-18 ms
Freq Drift	31992 Hz (-0.03 %)

Reverse

SSRC	0x2b5a3f8a
Max Delta	81.13 ms @ 7311
Max Jitter	21.35 ms
Mean Jitter	16.18 ms
Max Skew	60.92 ms
RTP Packets	3328
Expected	3328
Lost	0 (0.00 %)
Seq Errs	0
Duration	66.52 s
Clock Drift	-17 ms
Freq Drift	31992 Hz (-0.03 %)

2 streams found.

192.168.56.103:11110 ↔
192.168.56.105:7078

Forward

SSRC	0x7fd8f209
Max Delta	490.32 ms @ 11276
Max Jitter	59.49 ms
Mean Jitter	17.55 ms
Max Skew	-812.20 ms
RTP Packets	3327
Expected	3327
Lost	0 (0.00 %)
Seq Errs	0
Duration	66.48 s
Clock Drift	-251 ms
Freq Drift	31879 Hz (-0.38 %)

Reverse

SSRC	0x7fd8f209
Max Delta	102.18 ms @ 6838
Max Jitter	19.99 ms
Mean Jitter	14.64 ms
Max Skew	-106.12 ms
RTP Packets	3332
Expected	3332
Lost	0 (0.00 %)
Seq Errs	0
Duration	66.63 s
Clock Drift	-7 ms
Freq Drift	31997 Hz (-0.01 %)

2 streams found.

192.168.56.103:19200 ↔
192.168.56.105:7078

Forward

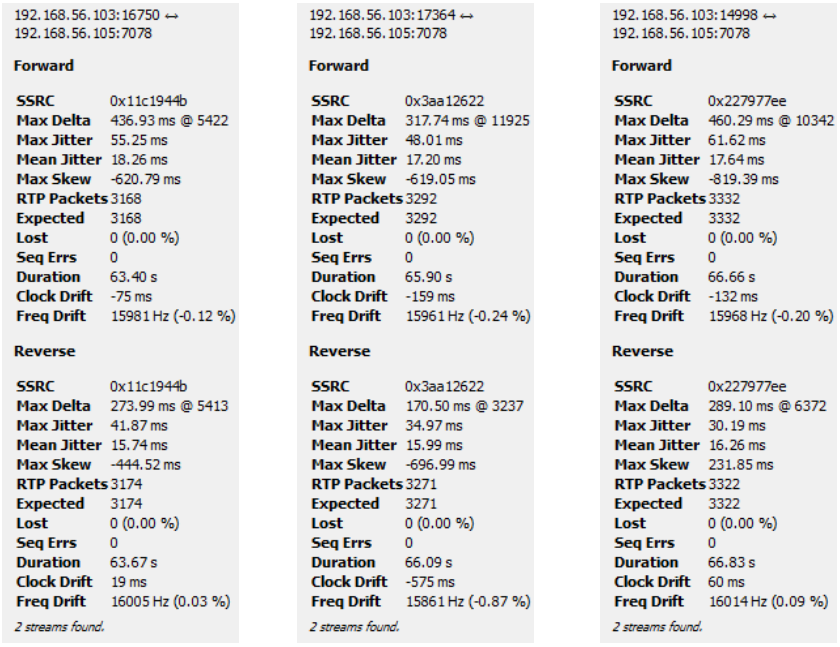
SSRC	0x47d90b1f
Max Delta	740.46 ms @ 11732
Max Jitter	130.18 ms
Mean Jitter	21.52 ms
Max Skew	-1339.72 ms
RTP Packets	3292
Expected	3292
Lost	0 (0.00 %)
Seq Errs	0
Duration	65.80 s
Clock Drift	-374 ms
Freq Drift	31818 Hz (-0.57 %)

Reverse

SSRC	0x47d90b1f
Max Delta	351.47 ms @ 11635
Max Jitter	56.63 ms
Mean Jitter	16.48 ms
Max Skew	-520.70 ms
RTP Packets	3272
Expected	3272
Lost	0 (0.00 %)
Seq Errs	0
Duration	66.03 s
Clock Drift	-67 ms
Freq Drift	31967 Hz (-0.10 %)

2 streams found.

Figure 29: Speex 32kHz results



192.168.56.103:16750 ↔
192.168.56.105:7078

Forward

SSRC	0x11c1944b
Max Delta	436.93 ms @ 5422
Max Jitter	55.25 ms
Mean Jitter	18.26 ms
Max Skew	-620.79 ms
RTP Packets	3168
Expected	3168
Lost	0 (0.00 %)
Seq Errs	0
Duration	63.40 s
Clock Drift	-75 ms
Freq Drift	15981 Hz (-0.12 %)

Reverse

SSRC	0x11c1944b
Max Delta	273.99 ms @ 5413
Max Jitter	41.87 ms
Mean Jitter	15.74 ms
Max Skew	-444.52 ms
RTP Packets	3174
Expected	3174
Lost	0 (0.00 %)
Seq Errs	0
Duration	63.67 s
Clock Drift	19 ms
Freq Drift	16005 Hz (0.03 %)

2 streams found.

192.168.56.103:17364 ↔
192.168.56.105:7078

Forward

SSRC	0x3aa12622
Max Delta	317.74 ms @ 11925
Max Jitter	48.01 ms
Mean Jitter	17.20 ms
Max Skew	-619.05 ms
RTP Packets	3292
Expected	3292
Lost	0 (0.00 %)
Seq Errs	0
Duration	65.90 s
Clock Drift	-159 ms
Freq Drift	15961 Hz (-0.24 %)

Reverse

SSRC	0x3aa12622
Max Delta	170.50 ms @ 3237
Max Jitter	34.97 ms
Mean Jitter	15.99 ms
Max Skew	-696.99 ms
RTP Packets	3271
Expected	3271
Lost	0 (0.00 %)
Seq Errs	0
Duration	66.09 s
Clock Drift	-575 ms
Freq Drift	15861 Hz (-0.87 %)

2 streams found.

192.168.56.103:14998 ↔
192.168.56.105:7078

Forward

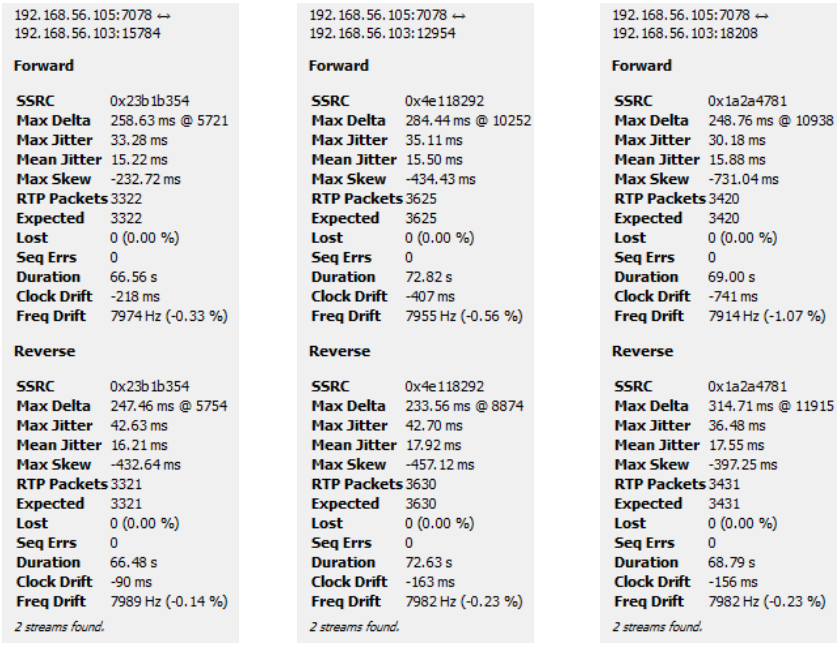
SSRC	0x227977ee
Max Delta	460.29 ms @ 10342
Max Jitter	61.62 ms
Mean Jitter	17.64 ms
Max Skew	-819.39 ms
RTP Packets	3332
Expected	3332
Lost	0 (0.00 %)
Seq Errs	0
Duration	66.66 s
Clock Drift	-132 ms
Freq Drift	15968 Hz (-0.20 %)

Reverse

SSRC	0x227977ee
Max Delta	289.10 ms @ 6372
Max Jitter	30.19 ms
Mean Jitter	16.26 ms
Max Skew	231.85 ms
RTP Packets	3322
Expected	3322
Lost	0 (0.00 %)
Seq Errs	0
Duration	66.83 s
Clock Drift	60 ms
Freq Drift	16014 Hz (0.09 %)

2 streams found.

Figure 30: Speex 16kHz results



192.168.56.105:7078 ↔
192.168.56.103:15784

Forward

SSRC	0x23b1b354
Max Delta	258.63 ms @ 5721
Max Jitter	33.28 ms
Mean Jitter	15.22 ms
Max Skew	-232.72 ms
RTP Packets	3322
Expected	3322
Lost	0 (0.00 %)
Seq Errs	0
Duration	66.56 s
Clock Drift	-218 ms
Freq Drift	7974 Hz (-0.33 %)

Reverse

SSRC	0x23b1b354
Max Delta	247.46 ms @ 5754
Max Jitter	42.63 ms
Mean Jitter	16.21 ms
Max Skew	-432.64 ms
RTP Packets	3321
Expected	3321
Lost	0 (0.00 %)
Seq Errs	0
Duration	66.48 s
Clock Drift	-90 ms
Freq Drift	7989 Hz (-0.14 %)

2 streams found.

192.168.56.105:7078 ↔
192.168.56.103:12954

Forward

SSRC	0x4e118292
Max Delta	284.44 ms @ 10252
Max Jitter	35.11 ms
Mean Jitter	15.50 ms
Max Skew	-434.43 ms
RTP Packets	3625
Expected	3625
Lost	0 (0.00 %)
Seq Errs	0
Duration	72.82 s
Clock Drift	-407 ms
Freq Drift	7955 Hz (-0.56 %)

Reverse

SSRC	0x4e118292
Max Delta	233.56 ms @ 8874
Max Jitter	42.70 ms
Mean Jitter	17.92 ms
Max Skew	-457.12 ms
RTP Packets	3630
Expected	3630
Lost	0 (0.00 %)
Seq Errs	0
Duration	72.63 s
Clock Drift	-163 ms
Freq Drift	7982 Hz (-0.23 %)

2 streams found.

192.168.56.105:7078 ↔
192.168.56.103:18208

Forward

SSRC	0x1a2a4781
Max Delta	248.76 ms @ 10938
Max Jitter	30.18 ms
Mean Jitter	15.88 ms
Max Skew	-731.04 ms
RTP Packets	3420
Expected	3420
Lost	0 (0.00 %)
Seq Errs	0
Duration	69.00 s
Clock Drift	-741 ms
Freq Drift	7914 Hz (-1.07 %)

Reverse

SSRC	0x1a2a4781
Max Delta	314.71 ms @ 11915
Max Jitter	36.48 ms
Mean Jitter	17.55 ms
Max Skew	-397.25 ms
RTP Packets	3431
Expected	3431
Lost	0 (0.00 %)
Seq Errs	0
Duration	68.79 s
Clock Drift	-156 ms
Freq Drift	7982 Hz (-0.23 %)

2 streams found.

Figure 31: Speex 8kHz results

192.168.56.104:7078 ↔
192.168.56.103:14672

Forward

SSRC	0x77763199
Max Delta	307.36 ms @ 7626
Max Jitter	38.45 ms
Mean Jitter	17.56 ms
Max Skew	-479.67 ms
RTP Packets	3545
Expected	3545
Lost	0 (0.00 %)
Seq Errs	0
Duration	70.96 s
Clock Drift	-217 ms
Freq Drift	7976 Hz (-0.31 %)

Reverse

SSRC	0x77763199
Max Delta	233.72 ms @ 7595
Max Jitter	28.01 ms
Mean Jitter	15.50 ms
Max Skew	-369.86 ms
RTP Packets	3535
Expected	3535
Lost	0 (0.00 %)
Seq Errs	0
Duration	70.94 s
Clock Drift	-342 ms
Freq Drift	7961 Hz (-0.48 %)

2 streams found.

192.168.56.104:7078 ↔
192.168.56.103:12144

Forward

SSRC	0x021660f5
Max Delta	178.61 ms @ 11872
Max Jitter	32.93 ms
Mean Jitter	17.23 ms
Max Skew	-278.49 ms
RTP Packets	3288
Expected	3288
Lost	0 (0.00 %)
Seq Errs	0
Duration	65.79 s
Clock Drift	-75 ms
Freq Drift	7991 Hz (-0.11 %)

Reverse

SSRC	0x021660f5
Max Delta	224.74 ms @ 10983
Max Jitter	29.16 ms
Mean Jitter	15.85 ms
Max Skew	209.76 ms
RTP Packets	3279
Expected	3279
Lost	0 (0.00 %)
Seq Errs	0
Duration	65.76 s
Clock Drift	104 ms
Freq Drift	8013 Hz (0.16 %)

2 streams found.

192.168.56.104:7078 ↔
192.168.56.103:10062

Forward

SSRC	0x4501d025
Max Delta	362.26 ms @ 3593
Max Jitter	54.85 ms
Mean Jitter	17.44 ms
Max Skew	-682.91 ms
RTP Packets	3272
Expected	3272
Lost	0 (0.00 %)
Seq Errs	0
Duration	65.57 s
Clock Drift	-268 ms
Freq Drift	7967 Hz (-0.41 %)

Reverse

SSRC	0x4501d025
Max Delta	188.36 ms @ 3527
Max Jitter	31.03 ms
Mean Jitter	16.06 ms
Max Skew	-222.99 ms
RTP Packets	3274
Expected	3274
Lost	0 (0.00 %)
Seq Errs	0
Duration	65.55 s
Clock Drift	-122 ms
Freq Drift	7985 Hz (-0.19 %)

2 streams found.

Figure 32: μ-law results

192.168.56.103:19194 ↔
192.168.56.105:7078

Forward

SSRC	0x4f0bc53a
Max Delta	95.11 ms @ 2072
Max Jitter	27.51 ms
Mean Jitter	16.60 ms
Max Skew	-70.53 ms
RTP Packets	3314
Expected	3314
Lost	0 (0.00 %)
Seq Errs	0
Duration	66.23 s
Clock Drift	-6 ms
Freq Drift	7999 Hz (-0.01 %)

Reverse

SSRC	0x4f0bc53a
Max Delta	77.30 ms @ 10979
Max Jitter	20.23 ms
Mean Jitter	14.66 ms
Max Skew	-43.91 ms
RTP Packets	3323
Expected	3323
Lost	0 (0.00 %)
Seq Errs	0
Duration	66.44 s
Clock Drift	-4 ms
Freq Drift	8000 Hz (-0.01 %)

2 streams found.

192.168.56.105:7078 ↔
192.168.56.103:13152

Forward

SSRC	0x08600682
Max Delta	86.03 ms @ 6287
Max Jitter	22.19 ms
Mean Jitter	15.35 ms
Max Skew	-54.92 ms
RTP Packets	4003
Expected	4003
Lost	0 (0.00 %)
Seq Errs	0
Duration	80.01 s
Clock Drift	-2 ms
Freq Drift	8000 Hz (-0.00 %)

Reverse

SSRC	0x08600682
Max Delta	111.10 ms @ 10057
Max Jitter	25.56 ms
Mean Jitter	16.16 ms
Max Skew	-103.91 ms
RTP Packets	3995
Expected	3995
Lost	0 (0.00 %)
Seq Errs	0
Duration	79.84 s
Clock Drift	-14 ms
Freq Drift	7999 Hz (-0.02 %)

2 streams found.

192.168.56.105:7078 ↔
192.168.56.103:13078

Forward

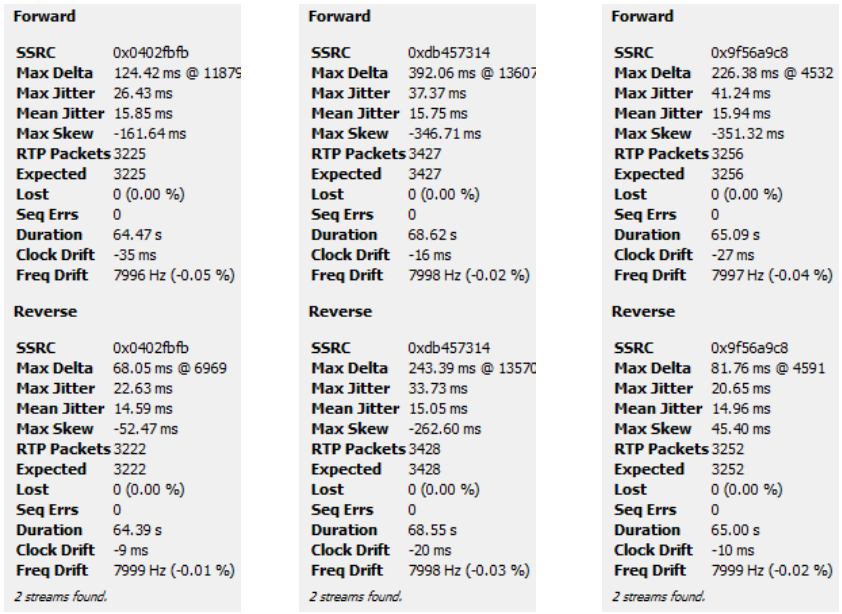
SSRC	0x256b48e8
Max Delta	99.17 ms @ 8595
Max Jitter	21.55 ms
Mean Jitter	15.08 ms
Max Skew	-85.17 ms
RTP Packets	3509
Expected	3509
Lost	0 (0.00 %)
Seq Errs	0
Duration	70.15 s
Clock Drift	-5 ms
Freq Drift	7999 Hz (-0.01 %)

Reverse

SSRC	0x256b48e8
Max Delta	100.97 ms @ 8974
Max Jitter	23.28 ms
Mean Jitter	16.23 ms
Max Skew	-93.93 ms
RTP Packets	3504
Expected	3504
Lost	0 (0.00 %)
Seq Errs	0
Duration	70.05 s
Clock Drift	-4 ms
Freq Drift	8000 Hz (-0.01 %)

2 streams found.

Figure 33: A-law results



Forward

SSRC	0x0402fbfb
Max Delta	124.42 ms @ 11879
Max Jitter	26.43 ms
Mean Jitter	15.85 ms
Max Skew	-161.64 ms
RTP Packets	3225
Expected	3225
Lost	0 (0.00 %)
Seq Errs	0
Duration	64.47 s
Clock Drift	-35 ms
Freq Drift	7996 Hz (-0.05 %)

Reverse

SSRC	0x0402fbfb
Max Delta	68.05 ms @ 6969
Max Jitter	22.63 ms
Mean Jitter	14.59 ms
Max Skew	-52.47 ms
RTP Packets	3222
Expected	3222
Lost	0 (0.00 %)
Seq Errs	0
Duration	64.39 s
Clock Drift	-9 ms
Freq Drift	7999 Hz (-0.01 %)

2 streams found.

Forward

SSRC	0xdb457314
Max Delta	392.06 ms @ 13607
Max Jitter	37.37 ms
Mean Jitter	15.75 ms
Max Skew	-346.71 ms
RTP Packets	3427
Expected	3427
Lost	0 (0.00 %)
Seq Errs	0
Duration	68.62 s
Clock Drift	-16 ms
Freq Drift	7998 Hz (-0.02 %)

Reverse

SSRC	0xdb457314
Max Delta	243.39 ms @ 13570
Max Jitter	33.73 ms
Mean Jitter	15.05 ms
Max Skew	-262.60 ms
RTP Packets	3428
Expected	3428
Lost	0 (0.00 %)
Seq Errs	0
Duration	68.55 s
Clock Drift	-20 ms
Freq Drift	7998 Hz (-0.03 %)

2 streams found.

Forward

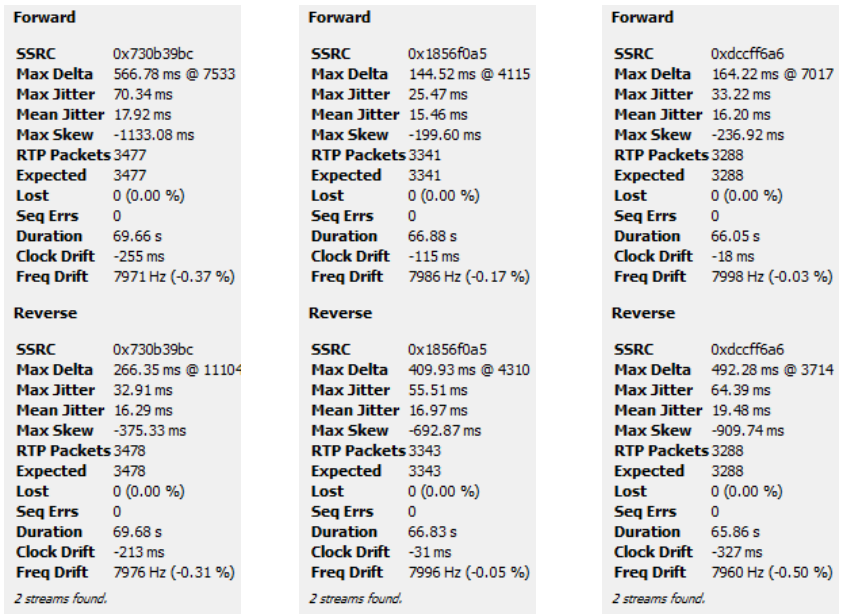
SSRC	0x9f56a9c8
Max Delta	226.38 ms @ 4532
Max Jitter	41.24 ms
Mean Jitter	15.94 ms
Max Skew	-351.32 ms
RTP Packets	3256
Expected	3256
Lost	0 (0.00 %)
Seq Errs	0
Duration	65.09 s
Clock Drift	-27 ms
Freq Drift	7997 Hz (-0.04 %)

Reverse

SSRC	0x9f56a9c8
Max Delta	81.76 ms @ 4591
Max Jitter	20.65 ms
Mean Jitter	14.96 ms
Max Skew	45.40 ms
RTP Packets	3252
Expected	3252
Lost	0 (0.00 %)
Seq Errs	0
Duration	65.00 s
Clock Drift	-10 ms
Freq Drift	7999 Hz (-0.02 %)

2 streams found.

Figure 34: G.722 results



Forward

SSRC	0x730b39bc
Max Delta	566.78 ms @ 7533
Max Jitter	70.34 ms
Mean Jitter	17.92 ms
Max Skew	-1133.08 ms
RTP Packets	3477
Expected	3477
Lost	0 (0.00 %)
Seq Errs	0
Duration	69.66 s
Clock Drift	-255 ms
Freq Drift	7971 Hz (-0.37 %)

Reverse

SSRC	0x730b39bc
Max Delta	266.35 ms @ 11104
Max Jitter	32.91 ms
Mean Jitter	16.29 ms
Max Skew	-375.33 ms
RTP Packets	3478
Expected	3478
Lost	0 (0.00 %)
Seq Errs	0
Duration	69.68 s
Clock Drift	-213 ms
Freq Drift	7976 Hz (-0.31 %)

2 streams found.

Forward

SSRC	0x1856f0a5
Max Delta	144.52 ms @ 4115
Max Jitter	25.47 ms
Mean Jitter	15.46 ms
Max Skew	-199.60 ms
RTP Packets	3341
Expected	3341
Lost	0 (0.00 %)
Seq Errs	0
Duration	66.88 s
Clock Drift	-115 ms
Freq Drift	7986 Hz (-0.17 %)

Reverse

SSRC	0x1856f0a5
Max Delta	409.93 ms @ 4310
Max Jitter	55.51 ms
Mean Jitter	16.97 ms
Max Skew	-692.87 ms
RTP Packets	3343
Expected	3343
Lost	0 (0.00 %)
Seq Errs	0
Duration	66.83 s
Clock Drift	-31 ms
Freq Drift	7996 Hz (-0.05 %)

2 streams found.

Forward

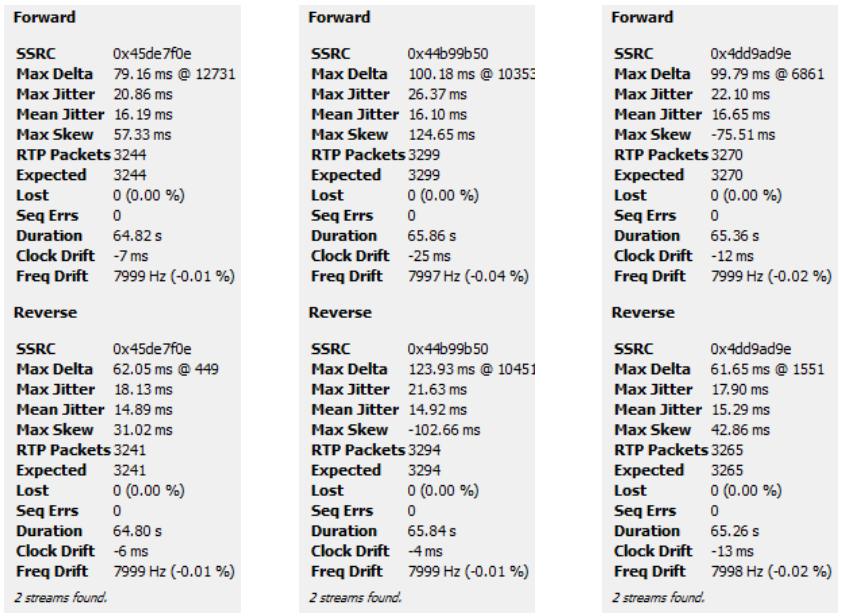
SSRC	0xdccff6a6
Max Delta	164.22 ms @ 7017
Max Jitter	33.22 ms
Mean Jitter	16.20 ms
Max Skew	-236.92 ms
RTP Packets	3288
Expected	3288
Lost	0 (0.00 %)
Seq Errs	0
Duration	66.05 s
Clock Drift	-18 ms
Freq Drift	7998 Hz (-0.03 %)

Reverse

SSRC	0xdccff6a6
Max Delta	492.28 ms @ 3714
Max Jitter	64.39 ms
Mean Jitter	19.48 ms
Max Skew	-909.74 ms
RTP Packets	3288
Expected	3288
Lost	0 (0.00 %)
Seq Errs	0
Duration	65.86 s
Clock Drift	-327 ms
Freq Drift	7960 Hz (-0.50 %)

2 streams found.

Figure 35: G.726 results



Forward

SSRC	0x45de7f0e
Max Delta	79.16 ms @ 12731
Max Jitter	20.86 ms
Mean Jitter	16.19 ms
Max Skew	57.33 ms
RTP Packets	3244
Expected	3244
Lost	0 (0.00 %)
Seq Errs	0
Duration	64.82 s
Clock Drift	-7 ms
Freq Drift	7999 Hz (-0.01 %)

Reverse

SSRC	0x45de7f0e
Max Delta	62.05 ms @ 449
Max Jitter	18.13 ms
Mean Jitter	14.89 ms
Max Skew	31.02 ms
RTP Packets	3241
Expected	3241
Lost	0 (0.00 %)
Seq Errs	0
Duration	64.80 s
Clock Drift	-6 ms
Freq Drift	7999 Hz (-0.01 %)

2 streams found.

Forward

SSRC	0x44b99b50
Max Delta	100.18 ms @ 10353
Max Jitter	26.37 ms
Mean Jitter	16.10 ms
Max Skew	124.65 ms
RTP Packets	3299
Expected	3299
Lost	0 (0.00 %)
Seq Errs	0
Duration	65.86 s
Clock Drift	-25 ms
Freq Drift	7997 Hz (-0.04 %)

Reverse

SSRC	0x44b99b50
Max Delta	123.93 ms @ 10451
Max Jitter	21.63 ms
Mean Jitter	14.92 ms
Max Skew	-102.66 ms
RTP Packets	3294
Expected	3294
Lost	0 (0.00 %)
Seq Errs	0
Duration	65.84 s
Clock Drift	-4 ms
Freq Drift	7999 Hz (-0.01 %)

2 streams found.

Forward

SSRC	0x4dd9ad9e
Max Delta	99.79 ms @ 6861
Max Jitter	22.10 ms
Mean Jitter	16.65 ms
Max Skew	-75.51 ms
RTP Packets	3270
Expected	3270
Lost	0 (0.00 %)
Seq Errs	0
Duration	65.36 s
Clock Drift	-12 ms
Freq Drift	7999 Hz (-0.02 %)

Reverse

SSRC	0x4dd9ad9e
Max Delta	61.65 ms @ 1551
Max Jitter	17.90 ms
Mean Jitter	15.29 ms
Max Skew	42.86 ms
RTP Packets	3265
Expected	3265
Lost	0 (0.00 %)
Seq Errs	0
Duration	65.26 s
Clock Drift	-13 ms
Freq Drift	7998 Hz (-0.02 %)

2 streams found.

Figure 36: GSM results

B

PACKETIZATION RESULTS

Different framing sizes are tested for the G.277 codec.

Table 15: Packetization results

	Normalized		Raw		
ms	**Packets**	**KiloBytes**	**Packets**	**Bytes**	**Time**
10	12003,92	1571	12862	1723508	64,29
20	6001,15	1254	6285	1344990	62,84
30	4001,85	1149	4335	1274490	65,00
40	3002,47	1097	3221	1204654	64,37
50	2400,51	1064	2803	1272562	70,06
60	2000,87	1043	2081	1111254	62,40
70	1715,26	1028	1960	1203440	68,56
80	1501,47	1018	1633	1133302	65,26
90	1334,45	1009	1457	1127718	65,51
100	1201,28	1002	1275	1088850	63,68
110	1092,74	997	1140	1064760	62,60
120	1000,30	991	1112	1127568	66,70
130	923,58	987	1041	1138854	67,63
140	857,90	984	943	1107082	65,95
150	857,59	983	941	1104734	65,84

C

ASTERISK CONFIGURATION

The asterisk configuration is done in the sip.conf file. The settings used in the tests is listed below.

```
# /etc/asterisk/sip.conf
[general]
allowoverlap=no
udpbindaddr=0.0.0.0
transport=udp
srvlookup=yes

disallow=all
allow=g722:80

useragent=Asterisk-Test
nat = yes
directmedia=no

[host-phone]
type=friend
context=phones
host=dynamic
secret=1234
[1235]
type=friend
context=phones
host=dynamic
secret=1234
[1236]
type=friend
context=phones
host=dynamic
secret=1234
```

D

VPN CONFIGURATIONS

D.1 OPENVPN

The configurations for the OpenVPN server and client are listed below.

D.1.1 *Server*

```
# /etc/openvpn-nl/server.conf
# Port to run OpenVPN on
port 1194
# Protocol - UDP/TCP
proto udp
# TUN or TAP device
dev tun
# Certificate and encryption files
ca /etc/openvpn-nl/ca.crt
cert /etc/openvpn-nl/server.crt
key /etc/openvpn-nl/server.key
dh /etc/openvpn-nl/dh.pem
# Topology style
topology subnet
# Private subnet configuration
server 10.8.0.0 255.255.255.0
ifconfig-pool-persist ipp.txt
# Enable client-to-client communications
client-to-client
# Allow the same certificate to connect twice
duplicate-cn
keepalive 10 120
# Cipher selection
cipher AES-256-CBC
persist-key
persist-tun
status openvpn-status.log
log-append  openvpn.log
verb 3
```

D.1.2 *Client*

```
# client.ovpn
client

;dev tap
dev tun
```

```
;proto tcp
proto udp

remote 192.168.56.101 1194

resolv-retry infinite

nobind

# Try to preserve some state across restarts.
persist-key
persist-tun

ca ca.crt
cert client.crt
key client.key

;ns-cert-type server
remote-cert-tls server


cipher AES-256-CBC
auth SHA256

# Enable compression on the VPN link.
;comp-lzo

verb 3
```

D.2 IPSEC

The configurations for the IPSec server and client are listed below.

D.2.1 *Server*

IPSec Configuration

```
# /etc/ipsec.conf - Openswan IPsec configuration file
config setup
        plutoopts="--perpeerlog"

        nat_traversal=yes

        virtual_private=%v4:10.0.0.0/8,%v4:192.168.0.0/16,%v
            4:172.16.0.0/12,%v4:25.0.0.0/8,%v6:fd00::/8,%v6:fe
            80::/10
        oe=off
        protostack=auto
        plutostderrlog=/var/log/pluto.log
    keep_alive=600
```

```
conn L2TP-PSK-noNAT
    authby=secret
    # pre-shared secret.

        # Enabled for different tests
    #forceencaps=yes
    #compress=yes

    pfs=no

    auto=add

    keyingtries=3

    ikelifetime=8h
    keylife=1h

    ike=aes256-sha1
    phase2alg=aes256-sha1

    type=transport
    #because we use l2tp as tunnel protocol

    left=192.168.56.101

    leftprotoport=17/1701
    right=%any
    rightprotoport=17/%any

    dpddelay=10
    dpdtimeout=20
    dpdaction=clear
```

XL2TPD Configuration

```
# /etc/xl2tpd/xl2tpd.conf
[global]
ipsec saref = yes
saref refinfo = 30

[lns default]
ip range = 172.16.1.30-172.16.1.100
local ip = 172.16.1.1
;refuse pap = yes
require authentication = yes
;ppp debug = yes
pppoptfile = /etc/ppp/options.xl2tpd
length bit = yes

# /etc/ppp/options.xl2tpd
require-mschap-v2
refuse-mschap
ms-dns 8.8.8.8
ms-dns 8.8.4.4
```

```
asyncmap 0
auth
crtscts
idle 1800
mtu 1200
mru 1200
lock
hide-password
local
#debug
name l2tpd
proxyarp
lcp-echo-interval 30
lcp-echo-failure 4
```

D.2.2 *Client*

IPSec Configuration

```
# /etc/ipsec.conf
config setup
  nat_traversal=yes
  protostack=netkey
conn L2TP-PSK
  authby=secret
  auto=add
  forceencaps=yes
  keyingtries=1
  dpddelay=30
  dpdtimeout=120
  dpdaction=clear
  rekey=yes
  ikelifetime=8h
  keylife=1h
  type=transport
  right=192.168.56.102
  rightprotoport=17/1701
  left=192.168.56.101
  leftprotoport=17/1701
```

XL2TPD Configuration

```
# /etc/xl2tpd/xl2tpd.conf
[lac vmipsecvpn]
lns = 192.168.56.101
ppp debug = yes
pppoptfile = /etc/ppp/options.l2tpd.client
length bit = yes

# /etc/ppp/options.l2tpd.client
ipcp-accept-local
ipcp-accept-remote
refuse-eap
```

```
require-mschap-v2
noccp
noauth
idle 1800
mtu 1410
mru 1410
defaultroute
usepeerdns
debug
lock
connect-delay 5000
name alice
password AlicePass
```

BIBLIOGRAPHY

[1] BBC Business. *Apple asks court to reverse FBI iPhone order*. http://www.bbc.com/news/business-35664904/. Accessed: 01-3-2016. 2015.

[2] WhatsApp. *WhatsApp Encryption Overview*. https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf. Accessed: 20-4-2016. 2016.

[3] Grazia Cecere and Nicoletta Corrocher. "The usage of VoIP services and other communication services: An empirical analysis of Italian consumers." In: *Technological Forecasting and Social Change* 79.3 (2012), pp. 570–578.

[4] Yaniv Masjedi. *VoIP Usage Sees Continued Growth*. https://www.nextiva.com/voip/voip-usage-sees-continued-growth-over-2009-2010.html. Accessed: 14-1-2016. 2011.

[5] Toon De Pessemier, Isabelle Stevens, Lieven De Marez, Luc Martens, and Wout Joseph. "Quality assessment and usage behavior of a mobile voice-over-IP service." In: *Telecommunication Systems* (2015), pp. 1–16.

[6] J Regis Jr. *Securing VoIP: Keeping Your VoIP Network Safe*. Elsevier, 2014.

[7] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. *SIP: Session Initiation Protocol*. RFC 3261. 2002.

[8] M. Handley, V. Jacobson, and C. Perkins. *SDP: Session Description Protocol*. RFC 4566. 2006.

[9] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. *RTP: A Transport Protocol for Real-Time Applications*. RFC 3550. 2002.

[10] H. Schulzrinne and S. Casner. *RTP Profile for Audio and Video Conferences with Minimal Control*. RFC 3551. 2003.

[11] Sicker McGann and Douglas C Sicker. "An analysis of security threats and tools in SIP-based VoIP systems." In: *Second VoIP security workshop*. 2005.

[12] Mark Collier. "Basic vulnerability issues for SIP security." In: *SecureLogix Corporation* (2005).

[13] Christian Wieser, Marko Laakso, and Henning G Schulzrinne. "Security testing of SIP implementations." In: (2003).

[14] Humberto Abdelnur, Tigran Avanesov, Michael Rusinowitch, and Radu State. "Abusing SIP authentication." In: *Information Assurance and Security, 2008. ISIAS'08. Fourth International Conference on*. IEEE. 2008, pp. 237–242.

[15] M Zubair Rafique, M Ali Akbar, and Muddassar Farooq. "Evaluating DoS attacks against SIP-based VoIP systems." In: *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*. IEEE. 2009, pp. 1–6.

[16] Ruishan Zhang, Xinyuan Wang, Xiaohui Yang, and Xuxian Jiang. "Billing Attacks on SIP-Based VoIP Systems." In: *WOOT* 7 (2007), pp. 1–8.

[17] Philippe Biondi and Fabrice Desclaux. "Silver needle in the Skype." In: *Black Hat Europe* 6 (2006), pp. 25–47.

[18] Jason Jaskolka and Ridha Khedri. "Exploring covert channels." In: *System Sciences (HICSS), 2011 44th Hawaii International Conference on*. IEEE. 2011, pp. 1–10.

[19] Thomas Eisenbarth, Sandeep Kumar, Christof Paar, Axel Poschmann, and Leif Uhsadel. "A survey of lightweight-cryptography implementations." In: *IEEE Design & Test of Computers* 6 (2007), pp. 522–533.

[20] Diaa Salama Abdul Elminaam, Hatem Mohamed Abdul Kader, and Mohie Mohamed Hadhoud. "Performance evaluation of symmetric encryption algorithms." In: *IJCSNS International Journal of Computer Science and Network Security* 8.12 (2008), pp. 280–286.

[21] Shashank Khanvilkar and Ashfaq Khokhar. "Virtual private networks: an overview with performance evaluation." In: *Communications Magazine, IEEE* 42.10 (2004), pp. 146–154.

[22] S. Kent. *IP Authentication Header*. RFC 4302. 2005.

[23] S. Kent. *IP Encapsulating Security Payload (ESP)*. RFC 4303. 2005.

[24] D. McGrew and Hoffman P. *Cryptographic Algorithm Implementation Requirements and Usage Guidance*. RFC 7321. 2014.

[25] Michael E Kounavis, Xiaozhu Kang, Ken Grewal, Mathew Eszenyi, Shay Gueron, and David Durham. "Encrypting the internet." In: *ACM SIGCOMM Computer Communication Review* 41.4 (2011), pp. 135–146.

[26] L. Law and J. Solinas. *Suite B Cryptographic Suites for IPsec*. RFC 6379. 2011.

[27] Stephan Schulz, Vijay Varadharajan, and Ahmad-Reza Sadeghi. "The Silence of the LANs: Efficient Leakage Resilience for IPsec VPNs." In: *Information Forensics and Security, IEEE Transactions on* 9.2 (2014), pp. 221–232.

[28] OpenVPN. *OpenVPN - Open Source VPN*. https://openvpn.net/. Accessed: 28-1-2016. 2016.

[29] ARM mbed. *mbed TLS*. https://tls.mbed.org/. Accessed: 28-1-2016. 2015.

[30] Fox-IT. *OpenVPN-NL*. https://openvpn.fox-it.com/. Accessed: 18-4-2016. 2016.

[31] I Kotuliak, P Rybár, and P Truchly. "Performance comparison of IPsec and TLS based VPN technologies." In: *Emerging eLearning Technologies and Applications (ICETA), 2011 9th International Conference on*. IEEE. 2011, pp. 217–221.

[32] K. Hamzeh, G. Pall, W. Verthein, J. Taarud, W. Little, and G. Zorn. *Point-to-Point Tunneling Protocol (PPTP)*. RFC 2637. 1999.

[33] Bruce Schneier et al. "Cryptanalysis of Microsoft's point-to-point tunneling protocol (PPTP)." In: *Proceedings of the 5th ACM conference on Computer and communications security*. ACM. 1998, pp. 132–141.

[34] Bruce Schneier, David Wagner, et al. "Cryptanalysis of Microsoft's PPTP Authentication Extensions (MS-CHAPv2)." In: *Secure Networking—CQRE [Secure]'99*. Springer, 1999, pp. 192–203.

[35] Jason L. Outen. *VPN Security and Methodology*. http://www.infosecwriters.com/Papers/JOuten-VPNsecurity.pdf. Accessed: 2-4-2016. 2014.

[36] André Zúquete and Carlos Frade. "Fast VPN mobility across Wi-Fi hotspots." In: *Security and Communication Networks (IWSCN), 2010 2nd International Workshop on*. IEEE. 2010, pp. 1–7.

[37] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina. *Generic Routing Encapsulation (GRE)*. RFC 2784. 2002.

[38] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman. *The Secure Real-time Transport Protocol (SRTP)*. RFC 3711. 2004.

[39] IDC. *Smartphone OS Market Share, 2015 Q2*. http://www.idc.com/prodserv/smartphone-os-market-share.jsp. Accessed: 12-1-2016. 2015.

[40] GSM Arena. *GSM Arena Phone specs*. http://www.gsmarena.com/. Accessed: 1-3-2016. 2016.

[41] Notebookcheck. *Laptop Video Graphics Cards - Benchmark List*. http://www.notebookcheck.net/Mobile-Graphics-Cards-Benchmark-List.844.0.html. Accessed: 14-3-2016. 2016.

[42] Aaron Carroll and Gernot Heiser. "An Analysis of Power Consumption in a Smartphone." In: *USENIX annual technical conference*. Vol. 14. Boston, MA. 2010.

[43] Luca Ardito, Giuseppe Procaccianti, Marco Torchiano, and Giuseppe Migliore. "Profiling power consumption on mobile devices." In: (2013).

[44] ARM. *Cortex-A Series*. https://www.arm.com/products/processors/cortex-a/index.php. Accessed: 15-3-2016. 2016.

[45] Abdullah Azfar, Kim-Kwang Raymond Choo, and Lin Liu. "A study of ten popular Android mobile VoIP applications: Are the communications encrypted?" In: *System Sciences (HICSS), 2014 47th Hawaii International Conference on*. IEEE. 2014, pp. 4858–4867.

[46] D Richard Kuhn, Thomas J Walsh, and Steffen Fries. "Security considerations for voice over IP systems." In: *NIST special publication* (2005), pp. 800–58.

[47] The Government of the Hong Kong Special Administrative Region. *Voice over IP Security*. http://www.infosec.gov.hk/english/technical/files/voice.pdf. Accessed: 1-4-2016. 2008.

[48] Homeland Security. *Voice over Internet Protocol*. https://www.dhs.gov/sites/default/files/publications/4300AHandbookAttachmentQ5-VoiceoverIP.pdf. Accessed: 3-4-2016. 2014.

[49] Takehiro Takahashi and Wenke Lee. "An assessment of VoIP covert channel threats." In: *Security and Privacy in Communications Networks and the Workshops, 2007. SecureComm 2007. Third International Conference on*. IEEE. 2007, pp. 371–380.

[50] Wojciech Mazurczyk and Krzysztof Szczypiorski. "Covert Channels in SIP for VoIP signalling." In: *Global E-Security*. Springer, 2008, pp. 65–72.

[51] Wojciech Mazurczyk and Krzysztof Szczypiorski. "Steganography of VoIP streams." In: *On the Move to Meaningful Internet Systems: OTM 2008*. Springer, 2008, pp. 1001–1018.

[52] Soumyendu Das, Subhendu Das, Bijoy Bandyopadhyay, and Sugata Sanyal. "Steganography and Steganalysis: different approaches." In: *arXiv preprint arXiv:1111.3758* (2011).

[53] YF Huang, Song Tang, and Ye Zhang. "Detection of covert voice-over Internet protocol communications using sliding window-based steganalysis." In: *Communications, IET* 5.7 (2011), pp. 929–936.

[54] Yi-Pai Huang, Song Tang, Chenlei Bao, and Yau Jim Yip. "Steganalysis of compressed speech to detect covert voice over Internet protocol channels." In: *Information Security, IET* 5.1 (2011), pp. 26–32.

[55] Charles V Wright, Lucas Ballard, Scott E Coull, Fabian Monrose, and Gerald M Masson. "Spot me if you can: Uncovering spoken phrases in encrypted VoIP conversations." In: *2008 IEEE Symposium on Security and Privacy (sp 2008)*. IEEE. 2008, pp. 35–49.

[56] Asterisk Wiki. *RTP Packetization*. https://wiki.asterisk.org/wiki/display/AST/RTP+Packetization. Accessed: 20-5-2016. 2016.

[57] Google. *How Hangouts encrypts information*. https://support.google.com/hangouts/answer/6046115?hl=en. Accessed: 01-8-2016. 2016.

[58] Skype. *Skype Encryption*. https://support.skype.com/en/faq/fa31/does-skype-use-encryption. Accessed: 01-8-2016. 2016.

[59] Ars Technica. *Think your Skype messages get end-to-end encryption? Think again*. http://arstechnica.com/security/2013/05/think-your-skype-messages-get-end-to-end-encryption-think-again/. Accessed: 01-8-2016. 2013.

[60] MFXJ Oberhumer. "LZO real-time data compression library." In: *User manual for LZO version 0.28, URL: http://www. infosys. tuwien. ac. at/Staff/lux/marco/lzo. html (February 1997)* (2005).