



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

**SEMI-PASSIVE TRANSPORT LAYER SECURITY
VERSION 1.3 DECRYPTION OPPORTUNITIES**

by

Alex M. Anderson

September 2023

Thesis Advisor:

Co-Advisor:

John C. McEachen

Murali Tummala

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE September 2023	3. REPORT TYPE AND DATES COVERED Master's thesis		
4. TITLE AND SUBTITLE SEMI-PASSIVE TRANSPORT LAYER SECURITY VERSION 1.3 DECRYPTION OPPORTUNITIES			5. FUNDING NUMBERS	
6. AUTHOR(S) Alex M. Anderson				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Laboratory for Telecommunication Sciences, College Park, MD 20740			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.			12b. DISTRIBUTION CODE A	
13. ABSTRACT (maximum 200 words) With the release of Transport Layer Security (TLS) version 1.3 protocol in 2018, many security improvements have proven challenging to overcome from both an offensive and defensive cybersecurity perspective. The adoption rate of this latest version is continuing to increase. TLS version 1.3 is now the preferred protocol for use in the vast majority of active websites. However, a security cryptosystem is only secure if the private keys are kept secret. In this research, methods of key logging will be explored, along with various ways of extraction to conserve network bandwidth or to provide a stealthy exfiltration. With the cryptographic keys now available, the end result is a successful decryption of passively captured network traffic. Key logging of distinct Gmail email sessions and multiple compression techniques were experimented and analyzed. The outcome has provided proven viable benefits for both the offensive and defensive cybersecurity analyst teams with follow-on development opportunities.				
14. SUBJECT TERMS TLS 1.3, transport layer security version 1.3, encryption, decryption, man-on-the-side attack, blue team, red team, defensive cyber, offensive cyber			15. NUMBER OF PAGES 73	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

**SEMI-PASSIVE TRANSPORT LAYER SECURITY VERSION 1.3
DECRYPTION OPPORTUNITIES**

Alex M. Anderson
Lieutenant Commander, United States Navy
BS, Purdue University, 2012

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

**NAVAL POSTGRADUATE SCHOOL
September 2023**

Approved by: John C. McEachen
Advisor

Murali Tummala
Co-Advisor

Douglas J. Fouts
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

With the release of Transport Layer Security (TLS) version 1.3 protocol in 2018, many security improvements have proven challenging to overcome from both an offensive and defensive cybersecurity perspective. The adoption rate of this latest version is continuing to increase. TLS version 1.3 is now the preferred protocol for use in the vast majority of active websites. However, a security cryptosystem is only secure if the private keys are kept secret. In this research, methods of key logging will be explored, along with various ways of extraction to conserve network bandwidth or to provide a stealthy exfiltration. With the cryptographic keys now available, the end result is a successful decryption of passively captured network traffic. Key logging of distinct Gmail email sessions and multiple compression techniques were experimented and analyzed. The outcome has provided proven viable benefits for both the offensive and defensive cybersecurity analyst teams with follow-on development opportunities.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	RED TEAM AND BLUE TEAM.....	1
B.	THESIS OBJECTIVE	1
C.	RELATED WORK	2
D.	THESIS OVERVIEW	3
II.	BACKGROUND	5
A.	WHAT IS TLS, SSL, AND HTTPS?.....	5
B.	TLS HANDSHAKE PROCESS.....	6
C.	TLS VERSION 1.2 PACKET INSPECTION	12
D.	DIFFERENCES BETWEEN TLS V1.2 AND V1.3	12
E.	SSLKEYLOGFILE.....	14
III.	METHODOLOGY	17
A.	SYSTEM SPECIFICATIONS	17
B.	METHOD FOR MAN-ON-THE-SIDE (MOTS) COLLECTION AND DECRYPTION	18
C.	HOW TO DECRYPT MAN-ON-THE-SIDE TRAFFIC	19
D.	CLIENT-SIDE KEY LOGGING	21
E.	CREATING A WEB SERVER.....	22
F.	SERVER-SIDE KEY LOGGING	24
IV.	EXPERIMENT RESULTS AND ANALYSIS	27
A.	CLIENT SSLKEYLOGFILE	27
B.	SERVER SSLKEYLOGFILE	28
C.	DECRYPTION.....	30
D.	COMPRESSION TECHNIQUES	33
E.	GMAIL EMAIL SESSIONS.....	38
V.	CONCLUSIONS	43
A.	CONTRIBUTIONS.....	43
B.	FUTURE WORK.....	44
	APPENDIX A. QUALYS SSL LABS REPORT	45

APPENDIX B. MATLAB CODE AND OUTPUT	49
LIST OF REFERENCES	53
INITIAL DISTRIBUTION LIST	55

LIST OF FIGURES

Figure 1.	TLS 1.2 Handshake Process. Source: [14].....	6
Figure 2.	TLS 1.2 Client Hello Message.....	7
Figure 3.	Server Hello Message Part 1.....	8
Figure 4.	Server Hello Message Part 2.....	9
Figure 5.	Client Key Exchange	10
Figure 6.	Cryptographic Key Generation: (a) Pre-Master Secret and (b) Master Secret. Derived from RFC 5246 [14].....	11
Figure 7.	TLS Handshake Complete	12
Figure 8.	TLS 1.2 Wireshark Packet Capture	13
Figure 9.	TLS 1.3 Wireshark Packet Capture	14
Figure 10.	TLS 1.3 Key Derivation Process. Derived from RFC 8446 [1].....	16
Figure 11.	MotS Collection and Decryption Workflow.....	19
Figure 12.	Decryption of a Packet Capture	21
Figure 13.	Windows Environment Variables	22
Figure 14.	options-ssl-apache.conf Configuration File.....	24
Figure 15.	Apache2 Service Override Configuration File Modifications	25
Figure 16.	Webserver Access.log Sample, Showing Remote IP Address, Date/Time of Access, HTTP Method, Response Codes, and User Agent String.....	30
Figure 17.	Wireshark TLS Plugin Configuration via Edit → Preferences from Home Screen.....	31
Figure 18.	Encrypted Packet Capture Showing Encrypted Packets after Handshake Agreement.....	32
Figure 19.	Decrypted Packet Capture Showing Plaintext Communications Highlighted in Green.....	33

Figure 20.	7zip Compression Results versus Number of Sessions (Y-Axis Log Scaled).....	35
Figure 21.	7zip Compression versus Traffic Packet Capture Size Ratio	37
Figure 22.	Packet Capture, Key Log, and 7zip Compressed File Size Scaling with Number of Sessions	38
Figure 23.	Key Quadruples Collected in Gmail Sessions Indicating Maximum, Average and Minimum for Each Measure of Time Duration.....	41

LIST OF TABLES

Table 1.	History of SSL and TLS Protocols. Adapted from [1], [10]–[13].	6
Table 2.	Client SSLKEYLOGFILE Showing its Captured TLS Encryption/ Decryption Keys	27
Table 3.	Server SSLKEYLOGFILE Showing its Captured TLS Encryption/ Decryption Keys	29
Table 4.	Gzip and 7zip Compression Results for Various Key File Sizes.	34
Table 5.	Compressed Key Files versus Packet Capture Size Ratio	36
Table 6.	Captured Gmail Email Session Keys	39
Table 7.	Gmail Email Session Keys Captured over Time	40

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
AMD	Advanced Micro Devices, Incorporated
AWS	Amazon Web Services
CBC	Cipher Block Chaining
CPU	Central Processing Unit
DDR4	Double Data Rate Fourth-Generation
DES	Data Encryption Standard
DNS	Domain Name Service
DOD	Department of Defense
EC2	Elastic Compute Cloud
ECDHE	Elliptic-Curve Diffie–Hellman Ephemeral
FOSS	Free Open-Source Software
GCM	Galois/Counter Mode
HTTPS	Hypertext Transfer Protocol Secure
IDS	Intrusion Detection Systems
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPS	Intrusion Protection Systems
MotS	Man-on-the-Side
NVMe	Nonvolatile Memory Express
Pcap	Packet Capture
PFS	Perfect Forward Secrecy

PKCS	Public-Key Cryptography Standards
RAM	Random Access Memory
RSA	Rivest-Shamir-Adleman, a public-key cryptosystem
SHA	Secure Hash Algorithm
SOC	Security Operations Center
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
URL	Uniform Resource Locator

ACKNOWLEDGMENTS

I would like to first thank my wife, Dr. Vanessa Cabe, for her support and encouragement throughout this enduring and arduous venture. Secondly, I would like to thank my advisors, Professors John McEachen and Murali Tummala. Their insight and guidance for over two years has challenged me to grow academically and professionally to feats I would not have imagined just years ago. Lastly, Professor Britta Hale was invaluable in helping me fully understand the TLS version 1.3 protocol.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

TLS (Transport Layer Security) is an internet protocol responsible for securing an internet connection [1]. TLS is also commonly referred to as Hypertext Transfer Protocol Secure (HTTPS) or TLS over HTTP. These are the protocols used in every web browser or web application that interacts with a web server on the worldwide web. In August 2018, the latest and current version of the TLS protocol, version 1.3, was released by the Internet Engineering Task Force (IETF) [1]. This new revision built upon the previous TLS version (1.2) and was updated to prevent eavesdropping, tampering, and message forgery [1]. With at least 63% of the top one million websites preferring TLS 1.3 as of 2021 [2], its prevalence will likely continue to increase in the future.

A. RED TEAM AND BLUE TEAM

The methods described in this thesis would be useful in both offensive and defensive cyber operations, commonly referred to as the “red team” and the “blue team,” respectively [3]. More specifically, these teaming efforts consist of job titles such as penetration tester, vulnerability analyst, and ethical hacker for red team and cybersecurity analyst, specialist, or engineer for blue team [3]. These teams are employed or deployed to organizations around the world to enhance the cybersecurity posture of an organization and ultimately reduce their risk of a data breach.

B. THESIS OBJECTIVE

The objective of this thesis is to explore methods of eavesdropping upon the TLS 1.3 protocol traffic from both a red team and blue team perspective. The TLS 1.2 protocol predominantly used the RSA (Rivest-Shamir-Adleman) encryption algorithm, which employed a static public and private key system. From the perspective of a server, the public key would be universally available and used by clients to decrypt content sent by the server and for clients to encrypt content being sent to the server. The private key would be kept confidential and be used to decrypt content being received by the server. If the private key of the server was extracted by a red team actor, the red team could then

masquerade as the server and impersonate nearly all of its actions, including decrypting any traffic intercepted off-site at a later date.

While servers using TLS 1.3 will still employ RSA ciphers for aspects, such as authentication, the private keys are no longer useful in decrypting its traffic to and from clients since Perfect Forward Secrecy (PFS) ciphers are now required. PFS ensures that there is no link between the server private key and each session key used for encryption, such as the Elliptic-Curve Diffie–Hellman Ephemeral (ECDHE) [4]. The ciphers that employ PFS help the Department of Defense (DOD) networks and other organizations who use them achieve the highest level of internet security available today.

C. RELATED WORK

In [5], Davies’ work in deconstructing and comparing the new TLS 1.3 handshake was helpful in understanding many of the security enhancements with the updated protocol. Previously, it was much easier to examine the TLS 1.2 handshake with packet capture tools, such as tcpdump or Wireshark, but now in TLS 1.3 everything after the server hello message is encrypted, including the certificate exchange [5]. The `Openssl s_client` command with the `-msg` option or full decryption in Wireshark with the TLS keys is required just to observe the protocol handshake (the cipher negotiation phase) before any of the intended communication begins [5].

While the adoption of TLS 1.3 is continuing to increase, many websites encounter issues of stability in supporting the 1.3 version [6]. This instability happens when downgrading the TLS version or when migrating to servers with older TLS versions [6]. The general downgrade attack simply reverts the TLS version from 1.3 to 1.2 or older, losing all the latest security benefits gained from TLS 1.3. This is an excellent example of what a red team actor would try to accomplish to attack the new protocol. These issues can be mitigated by strict web server administration and testing to ensure the proper and stable implementation of TLS 1.3 across all regions and associated platforms in use by the web service [6]. While the work proposed in this thesis does not focus on the TLS 1.3 downgrading issue, it will spotlight alternatives a red team can pursue to take advantage of the protocol.

While being the latest of many iterations, the TLS 1.3 protocol is certainly not perfect. The handshake agreement and unencrypted parameters are known to still leak significant information leading to the identification of the operating system [7]. This information leak along with other visible identifiers can in turn, for example, relate the fact that a specific mobile device with an Apple operating system is communicating with a specific mail server with the Linux operating system. The possibilities for further metadata extraction, analysis, and intelligence inferences are still quite prevalent. This metadata is useful, but if the rest of the encrypted communication could be analyzed, both the blue and red teams could benefit toward their final objectives.

Methods around the TLS 1.3 security improvements have been proposed, such as TLS session splitting [8]. This is similar to a man-in-the-middle attack where a middlebox is used to terminate the TLS 1.3 connection and re-establish the connection with the desired endpoint using a less secure protocol. This thesis examines methods that do not require terminating the connection prior to the server, thus making it less obvious the connection is being monitored. Another method discussed, specifically for clients, is the logging of TLS secrets via the SSLKEYLOGFILE [8]. The feature is restricted and limited to a few applications but will be further discussed in this research.

D. THESIS OVERVIEW

The rest of this thesis is structured as follows: Chapter II discusses the background of TLS, its handshake, cipher suites, and methods of decryption. Chapter III describes the test system specifications and procurement, technical details of how cryptographic key logging was performed, and debugging. Chapter IV presents experiment results analyzing various email sessions, the cryptographic keys associated with them, and methods of compression for lightweight storage or future exfiltration. Chapter V finishes with the conclusions and takeaways ascertained by the experiments. Appendix A includes an assessment of the experimental web server, and Appendix B is the MATLAB code and output used in the experiment analysis.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

In Chapter I, a brief introduction to TLS and the red and blue team perspectives was presented. In this chapter, the technical details of TLS, specifically version 1.3, will be discussed more thoroughly.

A. WHAT IS TLS, SSL, AND HTTPS?

TLS stems from its predecessor, Secure Sockets Layer (SSL), and is effectively a protocol or standard technology that keeps an internet connection secure [9]. This is accomplished by employing encryption algorithms to safeguard sensitive data that is being transferred between two systems, thereby preventing eavesdroppers from reading or tampering with the data being transferred [9].

SSL was developed and first publicly released as SSL v2 in 1995 [10]. It was then succeeded by SSL 3.0 and TLS 1.0 in 1999, with the changing of name to help indicate that SSL and TLS are not interoperable [10]. The next major update came in 2006 with TLS 1.1 and soon thereafter TLS 1.2 in 2008 [10]. Due to serious security considerations SSL v2 was prohibited and deprecated from use in 2011 by RFC 6176 [11]. Shortly thereafter, SSL 3.0 flaws led to its deprecation in 2015 in RFC 5246 [12]. Finally, TLS 1.0 and TLS 1.1 were deprecated in 2021 with RFC 8996 leaving only TLS 1.2 and TLS 1.3 in a currently supported status [13].

Hyper Text Transfer Protocol Secure (HTTPS) is simply what appears in the Uniform Resource Locator (URL) block of a website when it is secured by a TLS connection [9]. Further details of the public key certificate used to generate the session secret may be viewed by clicking on the lock icon next to the URL and exploring the security and information options. Table 1 summarizes the history of the SSL and TLS protocols.

Table 1. History of SSL and TLS Protocols. Adapted from [1], [10]–[13].

Protocol	Version	Released	Deprecated
SSL	2.0	1995	2011
SSL	3.0	1999	2015
TLS	1.0	1999	2021
TLS	1.1	2006	2021
TLS	1.2	2008	N/A
TLS	1.3	2018	N/A

B. TLS HANDSHAKE PROCESS

For a more thorough breakdown of the TLS handshake process, TLS 1.2 will be discussed in its four major steps: 1) Client Hello 2) Server Hello 3) Client key exchange/master secret computation 4) Complete [14]. This overall flow is depicted in Figure 1.

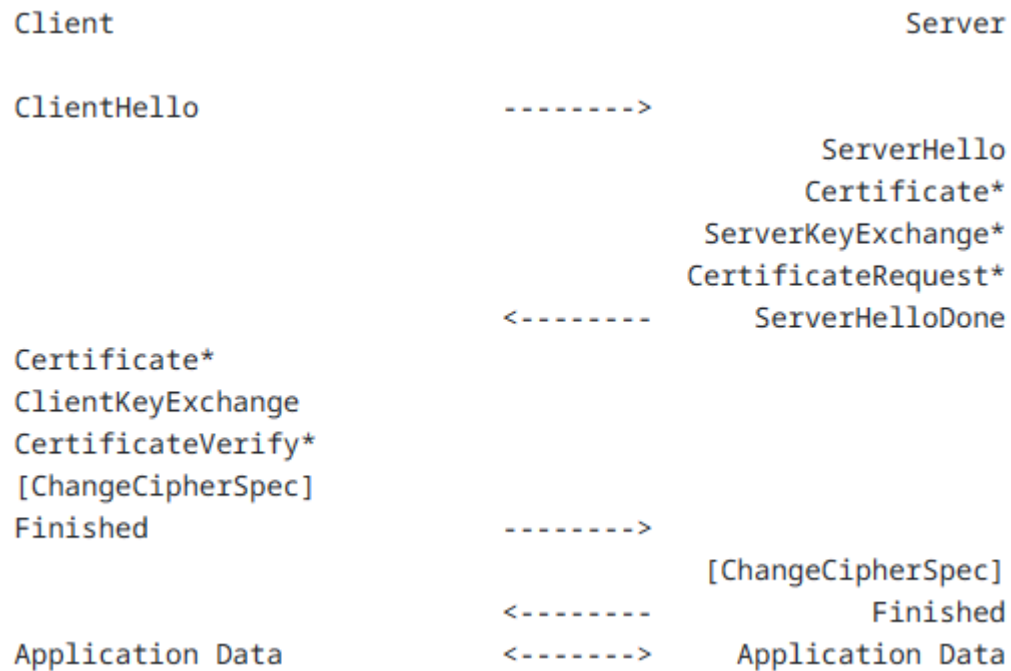


Figure 1. TLS 1.2 Handshake Process. Source: [14]

(1) Client Hello

The first step of the TLS 1.2 handshake begins with the Client Hello message, created by the client and sent to the web server, as shown in Figure 2.

```
> Internet Protocol Version 4, Src: 185.216.74.12, Dst: 172.31.22.121
> Transmission Control Protocol, Src Port: 4454, Dst Port: 443, Seq: 1, Ack: 1, Len: 517
▼ Transport Layer Security
  ▼ TLSv1.2 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 512
  ▼ Handshake Protocol: Client Hello
    Handshake Type: Client Hello (1)
    Length: 508
    Version: TLS 1.2 (0x0303)
    > Random: 0311dfea7eab8b035affee36659b388d33206634facac9a247a71e6b9af84a8a
      Session ID Length: 32
      Session ID: 5269a3e82468d4d4acb726ed378496d487b5d9edb502a91ccdd7e7ea7dfbc6fb
      Cipher Suites Length: 32
    ▼ Cipher Suites (16 suites)
      Cipher Suite: Reserved (GREASE) (0xaeaa)
      Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
      Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
```

Figure 2. TLS 1.2 Client Hello Message

This message contains valuable information in the negotiation process, most notable for the negotiation process are the TLS version TLS 1.2 (0x0303), the 28-byte random number Random: 0311dfea7eab8b035affee36659b388d33206634facac9a247a71e6b9af84a8a (Client Random) and the list of supported cipher suites. This Client Random will be part of the seed in the generation of the master secret to be introduced in future sections.

(2) Server Hello

The second step of the handshake is the Server Hello message, generated by the server and sent to the client, as shown in Figure 3.

```

> Internet Protocol Version 4, Src: 172.31.22.121, Dst: 185.216.74.12
> Transmission Control Protocol, Src Port: 443, Dst Port: 4454, Seq: 1, Ack: 518, Len: 2896
▼ Transport Layer Security
  ▼ TLSv1.2 Record Layer: Handshake Protocol: Server Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 112
  ▼ Handshake Protocol: Server Hello
    Handshake Type: Server Hello (2)
    Length: 108
    Version: TLS 1.2 (0x0303)
  > Random: e559bf059d533fe2d0474b01e6ae468f6dc3f9f03d0ff32cad8a6eaf098fc1
    Session ID Length: 32
    Session ID: 318f250e66aea0ceb3dc2880b1164396bbec4cb4d4bc351db99b926495230abc
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
    Compression Method: null (0)
    Extensions Length: 36
  > Extension: renegotiation_info (len=1)
  > Extension: server_name (len=0)
  > Extension: ec_point_formats (len=4)
  > Extension: application_layer_protocol_negotiation (len=11)

```

Figure 3. Server Hello Message Part 1

The server receives the Client Hello and determines if it supports the suggested TLS version (1.2 in this case) and then reciprocates with that supported version or suggests a new one. The Client Random is saved and a Server Random is generated for use in the next step: Random: e559bf059d533fe2d0474b01e6ae468f6dc3f9f03d0ff32cad8a6eaf098fc1. This Server Random is an additional part of the seed that will help generate the master secret. The server then picks a preferred cipher suite from the supported list supported by the client, Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f). The Server Hello step has an additional packet, shown in Figure 4.

```

> Transmission Control Protocol, Src Port: 443, Dst Port: 4454, Seq: 4097, Ack: 518, Len: 351
> [3 Reassembled TCP Segments (4016 bytes): #6(2779), #7(1200), #8(37)]
▼ Transport Layer Security
  ▼ TLSv1.2 Record Layer: Handshake Protocol: Certificate
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 4011
    ▼ Handshake Protocol: Certificate
      Handshake Type: Certificate (11)
      Length: 4007
      Certificates Length: 4004
      > Certificates (4004 bytes)
  ▼ Transport Layer Security
    ▼ TLSv1.2 Record Layer: Handshake Protocol: Server Key Exchange
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 300
      ▼ Handshake Protocol: Server Key Exchange
        Handshake Type: Server Key Exchange (12)
        Length: 296
        ▼ EC Diffie-Hellman Server Params
          Curve Type: named_curve (0x03)
          Named Curve: x25519 (0x001d)
          Pubkey Length: 32
          Pubkey: 165f20fe7a9fbad2980b39c1154c2dc2c33ed8e2e2ab183ebf15382cd6cebb15
          > Signature Algorithm: rsa_pss_rsae_sha256 (0x0804)
            Signature Length: 256
            Signature: 61738d522479cfb2a4789cbcf01166ac381cf113f71c094b00c36a051e13f4b65014e470...
        ▼ TLSv1.2 Record Layer: Handshake Protocol: Server Hello Done
          Content Type: Handshake (22)
          Version: TLS 1.2 (0x0303)
          Length: 4
          ▼ Handshake Protocol: Server Hello Done
            Handshake Type: Server Hello Done (14)
            Length: 0

```

Figure 4. Server Hello Message Part 2

The second packet from the server to the client includes the server certificate for the client to verify, the required parameters of the selected key exchange algorithm under EC Diffie-Hellman Server Params, and the server signature. This step finally concludes with Handshake Type: Server Hello Done (14).

(3) Client key exchange/master secret computation

The third step is largely performed by both the client and the server. First, the client browser must verify the server identity via the certificate sent in step 2, part 2 as well as the signature in the key exchange parameters. Assuming that all is valid, the browser will create a public key and add it to the Client Key Exchange / Client Params as shown in Figure 5.

```

> Internet Protocol Version 4, Src: 185.216.74.12, Dst: 172.31.22.121
> Transmission Control Protocol, Src Port: 4454, Dst Port: 443, Seq: 518, Ack: 4448, Len: 786
✓ Transport Layer Security
  ✓ TLSv1.2 Record Layer: Handshake Protocol: Client Key Exchange
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 37
  ✓ Handshake Protocol: Client Key Exchange
    Handshake Type: Client Key Exchange (16)
    Length: 33
  ✓ EC Diffie-Hellman Client Params
    Pubkey Length: 32
    Pubkey: e3ad276251799e661fcc238b519ee871c11af4e3a5afb3980e5f983543243d2a
  ✓ TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
    Content Type: Change Cipher Spec (20)
    Version: TLS 1.2 (0x0303)
    Length: 1
    Change Cipher Spec Message
  ✓ TLSv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 40
    Handshake Protocol: Encrypted Handshake Message
  ✓ TLSv1.2 Record Layer: Application Data Protocol: Hypertext Transfer Protocol
    Content Type: Application Data (23)
    Version: TLS 1.2 (0x0303)
    Length: 688
    Encrypted Application Data: 00000000000000001516a78c281f26bceef18f6f42869542bb4f316f018
    [Application Data Protocol: Hypertext Transfer Protocol]

```

Figure 5. Client Key Exchange

Now both the client and the server have all the components needed to create the master secret key which is used by both sides to generate more keys, such as the client session key and the server session key. This process is depicted in Figure 6.

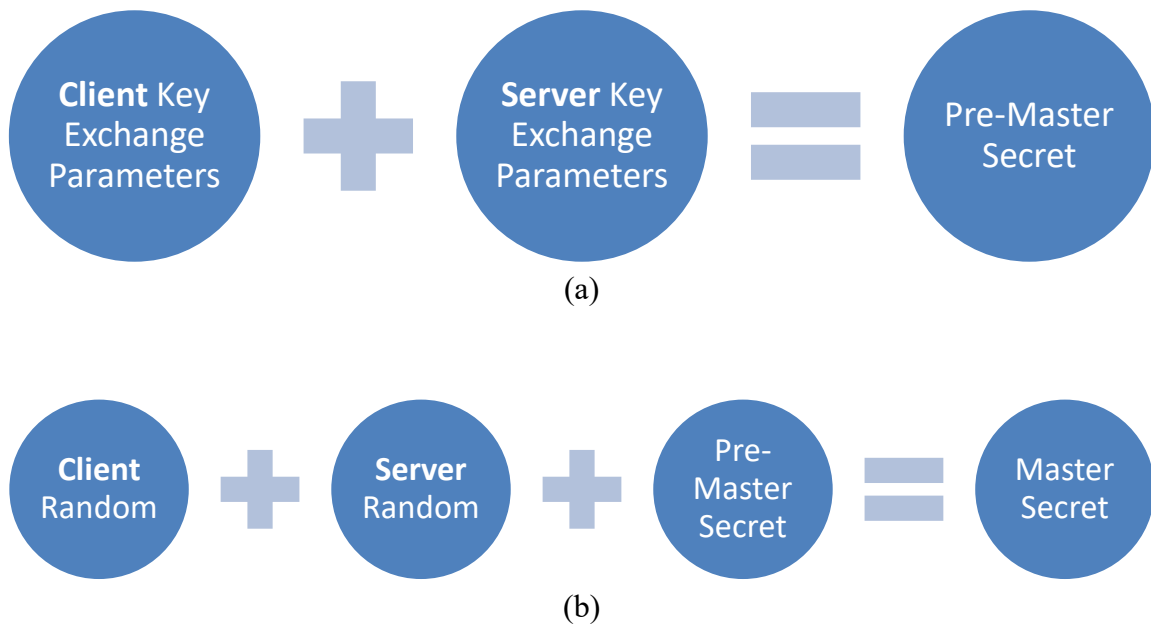


Figure 6. Cryptographic Key Generation: (a) Pre-Master Secret and (b) Master Secret. Derived from RFC 5246 [14].

Once the client successfully computes the master secret, it will then include the Change Cipher Spec and Encrypted Handshake Message to indicate that all future communications sent from the client will be encrypted.

(4) Complete

The last step is similar to the ending of step 3: the server verifies parameters sent by the client and whether encryption is expected next. The server concludes the TLS 1.2 handshake with the Change Cipher Spec and Encrypted Handshake Message as shown in Figure 7.

```

> Internet Protocol Version 4, Src: 172.31.22.121, Dst: 185.216.74.12
> Transmission Control Protocol, Src Port: 443, Dst Port: 4454, Seq: 4448, Ack: 1304, Len: 51
▼ Transport Layer Security
  ▼ TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
    Content Type: Change Cipher Spec (20)
    Version: TLS 1.2 (0x0303)
    Length: 1
    Change Cipher Spec Message
  ▼ TLSv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 40
    Handshake Protocol: Encrypted Handshake Message

```

Figure 7. TLS Handshake Complete

C. TLS VERSION 1.2 PACKET INSPECTION

TLS 1.2 was released in August 2008 by the Network Working Group and made many improvements over TLS 1.1 [14]. One of the various improvements was deprecating weak cipher suites, such as the Data Encryption Standard (DES) and implementing new strong cipher suites like TLS_RSA_WITH_AES_128_CBC_SHA. The ciphers employing the RSA algorithm have a feature called TLS inspection that allows for Intrusion Protection Systems (IPS) and Intrusion Detection Systems (IDS) to decrypt the traffic for inspection and analysis when given the private RSA key of the server [15].

The TLS 1.2 packet inspection feature also poses significant security risks [1]. If an adversary acquires the server private RSA key, then they too will be able to decrypt all of the traffic sent to and received by the server. This attack is quite stealthy in that an attacker just needs to capture traffic passively somewhere between the server and endpoint after obtaining said private key. The updated TLS 1.3 protocol has removed the RSA key exchange cipher methods from its approved list in order to remove this method of eavesdropping [1].

D. DIFFERENCES BETWEEN TLS V1.2 AND V1.3

There is a vast array of changes between TLS version 1.2 and version 1.3, but the ones most directly related to and affecting this research will be discussed in the following.

(1) Supported Encryption Algorithms

The variety of supported symmetric and asymmetric encryption algorithms has been drastically reduced with version 1.3. The pruned algorithms have been proven weak or obsolescent. Additionally, the encryption algorithms that remain are all of variety Authenticated Encryption with Associated Data (AEAD) algorithms [1].

(2) Key Exchange Mechanisms

Previously used static RSA and Diffie-Hellman cipher suites have been removed in version 1.3. All public-key based key exchange mechanisms will provide PFS. Perfect Forward Secrecy (PFS) ensures that when a private key is compromised, only the data of that session will be affected. No past or future sessions will use the same cryptographic keys [1]. PFS directly removes the TLS inspection feature which posed both positive and negative effects in cybersecurity.

(3) Handshake

The handshake process now takes fewer steps to complete. Additionally, all messages after the Server Hello are now encrypted in version 1.3 [1].

Figures 8 and 9 show the comparison between the TLS 1.2 and TLS 1.3 handshake process, respectively, in a Wireshark packet capture.

No.	Source	Destination	Protocol	Info
1	185.216.74.12	172.31.22.121	TCP	4454 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM TSval=1828875621 TSecr=0 WS=
2	172.31.22.121	185.216.74.12	TCP	443 → 4454 [SYN, ACK] Seq=0 Ack=1 Win=62643 Len=0 MSS=8961 SACK_PERM TSval=1219670082
3	185.216.74.12	172.31.22.121	TCP	4454 → 443 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1828875629 TSecr=1219670082
4	185.216.74.12	172.31.22.121	TLSv1.2	Client Hello
5	172.31.22.121	185.216.74.12	TCP	443 → 4454 [ACK] Seq=1 Ack=518 Win=62208 Len=0 TSval=1219670090 TSecr=1828875629
6	172.31.22.121	185.216.74.12	TLSv1.2	Server Hello
7	172.31.22.121	185.216.74.12	TCP	443 → 4454 [PSH, ACK] Seq=2897 Ack=518 Win=62208 Len=1200 TSval=1219670091 TSecr=18288
8	172.31.22.121	185.216.74.12	TLSv1.2	Certificate, Server Key Exchange, Server Hello Done
9	185.216.74.12	172.31.22.121	TCP	4454 → 443 [ACK] Seq=518 Ack=2897 Win=71680 Len=0 TSval=1828875638 TSecr=1219670091
10	185.216.74.12	172.31.22.121	TCP	4454 → 443 [ACK] Seq=518 Ack=4097 Win=74240 Len=0 TSval=1828875638 TSecr=1219670091
11	185.216.74.12	172.31.22.121	TCP	4454 → 443 [ACK] Seq=518 Ack=4448 Win=77312 Len=0 TSval=1828875639 TSecr=1219670092
12	185.216.74.12	172.31.22.121	TLSv1.2	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message, Application Data
13	172.31.22.121	185.216.74.12	TCP	443 → 4454 [ACK] Seq=4448 Ack=1304 Win=61440 Len=0 TSval=1219670124 TSecr=1828875663
14	172.31.22.121	185.216.74.12	TLSv1.2	Change Cipher Spec, Encrypted Handshake Message
15	172.31.22.121	185.216.74.12	TLSv1.2	Application Data
16	172.31.22.121	185.216.74.12	TLSv1.2	Application Data, Application Data
17	185.216.74.12	172.31.22.121	TCP	4454 → 443 [ACK] Seq=1304 Ack=4499 Win=77312 Len=0 TSval=1828875672 TSecr=1219670125
18	185.216.74.12	172.31.22.121	TCP	4454 → 443 [ACK] Seq=1304 Ack=7395 Win=82944 Len=0 TSval=1828875673 TSecr=1219670125
19	185.216.74.12	172.31.22.121	TCP	4454 → 443 [ACK] Seq=1304 Ack=8065 Win=86016 Len=0 TSval=1828875673 TSecr=1219670125
20	185.216.74.12	172.31.22.121	TLSv1.2	Application Data

Figure 8. TLS 1.2 Wireshark Packet Capture

No.	Time	Source	Destination	Protocol	Info
1	0.000000	185.216.74.12	172.31.22.121	TCP	58146 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK
2	0.000030	172.31.22.121	185.216.74.12	TCP	443 → 58146 [SYN, ACK] Seq=0 Ack=1 Win=62643 Len=0 MS
3	0.009580	185.216.74.12	172.31.22.121	TCP	58146 → 443 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1
4	0.009580	185.216.74.12	172.31.22.121	TLSv1.3	Client Hello
5	0.009618	172.31.22.121	185.216.74.12	TCP	443 → 58146 [ACK] Seq=1 Ack=518 Win=62208 Len=0 TSval=
6	0.010196	172.31.22.121	185.216.74.12	TLSv1.3	Server Hello, Change Cipher Spec, Application Data
7	0.010210	172.31.22.121	185.216.74.12	TCP	443 → 58146 [PSH, ACK] Seq=2897 Ack=518 Win=62208 Len=
8	0.011707	172.31.22.121	185.216.74.12	TLSv1.3	Application Data, Application Data, Application Data
9	0.019663	185.216.74.12	172.31.22.121	TCP	58146 → 443 [ACK] Seq=518 Ack=2897 Win=71680 Len=0 TS
10	0.019674	185.216.74.12	172.31.22.121	TCP	58146 → 443 [ACK] Seq=518 Ack=4097 Win=74240 Len=0 TS
11	0.021201	185.216.74.12	172.31.22.121	TCP	58146 → 443 [ACK] Seq=518 Ack=4565 Win=77312 Len=0 TS
12	0.046685	185.216.74.12	172.31.22.121	TLSv1.3	Change Cipher Spec, Application Data
13	0.046685	185.216.74.12	172.31.22.121	TLSv1.3	Application Data
14	0.046703	172.31.22.121	185.216.74.12	TCP	443 → 58146 [ACK] Seq=4565 Ack=582 Win=62208 Len=0 TS
15	0.046719	172.31.22.121	185.216.74.12	TCP	443 → 58146 [ACK] Seq=4565 Ack=1268 Win=61568 Len=0 T
16	0.046891	172.31.22.121	185.216.74.12	TLSv1.3	Application Data
17	0.047639	172.31.22.121	185.216.74.12	TLSv1.3	Application Data
18	0.047680	172.31.22.121	185.216.74.12	TLSv1.3	Application Data
19	0.047759	172.31.22.121	185.216.74.12	TLSv1.3	Application Data, Application Data

Figure 9. TLS 1.3 Wireshark Packet Capture

Excluding the TCP handshake and its subsequent acknowledgement messages, Figure 8 demonstrates that the TLS 1.2 handshake takes five packets to complete (packet numbers 4, 6, 8, 12, and 14) and encrypted data begins with the fourth packet in the handshake (packet number 12). Note that encrypted data is represented by the Application Data note of the “Info” column on the far right. In contrast, the TLS 1.3 handshake in Figure 9 completes in just 3 packets (packet numbers 4, 6, and 12) and encrypted data starts on the second packet (packet 6). This is accomplished largely due to the removal of the older cipher suites from TLS 1.2 [1]. In TLS 1.3, the Client Hello and Client Key Exchange messages can now be combined at the beginning of the handshake (packet 4). The server in turn now has everything required to begin encryption and thus does so beginning with the end of its Server Hello message, as well as acknowledging the client encryption (packet 6). Lastly, the client acknowledges the servers encryption with the Change Cipher Spec to complete the TLS 1.3 handshake (packet 12).

E. SSLKEYLOGFILE

Simple TLS key logging can be accomplished on a client by instantiating the SSLKEYLOGFILE operating system environment variable and using a web browser that supports this capability, such as Mozilla Firefox web browser or Google Chrome web

browser. Once established, this environment variable will log to the specified file the numerous encryption keys employed by the TLS protocol such as [16]:

- CLIENT_HANDSHAKE_TRAFFIC_SECRET
- CLIENT_TRAFFIC_SECRET_0
- SERVER_HANDSHAKE_TRAFFIC_SECRET
- SERVER_TRAFFIC_SECRET_0

Here the client/server HANDSHAKE_TRAFFIC_SECRET keys are used specifically for the encryption during the TLS 1.3 handshake process. These keys are derived from the ECDHE “Pre-Master Secret” described in Figure 6 but called “Handshake Secret” by the TLS 1.3 RFC. The pre-master secret is a result of a nested hashing function. First, the client hello and server hello messages are concatenated and hashed, then the resulting hash is concatenated with a unique string and the pre-master secret to create the HANDSHAKE_TRAFFIC_SECRET keys.

The client/server TRAFFIC_SECRET_0 keys are used for the subsequent application data, such as HTTPS. These keys are derived from the “Master Secret” described in Figure 6 and referred as such in the TLS 1.3 RFC [1]. These keys use a similar nested hashing function to the handshake keys. First, the client hello and server finished messages are concatenated and hashed, then the resulting hash is concatenated with a unique string and the master secret to create the TRAFFIC_SECRET_0 keys.

For new connections, the process begins with the (EC) DHE $\rightarrow$ HKDF-Extract = Handshake Secret section of the overall key derivation flow as shown in Figure 10.

```

(EC)DHE -> HKDF-Extract = Handshake Secret
|
+-----> Derive-Secret(., "c hs traffic",
|                               ClientHello...ServerHello)
|                               = client_handshake_traffic_secret
|
+-----> Derive-Secret(., "s hs traffic",
|                               ClientHello...ServerHello)
|                               = server_handshake_traffic_secret
|
v
Derive-Secret(., "derived", "")
|
v
0 -> HKDF-Extract = Master Secret
|
+-----> Derive-Secret(., "c ap traffic",
|                               ClientHello...server Finished)
|                               = client_application_traffic_secret_0
|
+-----> Derive-Secret(., "s ap traffic",
|                               ClientHello...server Finished)
|                               = server_application_traffic_secret_0
|

```

Figure 10. TLS 1.3 Key Derivation Process. Derived from RFC 8446 [1].

Note that the client/server TRAFFIC_SECRET_0 keys are referred to as client/server application_traffic_secret_0 in the key derivation process as shown in Figure 10. The process for enabling SSLKEYLOGFILE and use of its various keys will be discussed in detail in Chapter III.

In this chapter, an overview of TLS, its significance, history, and changes to the latest version were discussed. Also, an introduction to the cryptographic variable of significance, SSLKEYLOGFILE, was given. In the next chapter, the methodology of hacking the TLS 1.3 protocol from both a red and blue team perspective will be presented.

III. METHODOLOGY

In Chapter II, the necessary technical background of TLS was discussed. In this chapter, the finer details of how various experiments were set up will be presented.

A. SYSTEM SPECIFICATIONS

Two test systems were utilized for the purpose of this research: a personal computer and an Amazon Web Services (AWS) Elastic Compute Cloud (EC2) instance hosted in the cloud.

(1) Personal Computer Specifications:

CPU: AMD Ryzen 9 5900X

RAM: 32GB Crucial DDR4 3600

Drive: 1TB Sabrent Rocket 4.0 Plus NVMe SSD

Operating System: Microsoft Windows 11 Version 22H2 (OS build 22621)

Browser: Mozilla Firefox Version 111

Packet analyzer: Wireshark Version 4.0.5

(2) AWS EC2 Virtualized Instance Specifications:

CPU: Intel Xeon E5-2686 v4 (two cores)

RAM: 4GB

Drive: 30GB

Operating System: Ubuntu 22.04.2 LTS

Web server: Apache 2.4.52

Certificate Authority: Let's Encrypt, PKCS #1 RSA Encryption, 2048-bit modulus.

B. METHOD FOR MAN-ON-THE-SIDE (MOTS) COLLECTION AND DECRYPTION

With TLS 1.3 removing the TLS inspection feature, the HTTPS and other TLS 1.3 protocol traffic is securely encrypted and no longer available for IPS or IDS protection and analysis. The proposed method to work around this issue is to identify the high-value asset (server or client), employ TLS key logging capability to collect the encryption keys, capture network traffic going to and from the asset, and then move the keys to a third party.

The proposed method would greatly benefit from the advantage that a large amount of network traffic could be passively collected on the side and subsequently decrypted, instead of exfiltrating the data from the source asset. The TLS keys would still have to be collected from the asset itself, but the size of the key file versus the data it can decrypt should be dramatically smaller in comparison. This method is investigated in the next chapter.

Additionally, it is expected that compression of the TLS key file would further reduce the size of the key file, conserving significantly more network bandwidth and in turn making the key file extraction less observable. Note that this methodology would potentially work for both red team and blue team actors. The red team could utilize the stealthy exfiltration of compressed TLS keys and decrypt the passively collected network traffic to gather intelligence. The blue team could also utilize the key logging, forwarding the TLS keys to an IPS or IDS for decryption and subsequent analysis and protective actions. This method is depicted in Figure 11.

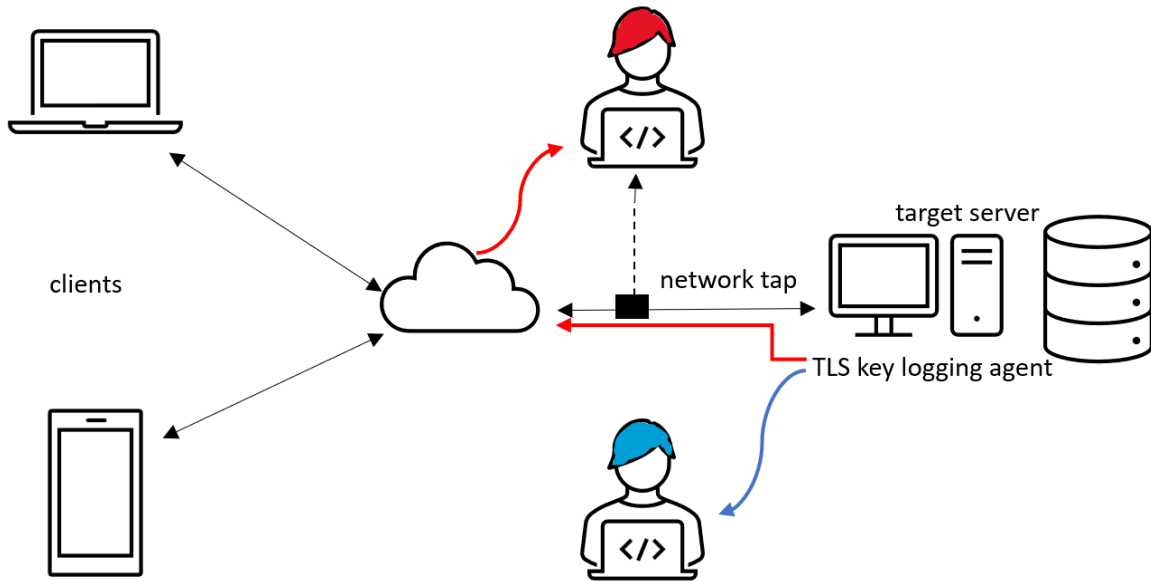


Figure 11. MotS Collection and Decryption Workflow

The methods of collection and decryption will be further discussed in the following sections.

C. HOW TO DECRYPT MAN-ON-THE-SIDE TRAFFIC

Given a packet capture session between a client and a server, the negotiation or handshake of the session must be captured in sequence followed by the encrypted data. Packet capture techniques will not be discussed in this work, but there are a wide range of available methods [17], [18]. For the purpose of this research, the Unix-based tool tcpdump [17] was used. Tcpdump was chosen for its robust history and flexibility in filtering network traffic. The tool was also advantageous in that it runs well from the command line when connecting remotely to the webserver. One of the more complex usages of tcpdump was when capturing Gmail traffic: the IP addresses had to be specifically isolated; otherwise, too much noise would be collected in the packet capture. An example of the tcpdump collection is shown below:

```
sudo tcpdump -i eth0 -v -s 0 -n 'host 192.168.28.128 and
net 142.0.0.0/8' -w packetcapture.pcap
```

where the flags `-i eth0` specifies the first ethernet interface, `-v` enables verbose mode to better understand the live capture, `-s 0` enables a snap length of zero to capture the packets in their entirety, `-n` disables domain name resolution, `host 192.168.28.128` and `net 142.0.0.0/8` narrows down the capture to only the conversation between the host machine IP and the network range of IPs associated with Gmail, and lastly `-w packetcapture.pcap` writes the collected packets to the `packetcapture.pcap` file.

Once a packet capture has been collected, two sets of two keys generated by the serving cryptography engine, OpenSSL for the webserver (Schannel for Microsoft Windows), are required to decrypt the session. Since TCP operates as a split, full-duplex system, there is one set of keys used for the client-to-server data and another set of keys used for the server-to-client data transmission. The client-side data is encrypted using the `CLIENT_HANDSHAKE_TRAFFIC_SECRET` and `CLIENT_TRAFFIC_SECRET_0` keys while the server-side data is encrypted using the `SERVER_HANDSHAKE_TRAFFIC_SECRET` and `SERVER_TRAFFIC_SECRET_0` keys [18].

While these keys are generated at both the client and server, they are not nominally stored outside of RAM, or volatile memory. In order to use these keys for MotS decryption, they must be written from memory to disk. The decryption process is shown in Figure 12.

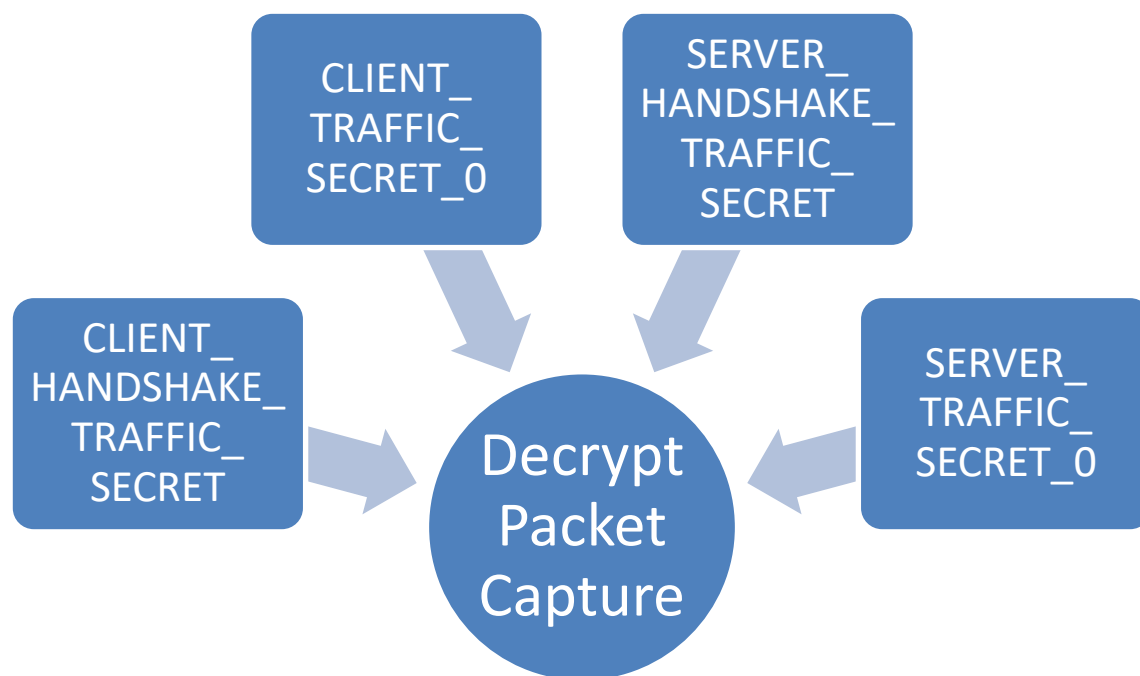


Figure 12. Decryption of a Packet Capture

These four secrets will be referred to as the quadruple and are just one piece of the greater TLS 1.3 key derivation process. This process is described in detail in the TLS 1.3 RFC 8446 and graphically depicted in Figure 10, as discussed in Chapter II, Section E.

D. CLIENT-SIDE KEY LOGGING

Client-side logging of the cryptographic keys may be accomplished in a variety of ways. From both a red and blue team perspective, client-side logging is advantageous in gaining a more holistic picture of client activities versus just one of its lines of communication with a specific server (assuming just a single point of logging). Both teams would likely develop their own key logging agent based upon their end goal: red team would want to be the most lightweight and least visible while the blue team would ensure full logging and forwarding of keys to a central Security Operations Center (SOC).

For this research, the simplest method was to utilize the Microsoft Windows environment variable system and the Mozilla Firefox web browser. This method also works when utilizing the Google Chrome web browser [16].

In Windows 11, opening “Edit the system environment variables” from the Control Panel is the first step. Then after selecting the “Environment Variables” button, the option to create new user and system environment variables will be available. Creating a user environment variable is the best option using the following syntax:

Variable: SSLKEYLOGFILE

Value: %USERPROFILE%\Desktop\sslkey.log

where %USERPROFILE% is the name of the logged in user and \Desktop\sslkey.log is the destination directory and desired filename, which will contain all the TLS keys [16]. An example of this final state is shown in Figure 13.

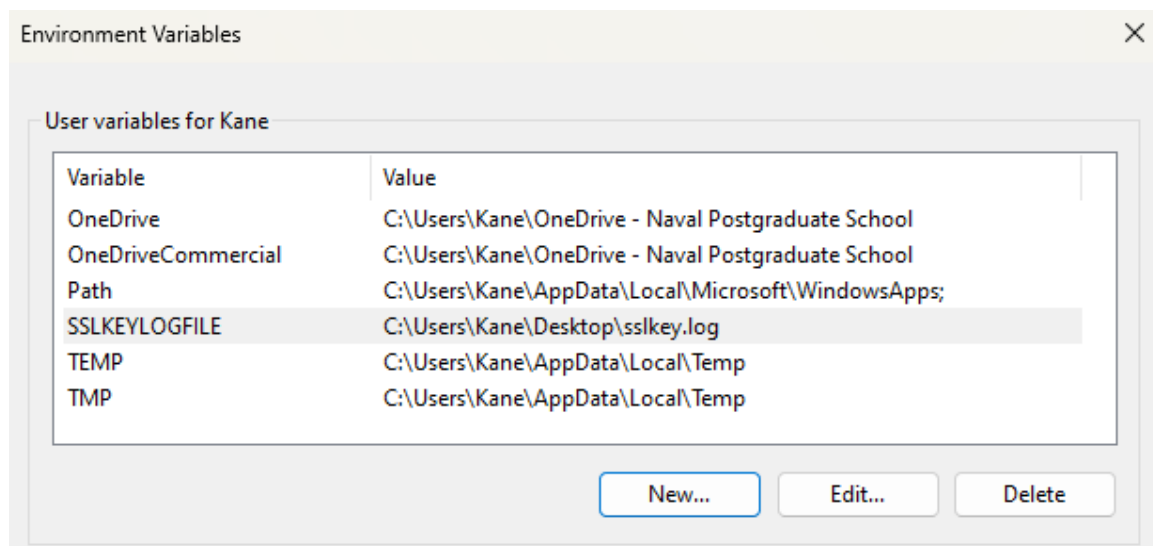


Figure 13. Windows Environment Variables

Once these steps are completed, opening a Chrome or Firefox web browser will create the associated SSLKEYLOGFILE variable value file and start logging the TLS keys.

E. CREATING A WEB SERVER

There are various options for web server products ranging from free and open-source software (FOSS) to expensive, proprietary solutions. Apache was chosen for this research since it is relatively easy to implement and there is substantial documentation for

all aspects of this solution. It is also the leading web server application suite for active websites [19].

(1) Installing Apache

Once the AWS EC2 instance was up and running, creating a webserver begins with installing apache2:

```
sudo apt install apache2
```

Configuration of DNS was performed using the AWS Route 53 service and will not be covered in detail here. Also, configuration of signed website certificates utilized the Let's Encrypt service with their provided tutorials.

(2) TLS version 1.3 configuration

To ensure the webserver was using the latest TLS version, all other SSL and TLS protocol versions were disabled. This was accomplished by editing the following configuration here:

```
/etc/letsencrypt/options-ssl-apache.conf
```

The `SSLProtocol` line was changed from the default configuration to `+TLSv1.3`, which constrains the web server from servicing any older version of the protocol. The final configuration file state is shown in Figure 14.

```

# This file contains important security parameters. If you modify this file
# manually, Certbot will be unable to automatically provide future security
# updates. Instead, Certbot will print and log an error message with a path to
# the up-to-date file that you will need to refer to when manually updating
# this file.

SSLEngine on

# Intermediate configuration, tweak to your needs
#SSLProtocol      all -SSLv2 -SSLv3 -TLSv1 -TLSv1.1
SSLProtocol       +TLSv1.3
SSLCipherSuite     ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-
SSLHonorCipherOrder off
SSLSessionTickets off

SSLOptions +StrictRequire

# Add vhost name to log entries:
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\"" vhost_combined
LogFormat "%v %h %l %u %t \"%r\" %>s %b" vhost_common

```

Figure 14. options-ssl-apache.conf Configuration File

Additionally, the configuration file is where the supported cipher suites are specified. This is shown in the `SSLCipherSuite` line. The configuration was tested and verified by using the Qualys SSL Labs free web server test with the results shown in Appendix A.

F. SERVER-SIDE KEY LOGGING

Server-side key logging can be accomplished using a variety of methods from paid, third-party products to FOSS solutions. The server-side key logging is critical to the blue team in order to decrypt and analyze the data of potential attacks against the system. This is also vital from a red team perspective in that all traffic to and from the server is available now as a collection point of intelligence since decryption is possible.

The method used to implement server-side key logging in this research used the open-source package `libssl-dev` to install a specific `libsslkeylog.so` file (shared object library) for the web server version of OpenSSL [20]. The `apache2` service override configuration file would then have to be edited to include the newly generated shared object library and the `SSLKEYLOGFILE` environment variable as shown in Figure 15.

```
### Editing /etc/systemd/system/apache2.service.d/override.conf
### Anything between here and the comment below will become the new contents of the file

[Service]
Environment=LD_PRELOAD=/usr/local/lib/libsslkeylog.so
Environment=SSLKEYLOGFILE=/tmp/your.keys

### Lines below this comment will be discarded

### /lib/systemd/system/apache2.service
```

Figure 15. Apache2 Service Override Configuration File Modifications

From here, the keys will be written to the `/tmp/system-private-*-apache2.service-*/tmp/your.keys` file where `*` represents unique identifiers for the specific service instance. In this web server case, the full file path was: `/tmp/systemd-private-d653abb949fc4c58b1af25bb4ca78ca8-apache2.service-KsY6bQ/tmp/your.keys`

Lastly, the file has to be writable by the default webserver process, so the command `chmod 602 your.keys` was used.

In this chapter, the methodology of collecting and decrypting data from the TLS 1.3 protocol, from both a red and blue team perspective, was described. The next chapter will present the various experiments that could be utilized by future red and blue team operators.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. EXPERIMENT RESULTS AND ANALYSIS

In Chapter III, the methodology behind the various tests and experiments was described. In this chapter, we will discuss the finer details of the experiments and present the results.

A. CLIENT SSLKEYLOGFILE

The client SSLKEYLOGFILE resides on the client endpoint and may be enabled as described in Chapter III. A sample of the file is shown in Table 2.

Table 2. Client SSLKEYLOGFILE Showing its Captured TLS Encryption/Decryption Keys

CLIENT HANDSHAKE TRAFFIC SECRET
9527eb9d34467c853528c872d62c83c39340f358f98be898bd8358f1e6917dc22e5e62e3e76eb0fcc4ff8da82f37ab1cfdeac9aab5000cd9e578e21ee158f861
e2d22a0c1cc94c21c7e97272c8fa43635c76d5290421c1b2bed34c1d29efd211b18639c3fb77e1c6b344ed6ee85ada5f3ef977d951b8cc85b7424330c20e682d
SERVER HANDSHAKE TRAFFIC SECRET
9527eb9d34467c853528c872d62c83c39340f358f98be898bd8358f1e6917dc264391cf902810718a3d51e8b0ff68a590954998d10c307b75bb93ae7a0fe8692
e2d22a0c1cc94c21c7e97272c8fa43635c76d5290421c1b2bed34c1d29efd2119cff763c2b08d48c97fbf8069ad78bfde3a6bbd9a5fe9f14bbada2ee152ea971
CLIENT TRAFFIC SECRET 0
9527eb9d34467c853528c872d62c83c39340f358f98be898bd8358f1e6917dc24a867834b7cfb144969065c35af9fbc03d521bd03414dfabbedda75621a62a78
e2d22a0c1cc94c21c7e97272c8fa43635c76d5290421c1b2bed34c1d29efd211ff5b353dfc24a6787d7e67f5bf8dc1d31f33a0c0d66dbb5337967d80e5ba3ede
SERVER TRAFFIC SECRET 0
9527eb9d34467c853528c872d62c83c39340f358f98be898bd8358f1e6917dc2d2dd9e7084a12169963f80c9011ff7e1ff9ae21bcacff0b8ac4c464b7f274425
e2d22a0c1cc94c21c7e97272c8fa43635c76d5290421c1b2bed34c1d29efd211df9d565a5408afbb6f6da34a9ac102233eee140a9dd99d7cbec69dae5cf4d300

It is important to note that either the blue team or red team actor would need direct access only to the client to view or retrieve the key file under client-based scenarios. If a specific communication to a server was desired, the same set of keys could also be used and potentially logged on that server, but only access to one endpoint of the communication is required. The logging and retrieval of keys may be accomplished in a variety of ways and is ultimately left up to the actor to determine based upon their red or blue team objectives.

This sample file was generated over the course of roughly 181 days, on a personal desktop, to get a better understanding of how many keys would likely be generated. This resulted in 2 963 486 keys being logged, an average of 16 373 keys per day.

B. SERVER SSLKEYLOGFILE

The server SSLKEYLOGFILE similarly resembles the client SSLKEYLOGFILE but created on the server side. A sample of the server SSLKEYLOGFILE is shown in Table 3.

Table 3. Server SSLKEYLOGFILE Showing its Captured TLS Encryption/Decryption Keys

CLIENT HANDSHAKE TRAFFIC SECRET
c1df54b8319cd232af9f151ae7c6ab378aff8b95c105756ce7c6e16967aa37c419ef78acfa29272c2856fab942d1f9de96531a83dce224f52bdab452d6dbb337
0936bdeef564390932eb346053dc188c9b90391758cef0c1d3efcb52b9e0f91bced1a5ae816058b547fbf5719a427f18f1f1505dde3fce6c3bb7dd28f94e7b45
SERVER HANDSHAKE TRAFFIC SECRET
c1df54b8319cd232af9f151ae7c6ab378aff8b95c105756ce7c6e16967aa37c47a29da983d1fb263b1ed35c25bbb9146eb769dee2a90ca69be6088f70d7c75a1
0936bdeef564390932eb346053dc188c9b90391758cef0c1d3efcb52b9e0f91bd5c3ad022fdbcb34a647b914c3c004cb8edafd343b6338684343070321c7ca9e7
CLIENT TRAFFIC SECRET 0
c1df54b8319cd232af9f151ae7c6ab378aff8b95c105756ce7c6e16967aa37c48d09f24d2f323ef1889698eebc0fabec2ca4fdcf7141ce38a37c2b5acd575dd4
0936bdeef564390932eb346053dc188c9b90391758cef0c1d3efcb52b9e0f91be5d82053f87c27571602f949f34fb3751b63f065b1e841253beb3f72d0092ec9
SERVER TRAFFIC SECRET 0
c1df54b8319cd232af9f151ae7c6ab378aff8b95c105756ce7c6e16967aa37c42823f5fa38ed8a893e8ea35a8acfc9588600db07d3af9965191076257bde6019
0936bdeef564390932eb346053dc188c9b90391758cef0c1d3efcb52b9e0f91b105111f1c7b77d659e2fa4990c137bb879521465b43e64dc4378fd5aa99d52652

The server SSLKEYLOGFILE was created by using the methods described in Chapter III. For the course of this test, 7685 keys were logged over roughly 37 days. This resulted in a mean average of 208 keys per day. Consequently, the default webserver logging stored in the `access.log` (of `/var/log/apache2/`) averaged 264 entries per day during this period. A sample of the webserver `access.log` is shown in Figure 16.

```

ubuntu@ip-172-22-150-153:/var/log/apache2$ head access.log
92.253.218.15 - - [08/May/2023:00:27:57 +0000] "GET / HTTP/1.1" 200 427 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/601.7.7 (KHTML, like Gecko) Version/9.1.2 Safari/601.7.7"
54.191.17.49 - - [08/May/2023:00:33:35 +0000] "GET / HTTP/1.1" 301 574 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.181 Safari/537.36"
54.191.17.49 - - [08/May/2023:00:33:35 +0000] "GET / HTTP/1.1" 200 5246 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.181 Safari/537.36"
43.154.141.71 - - [08/May/2023:00:38:15 +0000] "HEAD /Core/Skin/Login.aspx HTTP/1.1" 404 140 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.0.0 Safari/537.36"
135.125.244.48 - - [08/May/2023:01:04:37 +0000] "GET /.env HTTP/1.1" 404 491 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.129 Safari/537.36"
135.125.244.48 - - [08/May/2023:01:04:37 +0000] "POST / HTTP/1.1" 200 475 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.129 Safari/537.36"
5.188.210.227 - - [08/May/2023:01:28:52 +0000] "GET http://5.188.210.227/echo.php HTTP/1.1" 404 455 "https://www.google.com/" "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.77 Safari/537.36"
185.254.196.173 - - [08/May/2023:01:43:05 +0000] "GET /.env HTTP/1.1" 404 491 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.129 Safari/537.36"
185.254.196.173 - - [08/May/2023:01:43:06 +0000] "POST / HTTP/1.1" 200 475 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.129 Safari/537.36"
43.154.141.71 - - [08/May/2023:01:56:09 +0000] "HEAD /Core/Skin/Login.aspx HTTP/1.1" 404 140 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.0.0 Safari/537.36"

```

Figure 16. Webserver Access.log Sample, Showing Remote IP Address, Date/Time of Access, HTTP Method, Response Codes, and User Agent String

The access.log shows that the same IP address can have multiple entries for the same logged time, which likely results in the same session keys being used for multiple log entries. In other words, a higher number of entries in access.log versus the number of keys being logged indicates that the same keys are being used for multiple entries in the webserver logs.

C. DECRYPTION

Decryption of TLS 1.3 traffic was demonstrated by using the Wireshark TLS Protocol plugin. The protocol plugin configuration is shown in Figure 17.

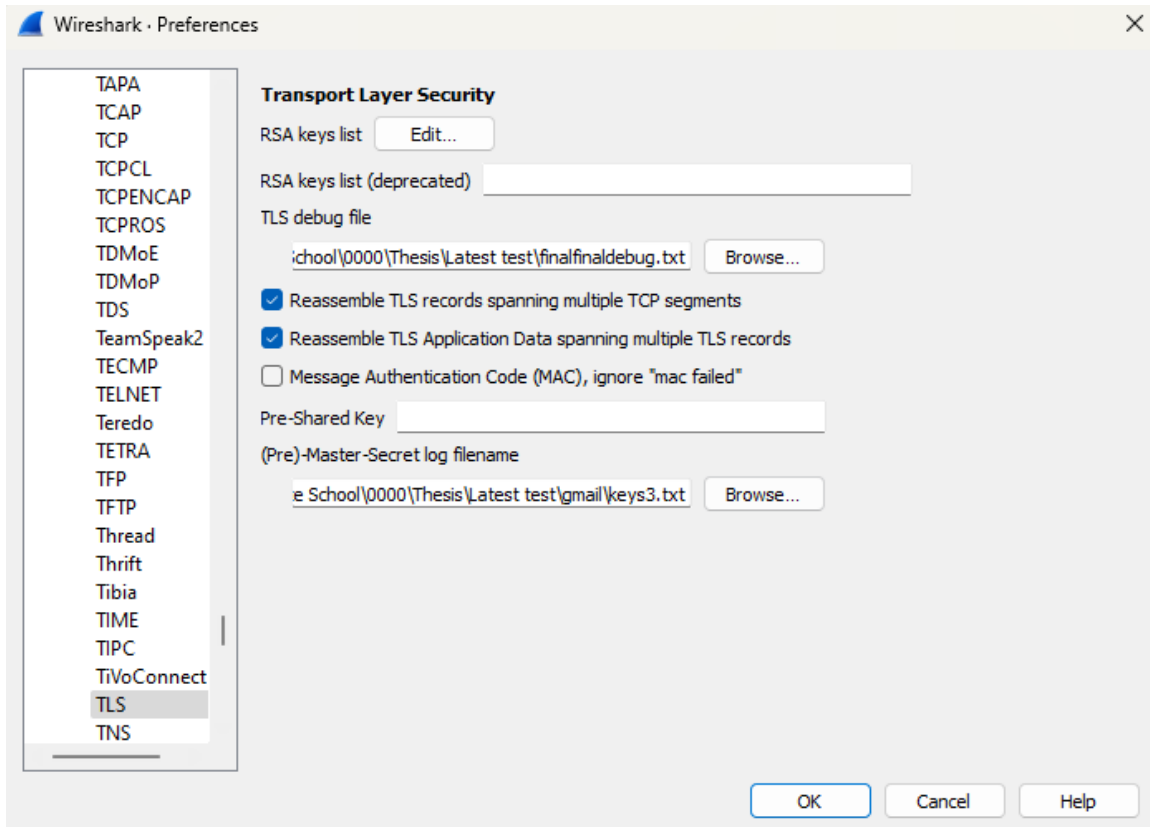


Figure 17. Wireshark TLS Plugin Configuration via Edit → Preferences from Home Screen

The most important aspect of the configuration page is to correctly identify the associated “(Pre)-Master-Secret log filename” file path. This is precisely the file that is created by the SSLKEYLOGFILE process on either the client or the server, which contains the keys exemplified in Tables 2 and 3.

The packet capture when encrypted does not allow for much information to be gained other than that a conversation is happening between a client and a server utilizing the TLS 1.3 protocol. The encrypted packet capture is shown in Figure 18.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	73.162.150.7	172.22.150.153	TCP	66	51402 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
2	0.000034	172.22.150.153	73.162.150.7	TCP	66	443 → 51402 [SYN, ACK] Seq=0 Ack=1 Win=62727 Len=0 MSS=8961 SACK_PERM WS=128
3	0.033609	73.162.150.7	172.22.150.153	TCP	54	51402 → 443 [ACK] Seq=1 Ack=1 Win=262656 Len=0
4	0.035642	73.162.150.7	172.22.150.153	TLSv1.3	571	Client Hello
5	0.035680	172.22.150.153	73.162.150.7	TCP	54	443 → 51402 [ACK] Seq=1 Ack=518 Win=62336 Len=0
6	0.036488	172.22.150.153	73.162.150.7	TLSv1.3	2974	Server Hello, Change Cipher Spec, Application Data
7	0.036504	172.22.150.153	73.162.150.7	TCP	1230	443 → 51402 [PSH, ACK] Seq=2921 Ack=518 Win=62336 Len=1176 [TCP segment of a reassembled PDU]
8	0.037419	172.22.150.153	73.162.150.7	TLSv1.3	533	Application Data, Application Data
9	0.069558	73.162.150.7	172.22.150.153	TLSv1.3	118	Change Cipher Spec, Application Data
10	0.069879	172.22.150.153	73.162.150.7	TLSv1.3	133	Application Data
11	0.069998	172.22.150.153	73.162.150.7	TLSv1.3	133	Application Data
12	0.071775	73.162.150.7	172.22.150.153	TCP	54	51402 → 443 [ACK] Seq=518 Ack=4097 Win=262656 Len=0
13	0.071784	172.22.150.153	73.162.150.7	TCP	54	[TCP Dup ACK 10#1] 443 → 51402 [ACK] Seq=4734 Ack=582 Win=62336 Len=0
14	0.071797	73.162.150.7	172.22.150.153	TCP	54	51402 → 443 [ACK] Seq=518 Ack=4576 Win=262144 Len=0
15	0.071801	73.162.150.7	172.22.150.153	TLSv1.3	743	Application Data
16	0.072222	172.22.150.153	73.162.150.7	TLSv1.3	551	Application Data
17	0.101488	73.162.150.7	172.22.150.153	TCP	54	51402 → 443 [ACK] Seq=1271 Ack=4734 Win=262144 Len=0
18	0.141232	73.162.150.7	172.22.150.153	TCP	54	51402 → 443 [ACK] Seq=1271 Ack=5231 Win=261632 Len=0
19	0.141275	73.162.150.7	172.22.150.153	TCP	66	51403 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
20	0.141301	172.22.150.153	73.162.150.7	TCP	66	443 → 51403 [SYN, ACK] Seq=0 Ack=1 Win=62727 Len=0 MSS=8961 SACK_PERM WS=128
21	0.158836	73.162.150.7	172.22.150.153	TLSv1.3	671	Application Data
22	0.159160	172.22.150.153	73.162.150.7	TLSv1.3	570	Application Data
23	0.169578	73.162.150.7	172.22.150.153	TCP	54	51403 → 443 [ACK] Seq=1 Ack=1 Win=262656 Len=0
24	0.176530	73.162.150.7	172.22.150.153	TLSv1.3	571	Client Hello
25	0.176560	172.22.150.153	73.162.150.7	TCP	54	443 → 51403 [ACK] Seq=1 Ack=518 Win=62336 Len=0
26	0.177349	172.22.150.153	73.162.150.7	TLSv1.3	298	Server Hello, Change Cipher Spec, Application Data, Application Data
27	0.209609	73.162.150.7	172.22.150.153	TLSv1.3	118	Change Cipher Spec, Application Data
28	0.209875	172.22.150.153	73.162.150.7	TLSv1.3	133	Application Data
29	0.232032	73.162.150.7	172.22.150.153	TCP	54	51402 → 443 [ACK] Seq=1888 Ack=5747 Win=262656 Len=0
30	0.289648	73.162.150.7	172.22.150.153	TCP	54	51403 → 443 [ACK] Seq=582 Ack=324 Win=262400 Len=0
31	5.160021	172.22.150.153	73.162.150.7	TLSv1.3	78	Application Data
32	5.160077	172.22.150.153	73.162.150.7	TCP	54	443 → 51402 [FIN, ACK] Seq=5771 Ack=1888 Win=61184 Len=0
33	5.193767	73.162.150.7	172.22.150.153	TCP	54	51402 → 443 [ACK] Seq=1888 Ack=5772 Win=262656 Len=0

Figure 18. Encrypted Packet Capture Showing Encrypted Packets after Handshake Agreement

This encrypted packet capture clearly shows “Client Hello” and “Server Hello” messages in packets number 4 and 6, respectively, but the rest of the conversation is encrypted. Encrypted contents of a packet are represented by the “Application Data” notation in the “Info” column.

After selecting the appropriate (Pre)-Master-Secret log filename in the TLS protocol preferences, the Wireshark protocol analyzer will successfully decrypt the packet capture. The previously shown Application Data notations will change to what the protocol analyzer recognizes and repopulate the Info column. The decrypted packet capture is shown in Figure 19.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	73.162.150.7	172.22.150.153	TCP	66	51402 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
2	0.000034	172.22.150.153	73.162.150.7	TCP	66	443 → 51402 [SYN, ACK] Seq=0 Ack=1 Win=62727 Len=0 MSS=8961 SACK_PERM WS=128
3	0.033609	73.162.150.7	172.22.150.153	TCP	54	51402 → 443 [ACK] Seq=1 Ack=1 Win=262656 Len=0
4	0.035642	73.162.150.7	172.22.150.153	TLsv1.3	571	Client Hello
5	0.035680	172.22.150.153	73.162.150.7	TCP	54	443 → 51402 [ACK] Seq=1 Ack=518 Win=62336 Len=0
6	0.036488	172.22.150.153	73.162.150.7	TLsv1.3	2974	Server Hello, Change Cipher Spec, Encrypted Extensions
7	0.036504	172.22.150.153	73.162.150.7	TCP	1230	443 → 51402 [PSH, ACK] Seq=2921 Ack=518 Win=62336 Len=1176 [TCP segment of a reassembled PDU]
8	0.037419	172.22.150.153	73.162.150.7	TLsv1.3	533	Certificate, Certificate Verify, Finished
9	0.069558	73.162.150.7	172.22.150.153	TLsv1.3	118	Change Cipher Spec, Finished
10	0.069879	172.22.150.153	73.162.150.7	TLsv1.3	133	New Session Ticket
11	0.069998	172.22.150.153	73.162.150.7	TLsv1.3	133	New Session Ticket
12	0.071775	73.162.150.7	172.22.150.153	TCP	54	51402 → 443 [ACK] Seq=518 Ack=4097 Win=262656 Len=0
13	0.071784	172.22.150.153	73.162.150.7	TCP	54	[TCP Dup ACK 10#1] 443 → 51402 [ACK] Seq=4734 Ack=582 Win=62336 Len=0
14	0.071797	73.162.150.7	172.22.150.153	TCP	54	51402 → 443 [ACK] Seq=518 Ack=4576 Win=262144 Len=0
15	0.071801	73.162.150.7	172.22.150.153	HTTP	743	GET / HTTP/1.1
16	0.072222	172.22.150.153	73.162.150.7	HTTP	551	HTTP/1.1 200 OK (text/html)
17	0.101488	73.162.150.7	172.22.150.153	TCP	54	51402 → 443 [ACK] Seq=1271 Ack=4734 Win=262144 Len=0
18	0.141232	73.162.150.7	172.22.150.153	TCP	54	51402 → 443 [ACK] Seq=1271 Ack=5231 Win=261632 Len=0
19	0.141275	73.162.150.7	172.22.150.153	TCP	66	51403 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
20	0.141301	172.22.150.153	73.162.150.7	TCP	66	443 → 51403 [SYN, ACK] Seq=0 Ack=1 Win=62727 Len=0 MSS=8961 SACK_PERM WS=128
21	0.158836	73.162.150.7	172.22.150.153	HTTP	671	GET /favicon.ico HTTP/1.1
22	0.159160	172.22.150.153	73.162.150.7	HTTP	570	HTTP/1.1 404 Not Found (text/html)
23	0.169578	73.162.150.7	172.22.150.153	TCP	54	51403 → 443 [ACK] Seq=1 Ack=1 Win=262656 Len=0
24	0.176530	73.162.150.7	172.22.150.153	TLsv1.3	571	Client Hello
25	0.176560	172.22.150.153	73.162.150.7	TCP	54	443 → 51403 [ACK] Seq=1 Ack=518 Win=62336 Len=0
26	0.177349	172.22.150.153	73.162.150.7	TLsv1.3	298	Server Hello, Change Cipher Spec, Encrypted Extensions, Finished
27	0.209609	73.162.150.7	172.22.150.153	TLsv1.3	118	Change Cipher Spec, Finished
28	0.209875	172.22.150.153	73.162.150.7	TLsv1.3	133	New Session Ticket
29	0.232032	73.162.150.7	172.22.150.153	TCP	54	51402 → 443 [ACK] Seq=1888 Ack=5747 Win=262656 Len=0
30	0.289648	73.162.150.7	172.22.150.153	TCP	54	51403 → 443 [ACK] Seq=582 Ack=324 Win=262400 Len=0
31	5.160021	172.22.150.153	73.162.150.7	TLsv1.3	78	Alert (Level: Warning, Description: Close Notify)
32	5.160077	172.22.150.153	73.162.150.7	TCP	54	443 → 51402 [FIN, ACK] Seq=5771 Ack=1888 Win=61184 Len=0
33	5.193767	73.162.150.7	172.22.150.153	TCP	54	51402 → 443 [ACK] Seq=1888 Ack=5772 Win=262656 Len=0

Figure 19. Decrypted Packet Capture Showing Plaintext Communications Highlighted in Green

Figure 19 demonstrates that the packet capture was successfully decrypted, and much new information can be learned from this conversation. After the previously encrypted Server Hello message, multiple packets now have a proper description. We can now ascertain that this conversation successfully executed a GET request for the root web directory (represented in packet 15) and then failed to retrieve the favicon, the URL icon that is associated with a webpage.

D. COMPRESSION TECHNIQUES

In order to best accomplish a stealthy or lightweight extraction of the SSLKEYLOGFILE by a red or blue team actor, two compression methods were analyzed: GNU Gzip [21] and 7zip [22]. Specifically, from a blue team perspective, compression techniques would be valuable in reducing network bandwidth demands and for long-term key retention in order to aid in potential network compromise investigations. Table 4 summarizes the results of testing a single key file of 404 entries and then expanding the file up to one million times its original size, resulting in 404 million keys.

Table 4. Gzip and 7zip Compression Results for Various Key File Sizes

Sessions	Key File Size (B)	Gzip File Size (B)	Gzip Compression %	7zip File Size (B)	7zip Compression %
1	61200	19900	32.5163	17600	28.7582
10	612000	199000	32.5163	17800	2.9085
100	6120000	1940000	31.6993	18600	0.3039
200	12240000	3880000	31.6993	19500	0.1593
300	18360000	5820000	31.6993	20400	0.1111
400	24480000	7760000	31.6993	21300	0.0870
500	30600000	9700000	31.6993	22200	0.0725
600	36720000	11640000	31.6993	23100	0.0629
700	42840000	13580000	31.6993	24000	0.0560
800	48960000	15520000	31.6993	24900	0.0509
900	55080000	17460000	31.6993	25700	0.0467
1000	61200000	19400000	31.6993	26600	0.0435
10000	612000000	194000000	31.6993	265000	0.0433
100000	6120000000	1940000000	31.6993	2480000	0.0405
1000000	61200000000	19400000000	31.6993	24800000	0.0405

From Table 4, it is evident that 7zip is the preferred method of compression resulting in substantially smaller key file sizes, plateauing at 0.0405% of the original key file size. The Gzip compression algorithm plateaus much sooner than 7zip, resulting in a best compression ratio of 31.6993%. When compressing a small volume of keys, around 400 or less, the compression method does not change the resulting benefit significantly. This is likely due to the metadata associated with the file creation itself that does not benefit from external compression techniques.

Another method to better see the benefits of 7zip compression is shown in graphical form in Figure 20.

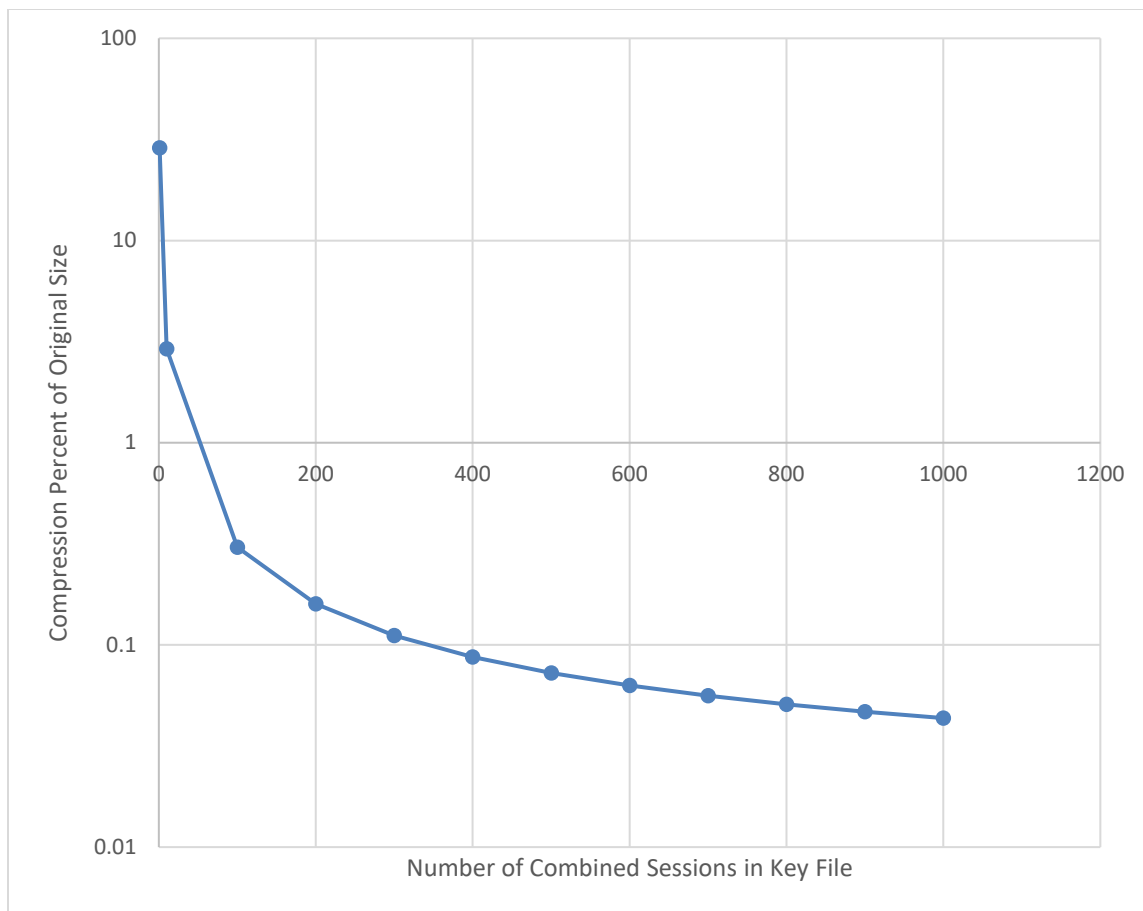


Figure 20. 7zip Compression Results versus Number of Sessions (Y-Axis Log Scaled)

The scaling shown in Figure 20 depicts combined session files varying in length from 1 to 1000 sessions to demonstrate the dramatic benefits of the 7zip compression as the number of keys collected increases.

A comparison of the compression techniques utilized versus the packet capture size is shown in Table 5.

Table 5. Compressed Key Files versus Packet Capture Size Ratio

Sessions	Capture Size (B)	Gzip File Size (B)	Gzip:Capture x10 ⁻⁵ ratio	7zip File Size (B)	7zip:Capture x10 ⁻⁵ ratio
1	15200000	19900	130.9211	17600	115.7895
10	152000000	199000	130.9211	17800	11.7105
100	1520000000	1940000	127.6316	18600	1.2237
200	3040000000	3880000	127.6316	19500	0.6414
300	4560000000	5820000	127.6316	20400	0.4474
400	6080000000	7760000	127.6316	21300	0.3503
500	7600000000	9700000	127.6316	22200	0.2921
600	9120000000	11640000	127.6316	23100	0.2533
700	10640000000	13580000	127.6316	24000	0.2256
800	12160000000	15520000	127.6316	24900	0.2048
900	13680000000	17460000	127.6316	25700	0.1879
1000	15200000000	19400000	127.6316	26600	0.1750
10000	1.52E+11	194000000	127.6316	265000	0.1743
100000	1.52E+12	1940000000	127.6316	2480000	0.1632
1000000	1.52E+13	19400000000	127.6316	24800000	0.1632

Isolating and exfiltrating the compressed key files shows immense benefit versus capturing the session from the client or server itself. With this proposed methodology, the packet capture would once again only have to capture the traffic on the side or at some point between the client and the server. For 7zip, the benefit plateau occurs at around 100 000 sessions worth of keys resulting in a 1.632×10^{-6} ratio to packet capture size savings. This benefit is also demonstrated further in Figure 21.

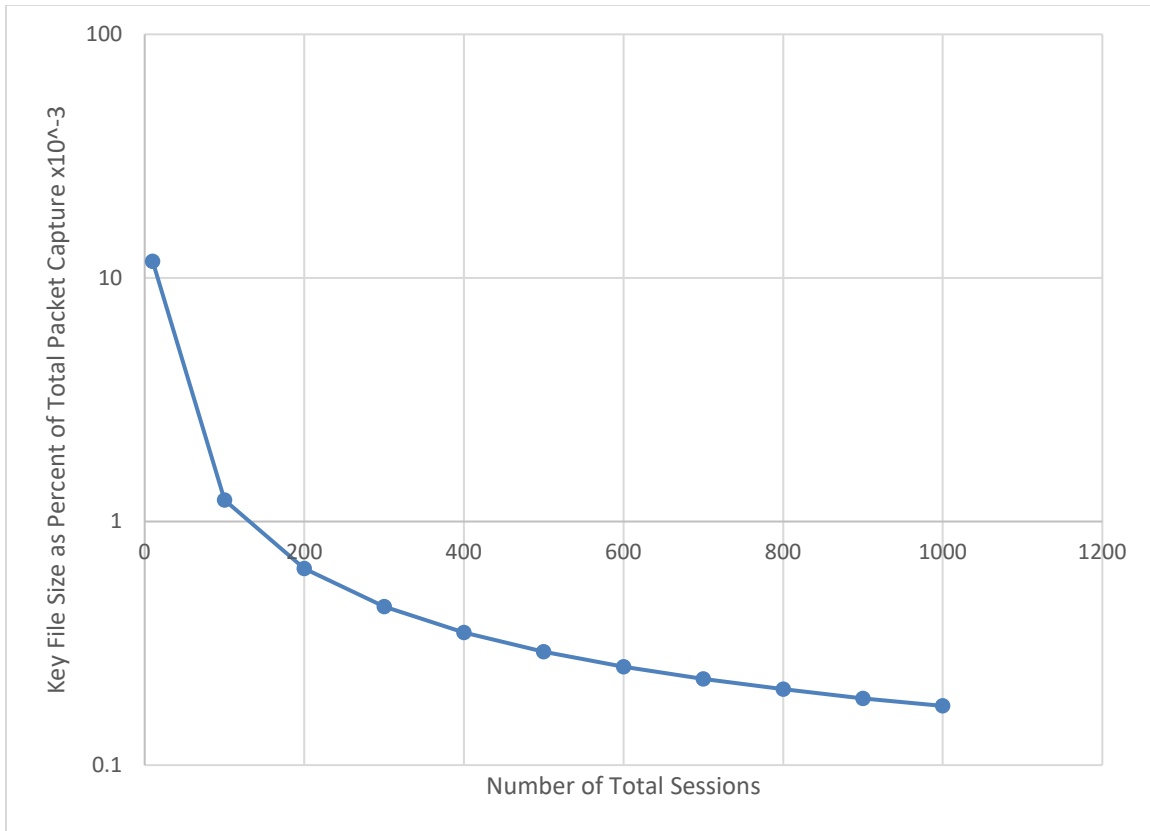


Figure 21. 7zip Compression versus Traffic Packet Capture Size Ratio

Lastly, a comparison of the various file types described so far versus how they change with increasing number of sessions is shown in Figure 22.

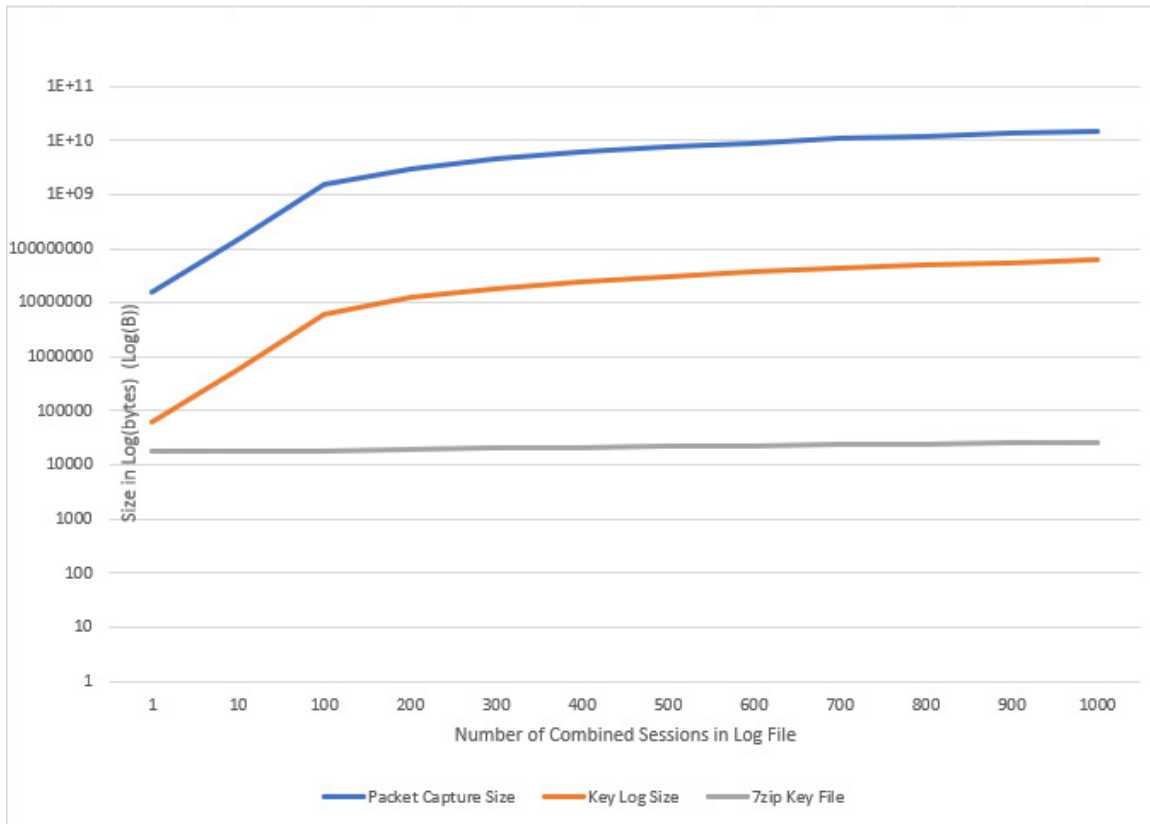


Figure 22. Packet Capture, Key Log, and 7zip Compressed File Size Scaling with Number of Sessions

Figure 22 shows the great depiction that as the number of sessions increases on the hundreds scale, the compressed size of the key file increases at a much slower rate.

Overall, a compressed key log file is many orders of magnitude smaller than the packet capture and original key log file. Reducing the size of required files for future decryption will prove to be advantageous in many aspects.

E. GMAIL EMAIL SESSIONS

The last experiment evaluates the real-life variability of 12 different Gmail email account sessions. All 12 accounts were previously setup and only used for research purposes. A summary of the data is shown in Table 6.

Table 6. Captured Gmail Email Session Keys

Account Reference Number	Keys Captured	Quadruples
1	364	91
2	364	91
3	384	96
4	368	92
5	364	91
6	344	86
7	332	83
8	368	92
9	356	89
10	348	87
11	372	93
12	356	89
Mean	359.75	89.94
Standard Deviation	13.27	3.32

Each session was collected over the course of 1 minute and 10 seconds and consisted of navigating to the Gmail homepage, logging in, viewing 2 emails, and sending 2 emails. When considering the keys as a data stream, they behave as a quadruple for encryption and decryption, as described in Section E, Chapter II. The total number of keys captured and the resulting quadruples occur with a relatively low variability distribution, although the sample size was quite limited.

The Jarque-Bera test was performed to evaluate whether the distribution of Gmail email keys was normal or not. Using MATLAB `jbttest` function [23] resulted in an h-value of 0, which indicates that the test failed to reject the null hypothesis (in that the distribution is not normal). The MATLAB code and output is shown in Appendix B.

Additionally, testing of the keys captured over time was performed for a personal Gmail email account. The results are depicted in Table 7.

Table 7. Gmail Email Session Keys Captured over Time

Trial	1 Minute Quadruples	5 Minute Quadruples	1<t<5 Quadruples per Minute	10 Minute Quadruples	5<t<10 Quadruples per Minute
1	68	94	6.5	111	3.4
2	69	88	4.75	106	3.6
3	65	93	7	108	3
4	66	77	2.75	91	2.8
5	68	93	6.25	105	2.4
6	66	85	4.75	103	3.6
7	65	86	5.25	100	2.8
8	70	93	5.75	107	2.8
9	64	88	6	102	2.8
10	65	88	5.75	106	3.6
11	66	87	5.25	105	3.6
12	63	84	5.25	102	3.6

Mean	66.22	87.87	5.31	103.72	3.14
Standard Deviation	2.01	4.67	1.04	4.81	0.42

The longer the Gmail email session is left open, the more keys will be generated. After the email session has been established and loaded, without opening or sending emails, the rate at which new quadruples are created continues to decrease. It should be noted that if a new email was received during these trials or another Gmail feature was activated, such as Gmail Chat, the quadruples created would dramatically increase. The rate of key quadruples generated over time is depicted in Figure 23.

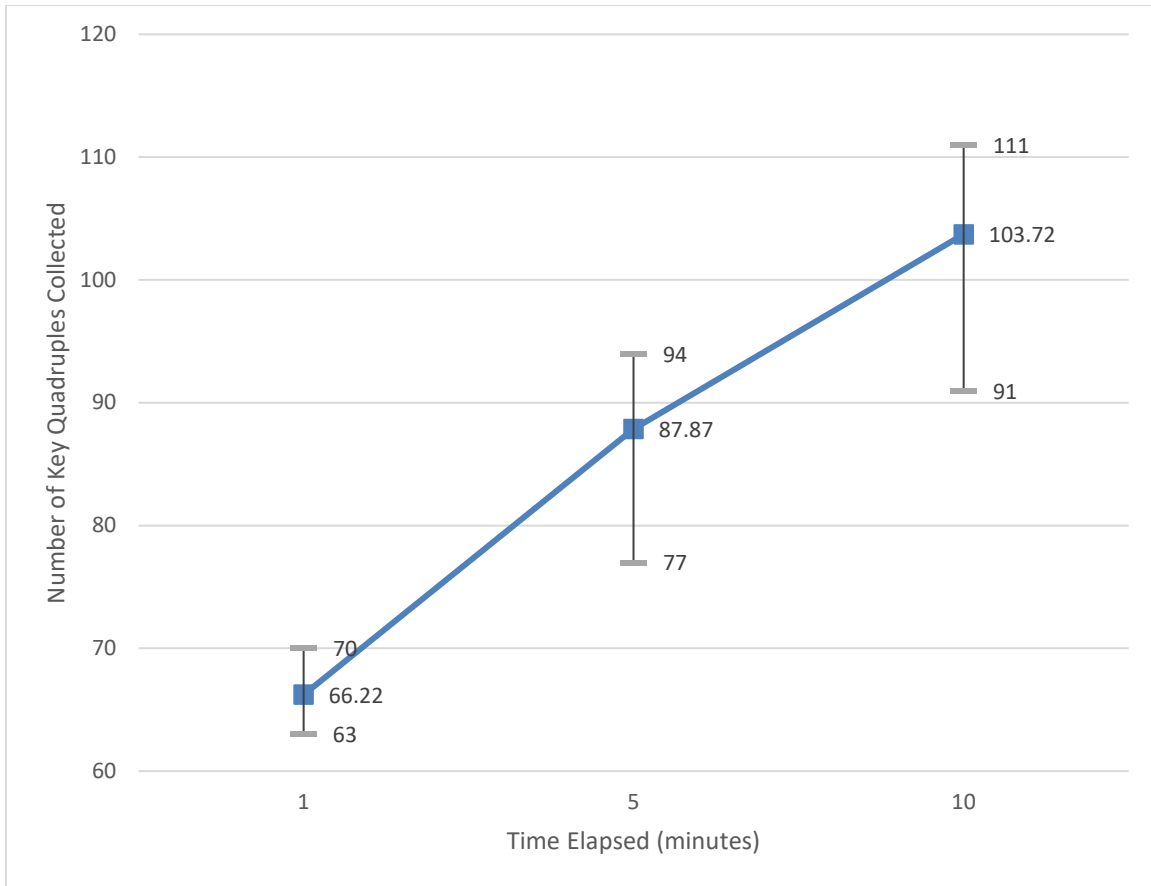


Figure 23. Key Quadruples Collected in Gmail Sessions Indicating Maximum, Average and Minimum for Each Measure of Time Duration

The Jarque-Bera test was performed once again to test the data for a normal distribution over each time sample. The default test was successful at the 5% significance level for the 1- and 5-minute samples, but the test was only successful at the 2% significance level for the 10-minute samples. This was due to trial number 4 being a significant outlier in the data. With this outlier omitted, the default 10-minute sample test would indicate normal.

This chapter presented the experiments of collecting and compressing a variety of TLS keys was explored. The use of the keys for decryption was also successful. In the next chapter, contributions and areas for future work will be presented.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS

Throughout this thesis, the security contributions and challenges in employing TLS 1.3 have been discussed and analyzed. Both the client and the server SSLKEYLOGFILE are valuable pieces of data and fortunately only one side is required for successful communication decryption.

From the blue team perspective, employing these methods of lightweight key logging on a high-value asset would contribute to the defense in depth strategy of protection. The keys could be transferred in real time to both an IDS and IPS to decrypt and analyze the live traffic whereas previously the TLS 1.3 data connections could not be decrypted. The methods discussed would not be resource intensive once implemented and would contribute greatly towards best protecting the asset.

For the red teams, the stealthy, lightweight exfiltration of compressed TLS 1.3 key files to decrypt MotS traffic passively from an asset would prove fruitful in a variety of areas. Not only would this semi-passive monitoring produce invaluable communications intelligence, but files that are transferred to and from the server would also be passively collected. Achieving these similar end-goals by a red team would otherwise be impossible or easily detectable.

A. CONTRIBUTIONS

Two significant contributions can be taken away from this thesis. First, from a defensive or blue team perspective, TLS 1.3 in network traffic can still be processed and analyzed where previous methods of inspection have been removed. The resources required to defend TLS 1.3 traffic will be more costly, but they can be reduced by using compression methods. Second, from an offensive or red team perspective, TLS 1.3 protocol security is not perfect. Asset access and subsequent stealthy exfiltration of the cryptographic keys can be achieved by actors with enough time and resources.

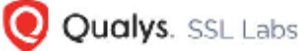
B. FUTURE WORK

There are numerous development and experiment opportunities that can be built upon this thesis research. For the blue team operators, developing an agent tool that would log the TLS keys and periodically forward them to an IPS or IDS solution would be useful. Development would continue on the IPS or IDS to integrate the TLS keys with the associated encrypted traffic to enable the decryption and follow-on processing.

The red team developers could build a stealthy agent that would reside on a host, log the TLS keys, periodically compress the file, and then export the key file to a destination when desired or on a periodic basis. The agent would be customizable such that it would only log the desired keys for specific communications that occur on the high-value asset. Both of these development opportunities could be performed on a test network and further refined for a future customer.

APPENDIX A. QUALYS SSL LABS REPORT

The Qualys SSL Labs Report assesses various web server functionalities, capabilities, and security features. Specifically, this report shows that only TLS 1.3 is configured for the experiment web server and also displays its available cipher suites.


[Home](#)
[Projects](#)
[Qualys Free Trial](#)
[Contact](#)

You are here: [Home](#) > [Projects](#) > [SSL Server Test](#) > anderthesis.net

SSL Report: anderthesis.net (52.32.70.128)

Assessed on: Fri, 12 May 2023 05:33:12 UTC | [Link](#) | [Clear cache](#) [Scan Another »](#)

Summary

Overall Rating

A

Certificate

Protocol Support

Key Exchange

Cipher Strength

0 20 40 60 80 100

Visit our [documentation page](#) for more information, configuration guides, and books. Known issues are documented [here](#).

This server supports TLS 1.3.

Certificate #1: RSA 2048 bits (SHA256withRSA)



Server Key and Certificate #1

Subject	anderthesis.net Fingerprint (SHA256): ea5662f1607fb60cd2945e668c30915d59b70cd909e3c76b761d6e Pin SHA256: y4E:8I4ZvE:YqH:bouUGpFp2U9nb07UTy6R:dwew
Common names	anderthesis.net
Alternative names	anderthesis.net
Serial Number	04d9b4eefee5b2b908d2e545a077b4e4f8
Valid from	Thu, 30 Mar 2023 15:33:36 UTC
Valid until	Wed, 28 Jun 2023 15:33:35 UTC (expires in 1 month and 16 days)
Key	RSA 2048 bits (a 65537)
Weak key (Debian)	No
Issuer	IC3 NA: http://3.11enc.org/
Signature algorithm	SHA256withRSA
Extended Validation	No
Certificate Transparency	Yes (certificate)
OCSP Must Staple	No
Revocation information	OCSP OCSP: http://3.11enc.org
Revocation status	Good (not revoked)
DNSCAA	No (more info)
Trusted	Yes Mozilla Apple Android Java Windows



Additional Certificates (if supplied)

Certificates provided	3 (4096 bytes)
Chain issues	None

<https://www.ssllabs.com/ssltest/analyze.html?d=anderthesis.net> 1/3

Additional Certificates (if supplied)

#2

Subject	R3 Fingerprint SHA256: 67ade116b5220ae61b0f585660c04c2ae56966079665672a3c7d737613d8 P in SHA256: jQUTb0gwd1TH9umWbFe0Gpg621gT3PwPNS0+
Valid until	Mon, 15 Sep 2025 16:00:00 UTC (expires in 2 years and 4 months)
Key	RSA 2048 bits (e 65537)
Issuer	ISRG Root X1
Signature algorithm	SHA256 with RSA

#3

Subject	ISRG Root X1 Fingerprint SHA256: 6d59b265b1d3b374763dbd465025e1f5b6c4c25666e47c7f1c04f P in SHA256: C5+q2TcdVnswC8kRPOdWABXhCwja0wHf68H+
Valid until	Mon, 30 Sep 2024 16:14:03 UTC (expires in 1 year and 4 months)
Key	RSA 4096 bits (e 65537)
Issuer	DST Root CA X3
Signature algorithm	SHA256 with RSA



Certification Paths


[Click here to expand](#)

Configuration



Protocols

TLS 1.3	Yes
TLS 1.2	No
TLS 1.1	No
TLS 1.0	No
SSL 3	No
SSL 2	No



Cipher Suites

TLS 1.3 (server has no preferences)

TLS_AES_128_GCM_SHA256 (0x1301)	ECDH x25519 (eq. 3072 bits RSA)	PS	128
TLS_AES_256_GCM_SHA384 (0x1302)	ECDH x25519 (eq. 3072 bits RSA)	PS	256
TLS_CHACHA20_POLY1305_SHA256 (0x1303)	ECDH x25519 (eq. 3072 bits RSA)	PS	256



Handshake Simulation

Android 8.1	- TLS 1.3	TLS_CHACHA20_POLY1305_SHA256	ECDH x25519	PS
Android 9.0	- TLS 1.3	TLS_CHACHA20_POLY1305_SHA256	ECDH x25519	PS
Chrome 80 / Win 7 R	- Server sent fatal alert: protocol_version			
Chrome 70 / Win 10	- TLS 1.3	TLS_AES_128_GCM_SHA256	ECDH x25519	PS
Chrome 80 / Win 10 R	- TLS 1.3	TLS_AES_128_GCM_SHA256	ECDH x25519	PS
Firefox 62 / Win 7 R	- Server sent fatal alert: protocol_version			
Firefox 73 / Win 10 R	- TLS 1.3	TLS_AES_128_GCM_SHA256	ECDH x25519	PS
Java 11.0.3	- TLS 1.3	TLS_AES_128_GCM_SHA256	ECDH x25519	PS
Java 12.0.1	- TLS 1.3	TLS_AES_128_GCM_SHA256	ECDH x25519	PS
OpenSSL 1.1.0k R	- Server sent fatal alert: protocol_version			

Handshake Simulation

OpenSSL 1.1.1f R	-	TLS 1.3	TLS_AES_256_GCM_SHA384 ECDHE x25519 PS
Safari 12.1.2/MacOS 10.14.8 R	-	TLS 1.3	TLS_CHACHA20_POLY1305_SHA256 ECDHE x25519 PS
Safari 12.1.1/iOS 12.1.1 R	-	TLS 1.3	TLS_CHACHA20_POLY1305_SHA256 ECDHE x25519 PS

Not simulated clients (Protocol mismatch)

[Click here to expand](#)

- (1) Clients that do not support Forward Secrecy (FS) are excluded when determining support for it.
 (2) No support for virtual SSL hosting (SNI). Connects to the default site if the server uses SNI.
 (3) Only first connection attempt simulated. Browsers sometimes retry with a lower protocol version.
 (R) Denotes a reference browser or client, with which we expect better effective security.
 (All) We use defaults, but some platforms do not use their best protocols and features (e.g., Java 6 & 7, older IE).
 (All) Certificate trust is not checked in handshake simulation, we only perform TLS handshake.



Protocol Details

	Unable to perform this test due to an internal error. (1) For a better understanding of this test, please read this longer explanation . (2) Key usage data kindly provided by the Censys network search engine; original DROWN website here . (3) Censys data is only indicative of possible key and certificate reuse; possibly out-of-date and not complete. INTERNAL ERROR: connect timed out INTERNAL ERROR: connect timed out	
DROWN		
ROBOT (vulnerability)	No	(more info)
Forward Secrecy	Yes (with most browsers)	ROBUST (more info)
ALPN	Yes	http/1.1
Session resumption (tickets)	No	
OCSP stapling	No	
Strict Transport Security (HSTS)	No	
HSTS Preloading	Not in:	Chrome Edge Firefox IE
Public Key Pinning (HPKP)	No	(more info)
Public Key Pinning Report-Only	No	
Public Key Pinning (Static)	Unknown	
Supported Named Groups	x25519, secp256r1, secp384r1, secp521r1, ffdhe2048, ffdhe3072 (server preferred order)	
SSL 2 handshake compatibility	No	
0-RTT enabled	No	



HTTP Requests

1 <https://andertneis.net/> (HTTP/1.1 200 OK)

Miscellaneous

Test date	Fri, 12 May 2023 05:31:51 UTC
Test duration	81.358 seconds
HTTP status code	200
HTTP server signature	Apache/2.4.52 (Ubuntu)
Server hostname	ec2-52-32-70-128.us-west-2.compute.amazonaws.com

SSL Report v2.1.10

Copyright © 2009-2023 [Qualys, Inc.](#) All Rights Reserved.[Try Qualys for free!](#) Experience the award-winning [Qualys Cloud Platform](#) and the entire collection of [Qualys Cloud Apps](#), including [certificate security](#) solutions.[Terms and Conditions](#)

APPENDIX B. MATLAB CODE AND OUTPUT

MATLAB was used to perform the Jarque-Bera test for the sample data. The default test is performed at the 5% significance level but can be adjusted by including a second parameter in the `jbtest` function.

```
% Gmail email keys captured for 12 different accounts

>> gmail_test = [364 364 384 368 364 344 332 368 356 348
372 356]

gmail_test =

    364    364    384    368    364    344    332    368    356
348    372    356

>> h = jbtest(gmail_test)

h =

    0

% Personal Gmail email keys captured over time
% One minute, 12 trials

>> min_one = [272    276    260    264    272    264    260    280    256
260    264    252]

min_one =
```

```
    272    276    260    264    272    264    260    280    256
260    264    252
```

```
>> h = jbstest(min_one)
```

```
h =
```

```
    0
```

```
% Five minutes, 12 trials
```

```
>> min_five = [376    352    372    308    372    340    344    372    352
               352    348    336]
```

```
min_five =
```

```
    376    352    372    308    372    340    344    372    352
352    348    336
```

```
>> h = jbstest(min_five)
```

```
h =
```

```
    0
```

```
% Ten minutes, 12 trials
```

```
>> min_ten = [444  424  432  364  420  412  400  428  408
               424  420  408]
```

```
min_ten =
```

```
    444    424    432    364    420    412    400    428    408
424    420    408
```

```
>> h = jbtest(min_ten, 0.03)
```

```
h =
```

```
    1
```

```
>> h = jbtest(min_ten, 0.02)
```

```
h =
```

```
    0
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

- [1] “The Transport Layer Security (TLS) protocol version 1.3,” Request for Comments, RFC 8446, August 2018. Available: <https://datatracker.ietf.org/doc/rfc8446/>
- [2] Warburton, David. “The 2021 TLS telemetry report.” F5 Labs, blog, October 20, 2021. Available: <https://www.f5.com/labs/articles/threat-intelligence/the-2021-tls-telemetry-report>
- [3] E. Tauchman, “Are you red team or blue team? How your skills fit into a cybersecurity career,” CompTIA, blog, 7 Apr. 2023. Available: <https://www.comptia.org/blog/cybersecurity-red-team-or-blue-team>
- [4] “SSL enabling forward secrecy,” DigiCert, May 2023. Available: <https://www.digicert.com/kb/ssl-support/ssl-enabling-perfect-forward-secrecy.htm>
- [5] J. Davies, “A walkthrough of a TLS 1.3 handshake,” Command Line Fanatic, blog, June 25, 2019. Available: <http://commandlinefanatic.com/cgi-bin/showarticle.cgi?article=art080>
- [6] Lee, Hyunwoo et al. “TLS 1.3 in practice: How TLS 1.3 contributes to the internet,” Proceedings of the Web Conference 2021, ACM, 2021, pp. 70–79. Available: <https://doi.org/10.1145/3442381.3450057>
- [7] Laštovička, Martin et al. “Using TLS fingerprints for OS identification in encrypted traffic,” NOMS 2020 – 2020 IEEE/IFIP Network Operations and Management Symposium, 2020, pp. 1–6. Available: <https://doi.org/10.1109/NOMS47738.2020.9110319>
- [8] de Carnavalet, Xavier de Carné, and Paul C. van Oorschot. “A survey and analysis of TLS interception mechanisms and motivations,” ACM Computing Surveys, Jan. 2023. Available: <https://doi.org/10.1145/3580522>
- [9] “What is SSL, TLS and HTTPS?,” DigiCert, May 2023. Available: <https://www.websecurity.digicert.com/security-topics/what-is-ssl-tls-https>
- [10] “The evolution of SSL and TLS,” DigiCert, May 2023. Available: <https://www.digicert.com/blog/evolution-of-ssl>
- [11] “Prohibiting Secure Sockets Layer (SSL) version 2.0,” Request for Comments, RFC 6176, March 2011. Available: <https://datatracker.ietf.org/doc/rfc6176/>
- [12] “Deprecating Secure Sockets Layer version 3.0,” Request for Comments, RFC 7568, June 2015. Available: <https://datatracker.ietf.org/doc/rfc7568/>

- [13] “Deprecating TLS 1.0 and TLS 1.1,” Request for Comments, RFC 8996, March 2021. Available: <https://datatracker.ietf.org/doc/rfc8996/>
- [14] “The Transport Layer Security (TLS) protocol version 1.2,” Request for Comments, RFC 5246, August 2008. Available: <https://datatracker.ietf.org/doc/rfc5246/>
- [15] “Inspect TLS traffic,” Deep Security, May 2023. Available: https://help.deepsecurity.trendmicro.com/20_0/on-premise/intrusion-prevention-ssl-traffic.html#Supporte
- [16] “How to decrypt SSL using Chrome or Firefox and Wireshark in windows,” Palo Alto Networks, Customer Support Portal, May 2023. Available: <https://knowledgebase.paloaltonetworks.com/KCSArticleDetail?id=kA14u000000HB8gCAG>
- [17] “Index of manpages,” The Tcpdump Group, May 2023. Available: <https://www.tcpdump.org/manpages/>
- [18] “TLS,” Wireshark, May 2023. Available: <https://wiki.wireshark.org/TLS>
- [19] “April 2023 web server survey,” Netcraft, Web Server Survey, April 2023. Available: <https://news.netcraft.com/archives/category/web-server-survey/>
- [20] Lekensteyn, “Extracting openssl pre-master secret from apache2,” Information Security Stack Exchange, blog, August 18, 2019. Available: <https://security.stackexchange.com/questions/215358/extracting-openssl-pre-master-secret-from-apache2>
- [21] “GNU Gzip,” GNU Operating System, April 2022. Available: <https://www.gnu.org/software/gzip/manual/gzip.html>
- [22] I. Pavlov. 7-Zip Format, 2022. Available: <https://www.7-zip.org/7z.html>
- [23] “Jarque-Bera test – MATLAB jbtest,” May 2023. Available: <https://www.mathworks.com/help/stats/jbtest.html>

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California



DUDLEY KNOX LIBRARY

NAVAL POSTGRADUATE SCHOOL

WWW.NPS.EDU

WHERE SCIENCE MEETS THE ART OF WARFARE