TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JUHO NURMI

ANALYZING PRACTICAL COMMUNICATION SECURITY OF ANDROID VENDOR APPLICATIONS

Master of Science Thesis

Examiner: Professor Jarmo Harju
Examiner and topic approved by the Council of the Faculty of Computing and Electrical Engineering on October 8th, 2014

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
Nurmi, Juho: Analyzing practical communication security of Android vendor applications
Master of Science Thesis, 87 pages, 9 Appendix pages
December 2014
Major: Communication Networks and Protocols
Examiner: Professor Jarmo Harju
Keywords: Android, Information security, mobile device, security weakness, MITM, SSL, TLS, certificate

The development of mobile devices and the new personalized services have gone to the point, where users do not alone control their data. While the devices are in constant communication with the cloud services the user's data and the data of the user move ever more to the services providers' cloud services. Little is known about how and how well service providers protect the users' information.

The work studies two biggest western Android based ecosystems, Google's and Amazon's, own applications' practical security in the communication process. The aim is to identify all mechanisms used to protect the information that is communicated with the Android device. The study used one device from Amazon and Google, and the application market was chosen from both service providers for in-depth study. The applications were selected on the basis that they must provide same service in order to make the comparison possible. In practice, the applications and devices were studied by performing active and passive Man-in-the-middle (MITM) attacks in network laboratory. The communications were intercepted and analysed afterwards.

Both vendors relied heavily on SSL/TLS protocol. Also in common was the usage, roles and acquirement of authorization tokens. Amazon's client applications were noticed to use digital signatures. The biggest difference between the market applications was that Google required authentication when buying an application, while Amazon did not require it. During the same authentication Google sent user's password in plaintext inside the TLS connection. During the less frequently happening registration of the user's Google account to the device the user's password is sent instead encrypted inside the TLS connection.

An active MITM attack was performed on the Google device and account to demonstrate what the attacker can do in practice, when SSL/TLS connection is compromised. With manipulating traffic and intercepting authorization tokens the attacker is able to spy the victim and access to nearly all the victim's Google data for the present. In addition, the attacker can "force" the victim to register herself again to the Android device and the attacker can use the victim's intercepted encrypted password to add the victim's Google account to her own device.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
Nurmi, Juho: Analyzing practical communication security of Android vendor applications
Diplomityö, 87 sivua, 9 liitesivua
Joulukuu 2014
Pääaine: Tietoliikenneverkot ja -protokollat
Tarkastaja: professori Jarmo Harju
Avainsanat: Android, tietoturvallisuus, mobiililaite, tietoturvaheikkous, välimieshyökkäys, SSL, TLS, sertifikaatti

Älylaitteiden kehitys ja palveluntarjoajien uudet henkilökohtaisemmat palvelut ovat johtaneet siihen, että käyttäjät eivät enää yksin hallitse tietojaan. Laitteiden ollessa jatkuvassa yhteydessä pilvipalveluihin käyttäjien tiedot ja tietoa heistä siirtyy yhä enemmän palveluntarjoajien pilvipalveluihin. Siitä miten ja kuinka hyvin palveluntarjoajat suojaavat käyttäjien tietoja, tiedetään hyvin vähän.

Työssä tutkitaan kahden suurimman länsimaisen Android-pohjaisen ekosysteemin, Googlen ja Amazonin, omien ohjelmien tietoliikenteen tietoturvallisuutta käytännössä. Tavoitteena oli selvittää kaikki ne mekanismit, joilla Android-laitteesta lähtevää tietoliikennettä suojataan. Tutkimuksessa käytettiin Amazonilta ja Googlelta yhtä laitetta, sekä molemmilta valittiin sovelluskauppa tarkempaa tutkimusta varten. Applikaatiot valittiin sillä perusteella, että niiden piti tuottaa käyttäjälle samaa palvelua vertailun mahdollistamiseksi. Laitteita ja applikaatioita tutkittiin käytännössä suorittamalla niille sekä passiivisia että aktiivisia välimieshyökkäyksiä verkkolaboratoriossa. Tietoliikenne kaapattiin talteen ja analysoitiin jälkikäteen.

Molempien valmistajien ohjelmien, sekä laitteiden tietoturvan havaittiin nojautuvan vahvasti kuljetuskerroksen tunnelointiprotokollaan (SSL/TLS). Lisäksi yhteistä oli auktorisointitokenien käyttö, niiden roolit sekä hakuprosessi. Amazonin asiakasapplikaation havaittiin käyttävän digitaalisia allekirjoituksia. Sovelluskauppojen isoimmaksi eroksi havaittiin Googlen vaativan käyttäjää tunnistautumaan ostaessaan applikaatiota, mitä Amazonilla ei vaadittu. Googlella samaisen tunnistautumisen yhteydessä havaittiin käyttäjän salasanan välittyvän selkokielisenä TLS-tunnelin sisällä. Harvoin tapahtuvan Google-käyttäjätunnuksen laitteeseen rekisteröimisen yhteydessä käyttäjän salasana sen sijaan välitetään salattuna TLS-tunnelin sisällä.

Googlen laitteelle ja tunnukselle suoritettiin vielä aktiivinen välimieshyökkäys demonstroimaan mitä hyökkääjä voi tehdä käytännössä, kun TLS-protokolla pettää. Manipuloimalla liikennettä ja kaappaamalla auktorisointitokeneita hyökkääjän havaittiin pystyvän vakoilemaan uhria ja pääsevän toistaiseksi käsiksi etänä lähes kaikkiin uhrin Google-tunnuksen tietoihin. Lisäksi hyökkääjä pystyy ”pakottamaan” uhrin rekisteröitymään Android-laitteelleen uudestaan, minkä yhteydessä hyökkääjä voi käyttää uhrin kaapattua salattua salasanaa uhrin Google-tunnuksen lisäämiseen omalle laitteelleen.

PREFACE

This thesis was written as a master's thesis for Tampere University of Technology (TUT) in department of Pervasive Computing. Thesis was examined by Professor Jarmo Harju from TUT. When I started to work on this thesis, I knew basically nothing of Android and especially its security, so the whole journey was very interesting. Everywhere you looked at, you saw something new and you had (urge) to understand it.

There are a lot of people I really have to thank for: I would like to thank Professor Jarmo Harju and Nokia Oyj for offering me this interesting topic. I would also like to thank Mika Anttila, Sami Majaniemi and Anssi Juvonen from Nokia for guiding me and giving me a flying start. In addition, I thank Joona Kannisto and Billy Bob Brumley for helping me with cryptology related questions and my colleagues for enduring my constant bombardment of questions.

Special thanks goes to Tanel for enduring in the trenches with me everything that TUT threw at us over the years. Most of all, I want thank my family for their support and encouragement during my studies.

Juho Nurmi,
11 November 2014

CONTENTS

ABBREVIATIONS

APK	Android Application Package
AWS	Amazon Web Services
CA	Certificate Authority
CN	Common Name
CBC	Cipher Block Chaining
DNS	Domain Name Service
GA	Google Auth
GLS	Google Login Service
GMS	Google Mobile Services
GPS	Google Play Services
HMAC	Hash-based Message Authentication Code
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IANA	Internet Assigned Numbers Authority
IETF	Internet Engineering Task Force
IKE	Internet Key Exchange
ITU	International Telecommunication Union
ITU-T	ITU Telecommunication Standardization Sector
IV	Initialization Vector
JSON	JavaScript Object Notation
MAC	Message Authentication Code
MITM	Man-In-The-Middle
OHA	Open Handset Alliance
OS	Operating System
PKI	Public Key Infrastructure
RA	Registration Authority
SAN	SubjectAltName
SNI	Server Name Indication
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
URI	Uniform Resource Identifier
XML	Extensible Markup Language

1 INTRODUCTION

Mobile devices, such as smartphones and tablets, have become an integrated part of our everyday life and are used now by all age groups. These devices are used for entertainment, communication and business, and hold data of great value, from personal and business perspective. Service providers and vendors have started to offer more personal services to users and the location of the user's data has blurred between the user's device and the service provider's cloud. This has set a requirement for the device to be constantly communicating with the cloud in order to get updates, synchronize data, etc. Since the devices are constantly communicating and accessing users' data, it is not enough that the data is secured just during the transit. Also, the access to the users' data in the cloud has to be secured.

According to Stallings [1] the threats that any data faces in the web could be classified in active and passive attacks. Passive attacks include eavesdropping on the traffic and the attacker gaining restricted information. Active attacks include message tampering during the transit and impersonating another user. [1] These attacks break the basic security properties, such as, integrity, confidentiality and authentication.

Android is an open source and customizable operating system (OS) for mobile devices that has become leading smartphone platform early in 2010 [2]. The two biggest Android versions in the western world, at the time of writing in fall of 2014, are Google Android (a.k.a. Android Open Handset Alliance (OHA)) and Amazon Fire OS [3]. This thesis concentrates on how Google and Amazon have secured the communications between their applications in the mobile devices and services in the cloud.

The aim is to identify what security mechanisms the vendors use in practice in the communication channel and analyse findings. Internal security mechanisms in the device and adequacy of the found mechanisms in the considered context are out of scope in this thesis.

The work was done by examining one device from both vendors and devices were studied in a networking laboratory. The study was done while performing a Man-in-the-middle (MITM) attack to the devices. Market application from both vendors (Amazon Shop and Google Play Store) was selected for closer inspection. These applications were selected because they execute common functionality and are therefore comparable. The communications were intercepted and in certain cases manipulated. Also, the communications were logged and analysed manually afterwards.

Based on the findings an attack is performed to the Google Android device and the author's Google account to demonstrate, what the attacker is capable of doing when SSL/TLS protocol fails. A proposal to counter the threat is given.

The thesis is divided into 9 chapters. After the introduction, in Chapter 2 the basic security mechanisms and protocols are covered. Chapter 3 presents the test environment, the used software and how the devices and applications were observed to communicate. Next, in Chapter 4 certain Google’s and Amazon’s specific services and protocols are covered for background information. Then, in Chapter 5 Google’s and in Chapter 6 Amazon’s results from the observations are presented in their respective chapters. This includes what tokens were found, how they are acquired and in-depth look to the market application. Chapter 7 describes what an attacker can do in practice when SSL/TLS protocol fails in a Google Android device and proposes ideas for prevention. In Chapter 8 Google’s and Amazon’s found security mechanisms are summarized and compared. And finally, in Chapter 9 conclusions are presented.

2 SECURITY BASICS

Chapter starts by covering the most basic information security terms. Next, it moves to present tokens. Then basic overview is given to cryptographic functions, starting from cryptographic hash function and then moving to digital signatures. Next, it covers public key infrastructure and moves then to cover the SSL/TLS protocol. The chapter ends by presenting the man-in-the-middle attack.

2.1 Basic terms

Authentication is a “process of verifying a claim that a system entity or system resource has a certain attribute value” [4]. In other words, it is a process of confirming, for example, the identity of a person by her identity documents. *Authorization* is a process of granting approval to a system entity to access a system resource [4]. Google’s own terminology in their documentation might be confusing, because they use term *Auth* to address both, authentication and authorization [5]. *Integrity* is guarding that data has not been changed, destroyed, or lost in an unauthorized event or in an accident [4].

In *symmetric cryptography* the same key is used to encrypt and decrypt the message. In *asymmetric cryptography* (a.k.a. public key cryptography) a pair of keys (public and private key) is used for different cryptographic operation (e.g. encryption and decryption, or signature creation and verification). [4]

Nonce is defined by NIST [6] as “A time-varying value that has at most a negligible chance of repeating, e.g., a random value that is generated anew for each use, a timestamp, a sequence number, or some combination of these” [6]. Nonces are used, for example, in an initialization vector (IV) of a cipher block chaining (CBC) mode and in internet key exchange (IKE) [1] and could be used to assure the recipient that a message is fresh and not an old message that an attacker has observed [7].

2.2 Token

According to RFC 4949 *Internet Security Glossary, Version 2* the token is an overloaded term in the computing literature. The term is used for describing, for example, a physical device that is used to store cryptographic information or a data object that is used to verify the identity in the authentication process. This thesis uses the definition used for *capability token*, whenever token is mentioned. Capability token is a (usually an unforgeable) data object that gives the holder or bearer the right to access a system resource. Possession of the token is accepted by a system as proof that the holder has been authorized to access the resource indicated by the token. [4]

Tokens were observed to be exchanged between mobile devices and service providers in 3 different ways in HTTP messages: in HTTP header (Authorization or custom HTTP header), HTTP cookies and in message body (e.g. plaintext, encoded in base64URL, JSON object, etc.).

```
POST /gsync/sub HTTP/1.1
Authorization: GoogleLogin auth=DQAAAPoAAADt0lDtAdd7cqYdK0nb0Zeu2m-
82WIyPMWrVc2fJh4l8rg_toKPVVcGhH97DE93faN0NrM7Wp4yAL9nA_Fp1-7i4Nsl4wvaC5oq6iPUa
3-X187awvlkEcIHwiuPghbuq8ch9uwCkxAYRUYEOEASZlANIH66tgp0Kd5z8hKmmJg96FwbBHVLY6f
EJ_llBA5VUDCdIpx3K-30ytk9lN7OtVCb4p5CpUoTiI-QVO3aGVBG6a-2UrS8oFchnAqJMB46VYTiq
AGfsKpkSSnLvqQN5UcF6go82ose5pP9G6eYlLJCowF7LxqR96izQ1bKRA-SvlRgYQI3pNpHJGu5V0I
9NVxs
Content-Length: 471
Connection: Keep-Alive
```

Listing 2.1 Token provided in the Authorization HTTP header field.

```
POST /proxy/gmail/mail/g/?version=... HTTP/1.1
Connection: Keep-Alive
Cookie: GX=DQAAAPoAAADt0lDtAdd7cqYdK0nb0Zeu2m-82WIyPMWrVc2fJh4l8rg_toKPVVcGhH9
7DE93faN0NrM7Wp4yAL9nA_Fp1-7i4Nsl4wvaC5oq6iPUa3-X187awvlkEcIHwiuPghbuq8ch9uwCk
xAYRUYEOEASZlANIH66tgp0Kd5z8hKmmJg96FwbBHVLY6fEJ_llBA5VUDCdIpx3K-30ytk9lN7OtVC
b4p5CpUoTiI-QVO3aGVBG6a-2UrS8oFchnAqJMB46VYTiqAGfsKpkSSnLvqQN5UcF6go82ose5pP9G
6eYlLJCowF7LxqR96izQ1bKRA-SvlRgYQI3pNpHJGu5V0I9NVxs
Cookie2: $Version=1
```

Listing 2.2 Token provided in a HTTP cookie.

```
POST /auth HTTP/1.1
content-type: application/x-www-form-urlencoded

device_country=fi& ... &EncryptedPasswd=oauth2rt_1%2F406mny5jZzr7hLLS8RMwYUX0M
0okCkeBLRvA1c1ag3M
```

Listing 2.3 Token given in HTTP message body.

It should be noted that usage of observed HTTP headers (Authorization or custom), do not all the time strictly follow HTTP specifications. For example, the authentication scheme “GoogleLogin” in Listing 2.1 is not found in Internet assigned number authority (IANA) maintained HTTP authentication scheme registry [8]. Also, many observed custom headers started with prefix “X-“, which RFC 7231 advises not to use [9].

2.3 Cryptographic hash function

According to Stallings [1] the cryptographic hash function might be the most versatile cryptographic algorithm and it is widely used in security applications and Internet protocols. For example, cryptographic hash functions are used in message authentication, digital signatures, one-way password files, intrusion detection, virus detection and pseudorandom number generators. In general, hash function’s main role is to produce data integrity, because its characteristics provide a way to know whether or not data has changed. [1] In the simplest way hash function can be described as a function, which takes as an input a variable length message or data block and produces a fixed length output. Figure 2.1 presents a widely used hash function structure.

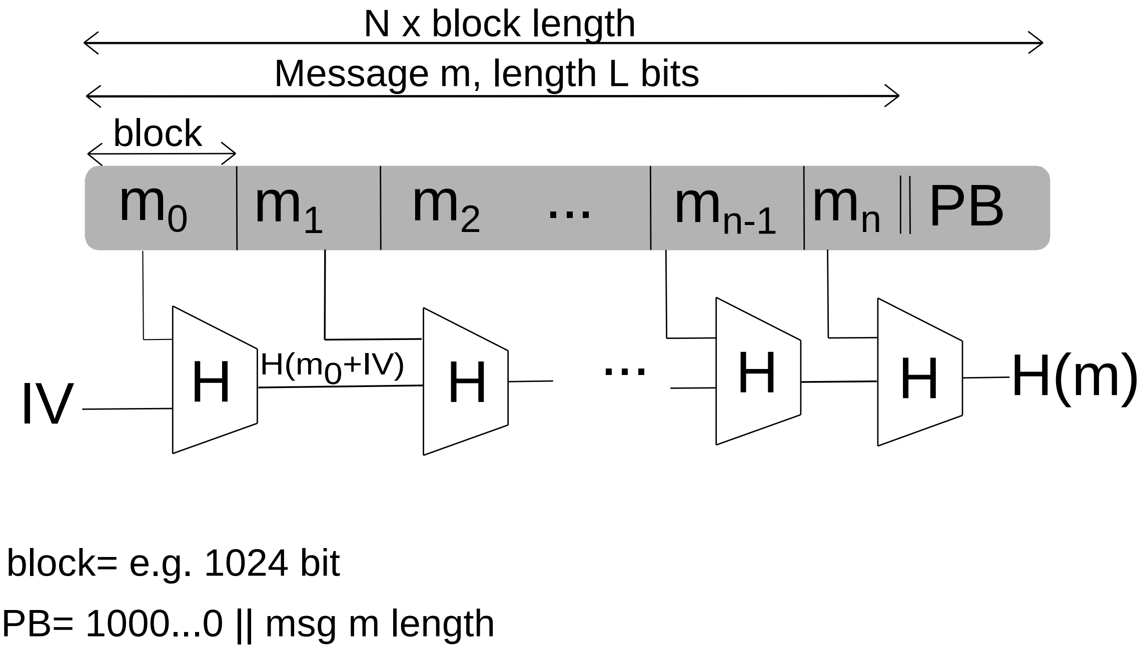


Figure 2.1 *Diagram of Merkle - Damgård structured cryptographic hash function.*

A hash function H produces a fixed-length hash value $H(m)$ or message digest from a variable length (L bits) message m, which is padded with padding block PB. Message m is divided to fixed length blocks (e.g. 1024 bits). In the Figure 2.1 the input for the hash function H starts with fixed value IV and message block m_0. The output $H(m_0 + IV)$ is then fed as input together with next message block m_1 to the hash function H. This is iterated until the final message block m_n, which is padded with padding block PB in order fill the final block to the fixed block length. If the message m length is a multiple of the fixed block length, then new block is concatenated consisting of just the padding block PB. The PB binary value starts with 1 followed by variable amount of zeroes and ends with fixed length (e.g. 64 bit or 128 bit) field containing the value of the message length L. The Merkle – Damgård type structure is used, for example, in MD5 and SHA-1 hash functions. [10]

Properties for a “good” hash function are that a large set of inputs produce evenly distributed and apparently random outputs. Also a change to one or any of the bits in the message M results in the output hash value $H(m)$ to change with a high probability. [1]

A cryptographic hash function has more strict requirements in order to be suitable for security applications. Stallings [1] defines a cryptographic hash function as an algorithm for which no attack is significantly more efficient than brute-force and therefore is computationally infeasible to find either:

- a data object mapping to a pre-specified hash result (the one-way property)
- two data objects that map to the same hash result (the collision-free property)

If an attack is found, for a cryptographic hash function, that breaks algorithm's one-way property or the collision-free property then that algorithm is considered cryptographically broken and is not suitable for use anymore. For example, in 2004, 2005 and 2008 weaknesses were found on the cryptographic hash function MD5 [11], which allowed an attacker to generate collisions [12]. In practice, this gave the attacker the ability to spoof SSL CA certificates that used MD5 signing algorithm [12].

2.4 MAC and HMAC

Message authentication code (MAC) provides message integrity (like cryptographic hash function) and authenticity of the sender. MAC algorithm requires the use of a secret key, which is used as an input together with the message. The secret key is shared with the recipient, who uses the key to generate the MAC and verify the message. [1] Figure 2.2 presents the basic usage of MAC.

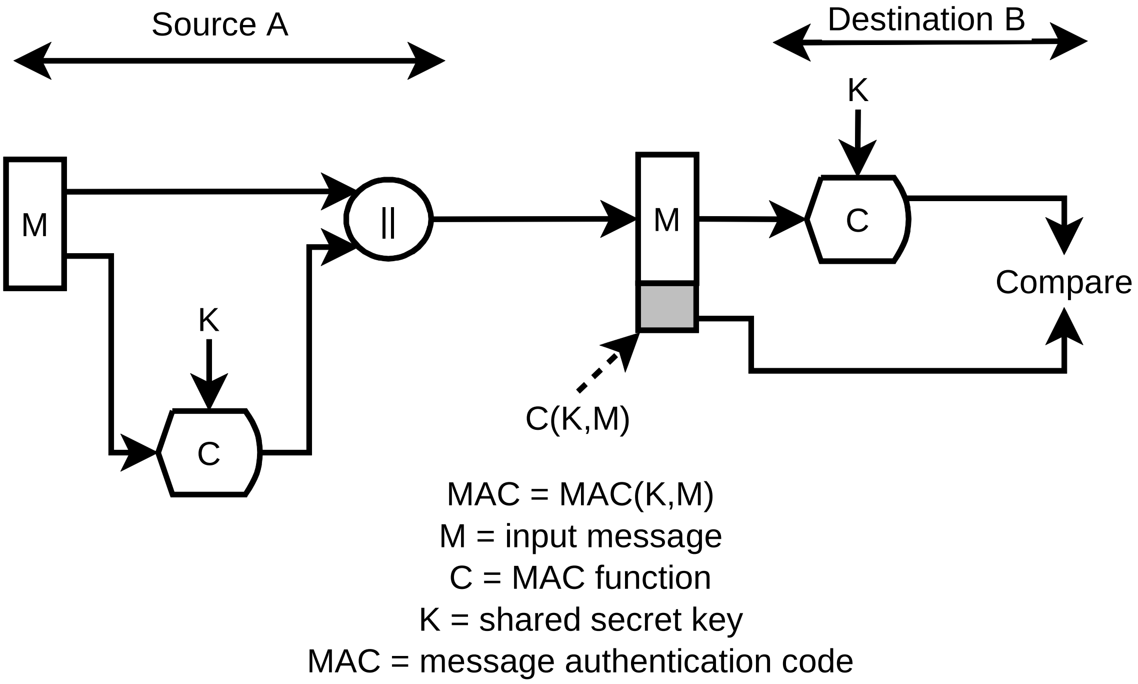


Figure 2.2 *Basic use of message authentication* [1].

In Figure 2.2 the sender (Source A) creates the MAC by using the message M and shared secret key K. The generated MAC is then concatenated with the message M and sent to the recipient (Destination B). The recipient generates the MAC from the received message M using the shared secret key K. Then recipient verifies the message M by comparing her own generated MAC to the given MAC. If the recipient gets same MACs then the message has not been altered and it has come from known sender. [1]

MAC functions can be created using different means, such as, symmetric block cipher or cryptographic hash function. The latter is also known as hash-based message authentication code (HMAC). The usage of HMAC has been motivated by the facts that hash functions are generally executed faster than symmetric block ciphers and the library codes for the hash functions are widely available. HMAC is used in SSL and IP security. [1] Another useful feature is that in RFC 2104 [13] HMAC was designed in a way the embedded cryptographic hash function can be changed, for example, when a faster or more secure hash function is required.

2.5 Digital signature

A digital signature is a security mechanism that provides authentication, data origin authentication, data integrity and nonrepudiation for messages or digital documents. It enables the creator of the message to attach a code that acts as a signature. [1] Figure 2.3 provides a generic model of the digital signature process and Figure 2.4 a simplified depiction of digital signature process's essential elements.

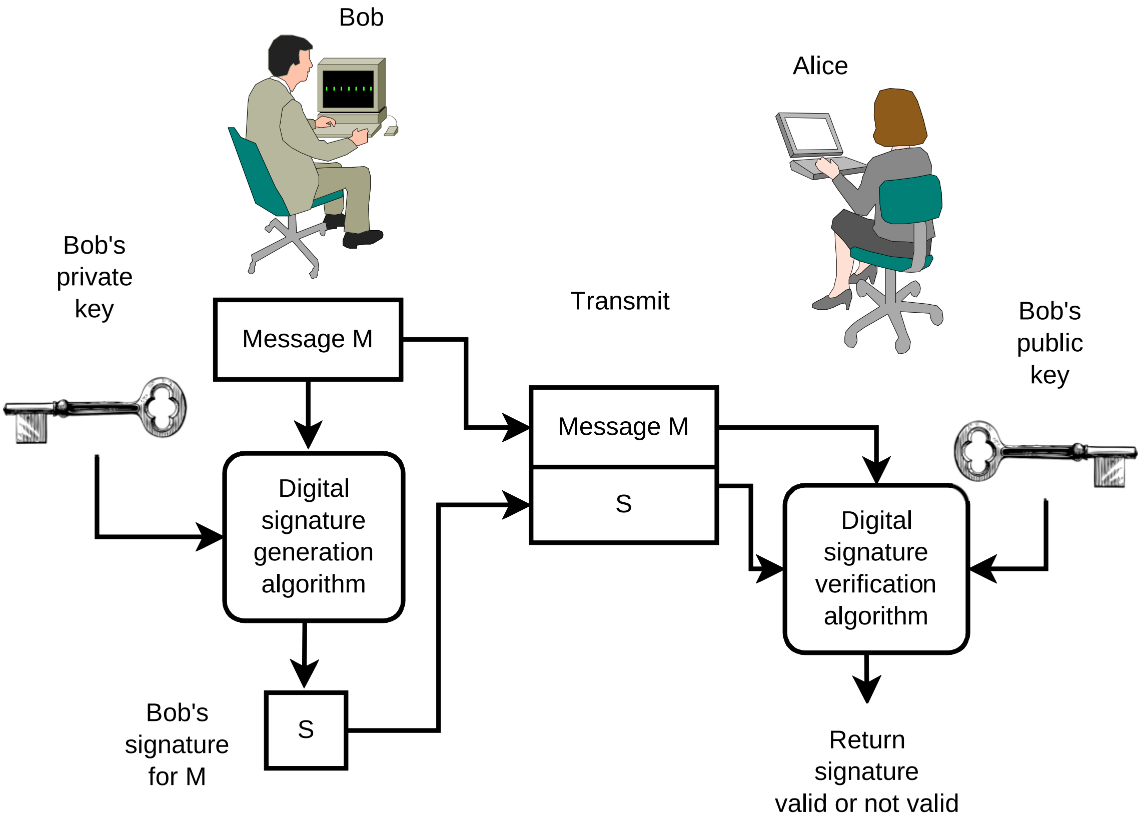

Bob
Alice
Bob's private key
Transmit
Bob's public key
Message M
Message M
S
Digital signature generation algorithm
Digital signature verification algorithm
Bob's signature for M
S
Return signature valid or not valid

Figure 2.3 *Generic model of digital signature process* [1].

The digital signature process starts by Bob creating a message M. Bob then uses his private key to create a signature S from the message M. The message M and the signature S are then transmitted together to Alice (the receiver), which then verifies the signature using the message M, signature S and Bob's public key. The digital signature verification algorithm returns information whether the signature is valid or not valid. [1]

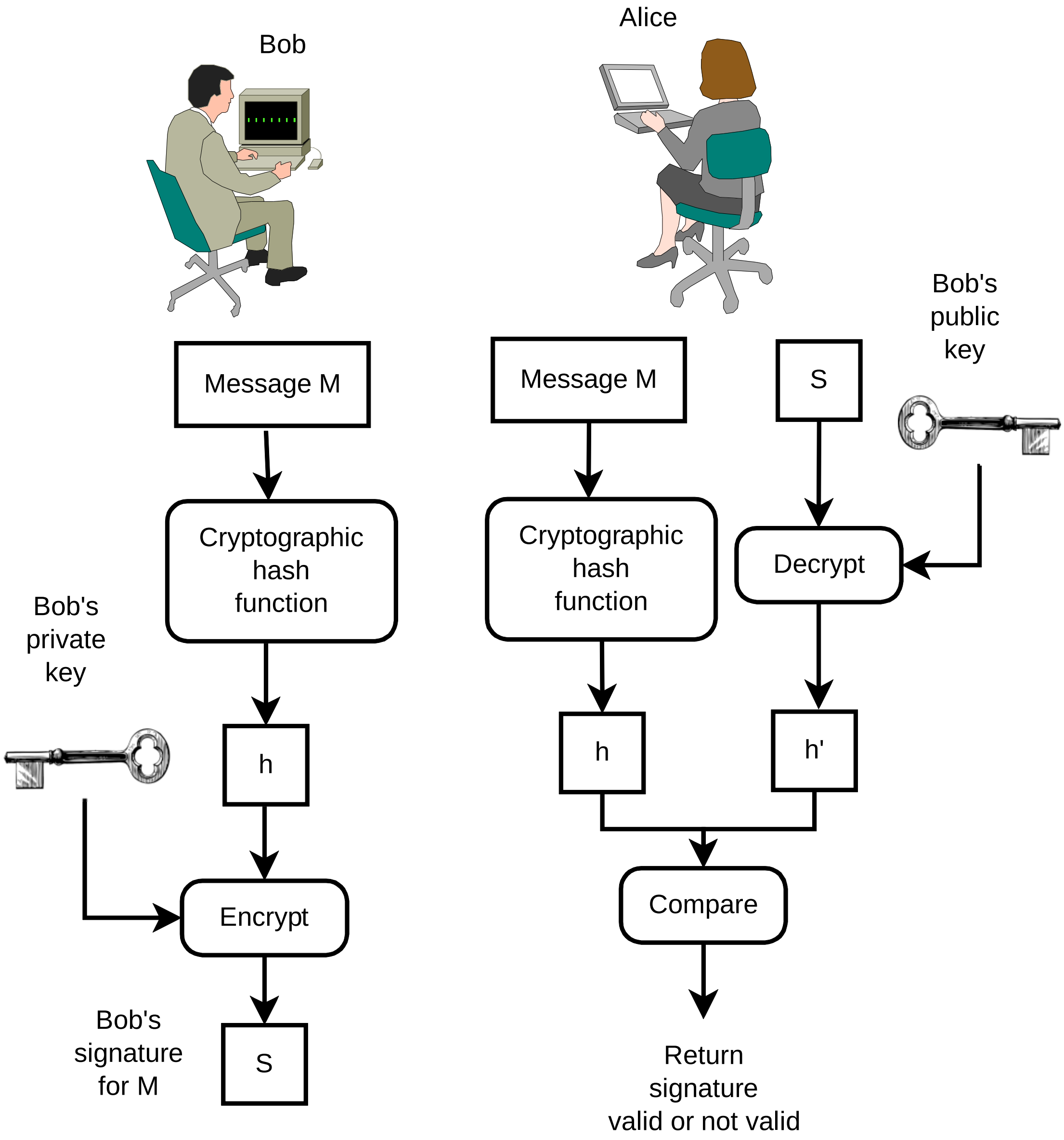


Figure 2.4 *Simplified depictions of essential elements of digital signature process* [1].

The digital generation algorithm in its essential parts is depicted in Figure 2.4 below Bob. The process starts by Bob creating a cryptographic hash value h of the message M. Bob then encrypts the hash value h with his own private key and the encrypted value is the signature S that is sent with the message M to Alice. [1]

Once Alice receives the message M and signature S she starts the verification process. First she creates a cryptographic hash value h from the received message M. Then she decrypts the received signature S with Bob’s public key and gets hash value h’. Hash values h and h’ are then compared together whether they have the same value. If the values are the same then Alice knows the signature S is valid otherwise the message M cannot be trusted. [1]

Cryptographic hash function’s role is to provide message integrity. Alice can be certain the message has not changed after Bob’s signing, when she creates s hash value of the received message M and compares it to the given (decrypted) hash value h' and the values are the same. Alice's certainty of the message originating from Bob comes from the fact that only Bob holds the key used to encrypt the hash value h from the message M. The data origin and nonrepudiation features also stem from the same fact: only Bob holds the used encryption key and therefore the message M must have come from Bob and he cannot repudiate from sending the message.

2.6 Public Key Infrastructure X.509

According to RFC 4949 [4] public key infrastructure is "the set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography". The Public key infrastructure X.509 (PKIX) is a model, created by IETF that is suitable for a certificate-based architecture on the internet. The model is presented in Figure 2.5.

X.509 is originally a standard created by International Telecommunication Union (ITU) Telecommunication Standardization Sector (ITU-T) and later on further developed by IETF, e.g. in RFC 5280 [14]. X.509 defines, among other things, the structure of the certificate and authentication protocol, which are used in other protocols (e.g. Secure Socket Layer (SSL)/Transport Layer Security (TLS)) [1].

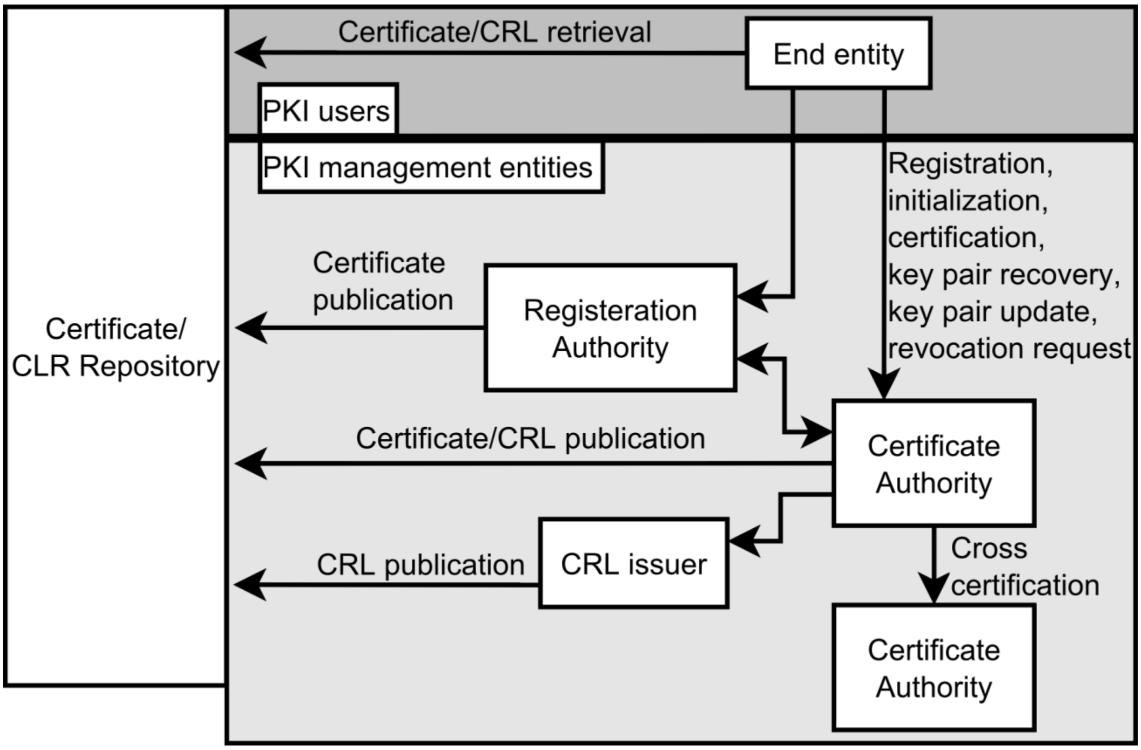


Figure 2.5 *PKIX architectural model* [1].

In the Figure 2.5 term *end entity* stands for end users, devices and other entities that can be identified in the subject field in the certificate. *Certification Authority* (CA) is the issuer of the certificate and (usually) *certificate revocation lists* (CRL). *Registration*

Authority (RA) is an optional party that can assume certain administrative functions (often associated in the registration process) from the CA. *CRL issuer* is another optional component that a CA can delegate to publish CRLs. *Repository* is a term used to describe any method for storing certificates and CRLs, and distributing them for end entities. [1]

The PKIX model has certain management functions. *Registration* is the process where a user makes itself known to the CA that issues a certificate to the user. This process involves some form of mutual authentication. *Initialization* process where client system acquires securely the public key and other assured information of the trusted CA(s) in order to validate certificate paths. *Certification* is the process where CA issues a certificate for the user's public key, and returns the certificate and/or posts that certificate to the repository. *Key pair recovery* allows end entities to recover their encryption/decryption key pair from authorized key backup facility, which is usually the CA that issued the certificate for the end entity. *Key pair update* is a process where the end entity's keys are updated and new certificates are issued. Update is required when, for example, the certificate expires. In *revocation request* an authorized person requests from the CA the revocation of a certificate. The revocation is done when, for example, the private key to the certificate has been compromised. *Cross certification* is a process where two CAs exchange information in order to create a cross-certificate, which is a certificate issued by one CA to another CA that contains a CA signature key used for issuing certificates. [1]

2.7 SSL/TLS

SSL is a security protocol designed by Netscape [15] and it provides end-to-end security services on top of the Transmission control protocol (TCP) [1]. Later on, SSL version 3.0 was standardized and further developed by IETF and at the same time the name was changed to Transport layer security (TLS). SSL protocol stack is presented in Figure 2.6.

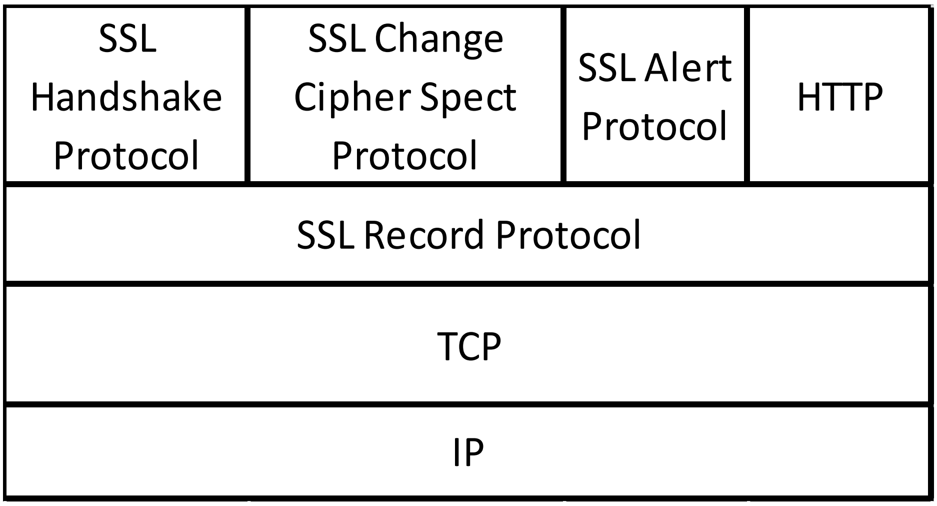


Figure 2.6 *SSL protocol stack* [1].

SSL is a protocol that consists of two layers. The *SSL record protocol* provides basic security services (confidentiality and message integrity) for higher lever protocols, especially for the Hypertext transfer protocol (HTTP). The provided security services are confidentiality and message integrity. The *handshake protocol* defines the keys used in both services. The confidentiality is provided using encryption and message integrity with message authentication code (MAC). [1]

The Handshake protocol allows the server and client to authenticate each other, negotiate an encryption and MAC algorithm, and cryptographic keys used to protect the sent data. The *change cipher spec protocol* uses the SSL record protocol. The change cipher spec protocol consist of a single byte with the value 1 and its purpose is to cause a pending state to be copied into the current state, which updates the cipher suite to be used on this connection. The *alert protocol* is used to transmit SSL-related alerts, such as, handshake failure, bad certificate, certificate revoked, etc. Each alert is either *warning* or *fatal*. In the case of fatal the connection is immediately terminated. [1]

2.8 Man-in-the-middle attack

RFC 4949 [4] defines Man-in-the-middle (MITM) attack as an active attack where the attacker intercepts and selectively modifies communicated data to masquerade as one or more of the entities involved in a communication association [4]. In this thesis the term MITM attack is also considered to include passive attacks, such as, eavesdropping. Figure 2.7 visualises a MITM attack.

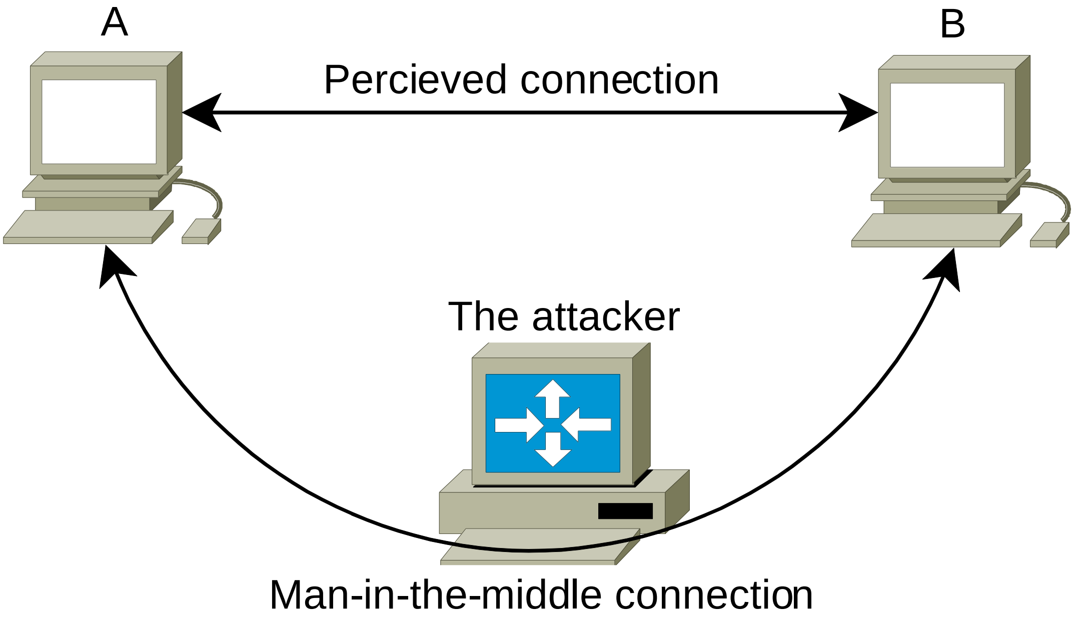


Figure 2.7 *MITM connection and perceived connection.*

In the MITM attack, users A and B perceive that their connection goes straight between A and B. However, in reality the connection goes through the attacker, which is in position to eavesdrop and modify the messages of the communication at will.

3 TEST ENVIRONMENT

The chapter starts with presenting Android OS, what software was used in the work and what is was used for. The chapter concludes in presenting the used devices, the environment and how it worked.

3.1 Android and Used software

This section briefly covers the software used in the test environment. The section starts with OpenSSL. Then it moves to consider software used to perform the MITM attacks and then to the used network analysing tool Wireshark. The section concludes in covering the other used software and operating systems.

3.1.1 Android OS

Android is a mobile operating system based on Linux kernel. Android OS is open source, but in practice nearly every device comes with open source and proprietary software [16]. From 2007 Android has been developed by the Google led Open Handset Alliance (OHA) [17] consortium of 87 hardware, software and telecommunication companies [18]. In smartphones, Android is the most popular OS by having the biggest market share at nearly 85% in Q2 2014 [19].

3.1.2 OpenSSL

OpenSSL is an open source toolkit implementing SSL and TLS protocols, and has a general purpose cryptographic library [20]. In this thesis OpenSSL was used for three things: creating a private and public key pair, creating a CA SSL certificate, and as a generic SSL/TLS client.

3.1.3 MITM software – SSLsplit & mitmproxy

The software used to implement the actual MITM attacks were SSLsplit [21] and mitmproxy [22]. Mitmproxy is capable of proxying only HTTP(S) connections and modifying the HTTP traffic. SSLsplit is a more generic MITM attack software than mitmproxy, as it is capable of performing the attack on any SSL/TLS connection. However, SSLsplit cannot be used to modify the traffic. SSLsplit and mitmproxy were used as transparent proxies, since Android application’s behaviour cannot be changed.

3.1.4 Wireshark

Wireshark is a network protocol analyser capable of deep inspecting and decrypting hundreds of protocols, live data capture and more [23]. In this work Wireshark was used to intercept SSL/TLS traffic and analyse the contents offline. For example, SSLsplit is fully capable of capturing the all the data from SSL/TLS traffic, but it is in a form that is much harder to analyse.

3.1.5 Other software

The MITM attack was done with the author’s laptop, which was running Kali Linux. Kali Linux is a Linux distribution made for penetration testing [24]. In the laptop hostapd application was used to create a WiFi (802.11) access point, where the “victim” would connect. Iptables software was used to route traffic in the laptop to certain ports, for example, those that SSLsplit was listening.

3.2 The environment

The test environment is presented in Figure 3.1. The used devices are a laptop (with software described above) and Android devices: Samsung Galaxy S3 4G (Android OS 4.3.), Samsung Galaxy Trend Plus (Android OS 4.2.2) and Kindle fire HDX 7” tablet Fire OS 3.0 (compatible with Android 4.2.2. [25]).

The work begins (not shown in the Figure 3.1) by creating a public and private RSA key pair, which is then used in creation of self-signed CA SSL certificate. The key pair and certificate were created with OpenSSL. Then the CA SSL certificate is installed to the Android device (red dotted line in Figure 3.1). All the Android devices had an option in the security settings to install a trusted CA certificate, so no special trickery is required from the attacker.

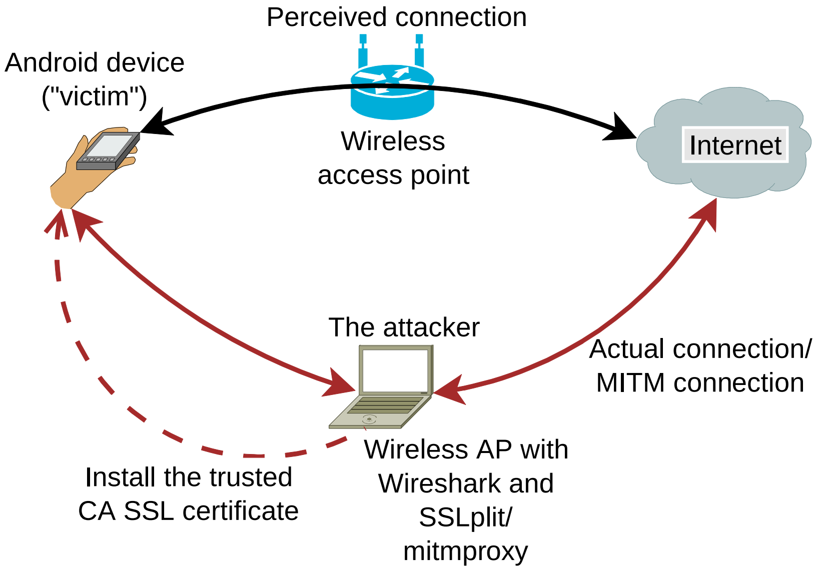


Figure 3.1 *The test environment.*

The laptop is setup to be a wireless access point, where Android devices would connect to. The created private key is given to the SSLsplit software, so that it uses that key instead of creating its own keys, when forging certificates. This is a helpful feature, because the same key can now be given to the Wireshark, in order to decrypt and analyse the captured traffic between the Android device and the attacker's laptop.

When the Android device connects to a service in the Internet it perceives the connection to be straight (black connection in Figure 3.1). During the MITM attack the Android device's connection is actually terminated in the attacker's laptop by the MITM software and then the MITM software initiates a new connection to the original destination (red connections in Figure 3.1). Figure 3.2 presents how mitmproxy proxies HTTPS transparently.

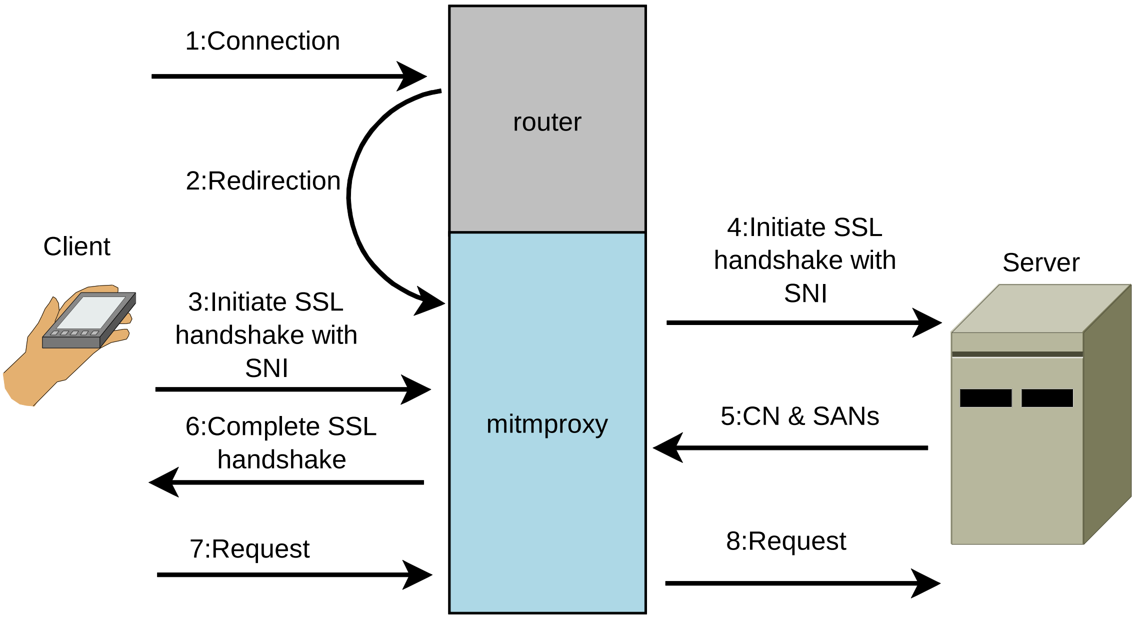


Figure 3.2 *Transparent HTTPS with mitmproxy* [26].

First the client makes a TCP connection to the server. The router redirects the connection to the mitmproxy. In the environment the router was actually the attacker's laptop and the mitmproxy was in the same host (listening to another port). In the third phase, the client believes it is talking to the server, initiates SSL connection with the handshake, and uses the server name indication (SNI) to indicate the hostname it is connecting to. [26] SNI is an extension to the TLS protocol and it is a mechanism for client to tell the server (during the handshake) the hostname the client is connecting to [27].

The mitmproxy will pause the SSL handshake with the client and then it connects to the server and establishes an SSL connection with the given SNI. The server responds with SSL certificate, which contains the common name (CN) and SubjectAltName (SAN) values needed for the forged SSL certificate. [26] SAN is an extension in the X.509 certificate that allows identities to be bound to the subject of the certificate (e.g. email and IP addresses, URIs, DNS names, etc.) [14]. In the sixth phase the mitmproxy gives forged certificate to the client and continues the paused SSL handshake with the client. [26] The client trusts to forged certificates, because the attacker has installed her own rogue CA SSL certificate to the client's device as the trusted CA certificate. In the final seventh and eighth phases, the client generates request and mitmproxy passed the request to the server, once the SSL/TLS connection has been established between the client and mitmproxy.

4 GOOGLE AND AMAZON

The chapter starts with presenting how Google's and Amazon's market places generally work. Then certain Google's services and protocols are described for background information.

4.1 Google's and Amazon's market places

Google's Play Store and Amazon's Shop applications are market places that provide, for example, applications, movies, books, etc. to download and buy. The application market was selected for closer inspection, because it was a feature common to both applications and they would be therefore comparable.

In general, the process flow in a simple use case, when downloading a new mobile application from Google Play Store and Amazon Shop were similar in both markets places. The market place process flow is presented in Figure 4.1. However, it should be kept in mind that Figure 4.1 generalizes the process flow and therefore does not represent all actions market places take. Full listing of actions would have required more detailed reverse engineering, which is out of scope of this thesis.

For a user the use case consisted mainly four parts. First starting-up the market application, then searching and browsing of applications in the market. Thirdly, selecting the desired application and in the fourth part the decision to buy and install the application. In Google Play Store the user had to take one more action, authenticate herself, when buying paid software from the Play Store.

Amazon shop did not require the user to authenticate herself at all during the use case. The user was authenticated when she added her Amazon account to the device and when she added credit card details to the Amazon account at Amazon's website. Amazon required credit card details to be added in order to download any application from the Amazon shop. Google required credit card details only when buying paid applications.

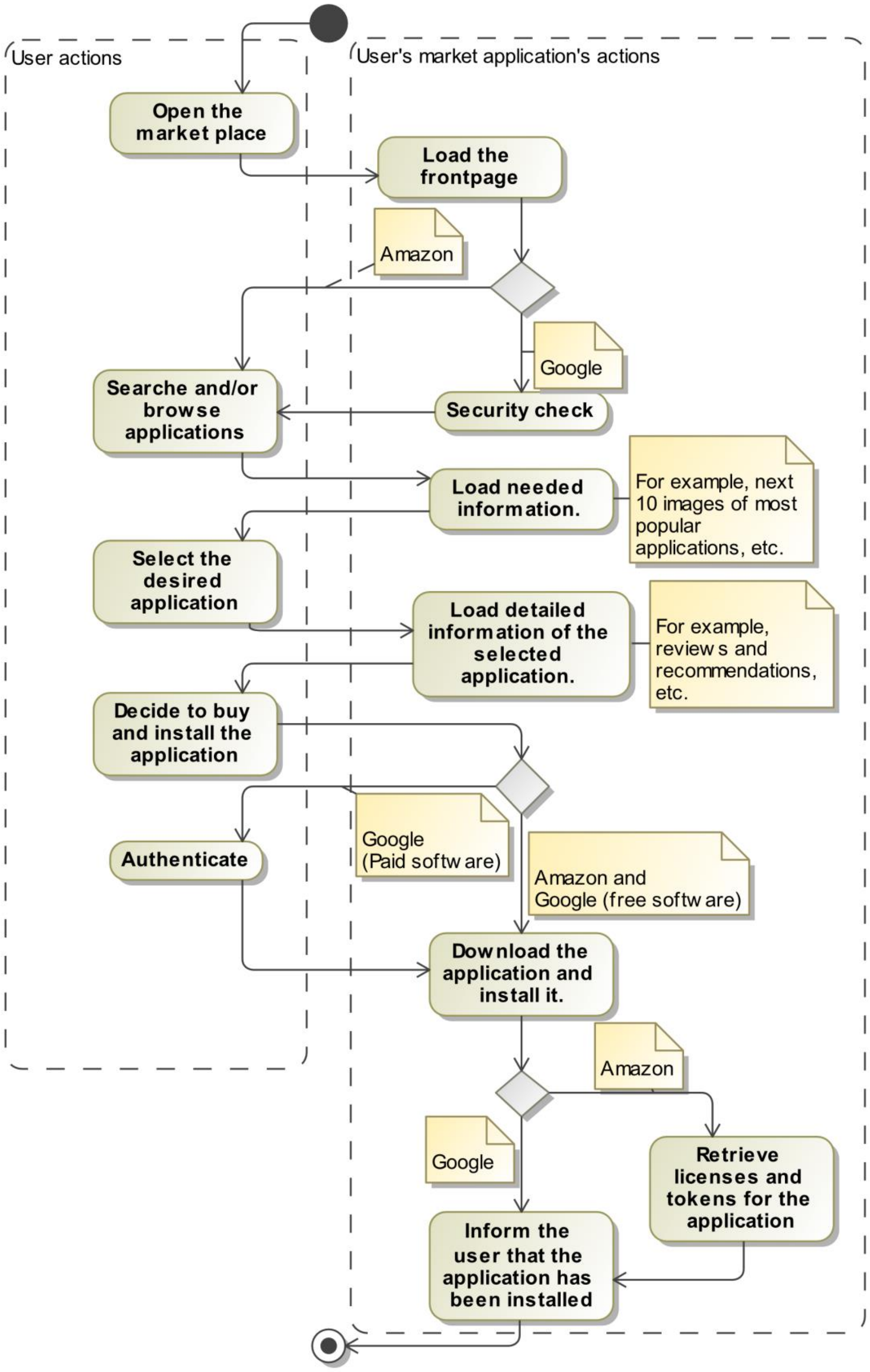

User actions
User's market application's actions
Open the market place
Load the frontpage
Amazon
Google
Searche and/or browse applications
Security check
Load needed information.
For example, next 10 images of most popular applications, etc.
Select the desired application
Load detailed information of the selected application.
For example, reviews and recommendations, etc.
Decide to buy and install the application
Authenticate
Google (Paid software)
Amazon and Google (free software)
Download the application and install it.
Amazon
Google
Retrieve licenses and tokens for the application
Inform the user that the application has been installed

Figure 4.1 *Generalized flow diagram for a simple market use case.*

Market places had five actions in common and one distinct action from the other. First, when the user started the market application, it loaded the front page (which could include the images of top recommended applications etc.). Google’s Play Store also does one security check at this point. The second and third common phases happen when the user starts to search and browse applications, the market then loads the needed resources such as, images for next ten most popular applications. And when the user selects the desired application in the third phase, market downloads the application specific information, such as, details of the application, reviews and recommended applications.

Google's Play Store and Amazon's shop start to differentiate from each other when the user decides to buy the desired application. In Google's case, when the user buys a paid software he/she is prompted to authenticate before the transaction is confirmed. After the authentication the application is downloaded and installed to the device. The user is not required to authenticate when installing and downloading free software from the Google Play Store. Also in Google's case the application receives a security token after the installation when the application is started the first time (not shown in Figure 4.1).

Amazon's shop does not require the user to authenticate herself when the user decides to buy the desired application. Amazon's shop also retrieves automatically the necessary licenses and tokens after the application has been downloaded and installed to the device.

4.2 Google specific details

This section starts with presenting Google Login Service (GLS) and Google Play Services (GPS). Then it moves to cover Google's own proprietary mechanism called ClientLogin, which is used in the Google's Play Store. And finally, the Google 2-step verification is presented.

4.2.1 Google Login Service and Google Play Services

When a user's Google account is successfully added to an Android phone, the phone offers to synchronise the user's data with Google online services (e.g. Gmail, etc.). During this process Google's applications in the mobile device get tokens for the services they represent. The applications get their tokens with the help of Google Login Service (GLS), which works as authentication provider for Google accounts. [28]

It should be noted that the description above concerns only the adding of *a user's Google account*. Since online services have different ways of handling accounts and authentication, Android OS has an account manager, which provides a centralized registry of the user's online accounts, for example, Facebook, Google, Amazon, etc. The account manager uses pluggable authenticator modules (which may be developed by a third party) for different types of accounts, which actually handle validating account credentials, etc. to the specific online service. [29]

While Android OS is open source software, Google's applications are not. Google has set certain compatibility requirements for devices, before phone manufacturer is eligible to license Google Mobile Services (GMS). [30, 31] This means that there are

Android devices without Google applications and services, which therefore work differently (e.g. Amazon Fire tablets) when a user is added to a device.

Google Play Services (GPS) is a platform that offers among other things OAuth 2.0 tokens. GPS is tightly integrated with the Android OS [32] and according to Ars Technica [33] Google applications do not work if Play Services is disabled on the mobile device.

4.2.2 ClientLogin

ClientLogin is a Google's own proprietary mechanism which provides authorization and authentication [34, 35]. Figure 4.2 presents ClientLogin message flow for installed applications, which seemed to be the closest documentation to the observed behaviour.

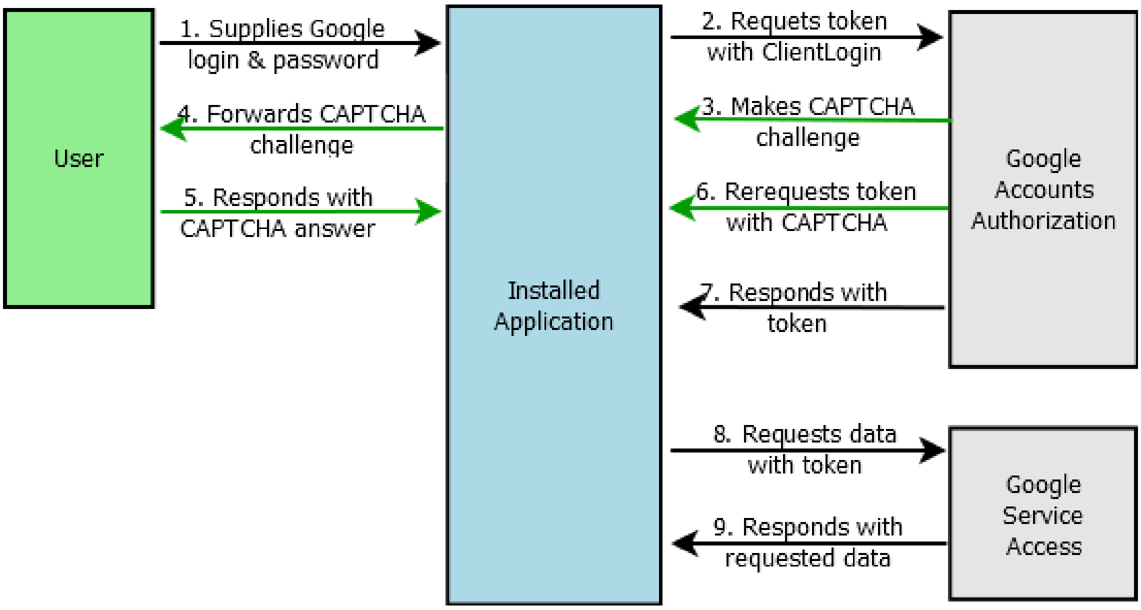

User
1. Supplies Google login & password
4. Forwards CAPTCHA challenge
5. Responds with CAPTCHA answer
Installed Application
2. Requets token with ClientLogin
3. Makes CAPTCHA challenge
6. Rerequests token with CAPTCHA
7. Responds with token
Google Accounts Authorization
8. Requests data with token
9. Responds with requested data
Google Service Access

Figure 4.2 *ClientLogin message flow for installed applications* [34].

ClientLogin works as follows [34]: First, the installed application provides user interface for the user to supply login credentials. When the user has provided her credentials the installed application forwards them to Google. Steps 3 – 6 in Figure 4.2 cover additional vetting, which Google might require. In such case Google issues a CAPTCHA challenge to which the user must answer. After a successful authentication Google provides token for the installed application (step 7 in Figure 4.2). Finally in steps 8 – 9, the installed application sends its request to Google service and is provided with a reply. [34]

4.2.3 2-step verification

2-step verification is a Google’s own mechanism for its users to authenticate more securely to the Google services. During the sign-in the user is required to provide something she knows (password) and something she possesses (phone or security key). In practice, the possession of something is to give a verification code or insert a security key. Google provides several methods for getting verification codes, for example, text message, dedicated application, phone call, printed backup codes, etc.). The verification code is a six-digit one time password (OTP) and the security key has to be compatible with the open standard called “FIDO Universal 2nd Factor (U2F)”. [36, 37]

4.3 Absence of Amazon’s specific details

This thesis does not present any Amazon’s specific feature or protocol, because none was observed. The author personally believes the reason for this observation is that Amazon software is so integrated to their device that they have nearly complete control over it. Amazon also has only its own and a handful of devices, where their software has to work.

This standpoint is completely different from Google, where their software runs in other device manufacturers’ devices. In addition, Google provides more services and applications for its users than Amazon. Also the devices and Android OS version, where their software has to work is more numerous. So the sheer number of services and backward compatibility requires Google to have more protocols and features.

5 GOOGLE RESULTS

This chapter first presents how Google's acquires, uses and renews tokens. After that Google Play Store's security is observed and chapter concludes with discussions on findings. All listings in this chapter have been modified (e.g. parts omitted, bolded, etc.) for readability.

5.1 Tokens

Google Android client applications used tokens heavily. They were observed in nearly every message sent to Google. It can be assumed that some kind of token was always sent along in every message unless otherwise explicitly stated. One general exception to this rule was image downloads, which did not use tokens at all.

During the observations at least five different tokens were observed, which are presented in Listing 5.1. Some additional token-like things were observed such as config-tokens and HTTP cookies. These token-like things are not addressed in depth, because their usage and roles were not well understood.

Tokens were observed to be delivered by client applications in three different ways, which were described in section 2.2. An important way is to use HTTP header Authorization attribute (Figure 2.1), whose value was consisted of two parts; mechanism and token value. Mechanism part consisted of a name of the mechanism used to acquire the token, which was observed not always being the case. Observed example values of the mechanism are; GoogleLogin, AuthSub, OAuth and Bearer.

Tokens differentiated from each other mainly by length and some tokens could be recognised from first few characters. For example, GLS tokens seemed always to begin with DQAA and GA tokens with ya29.1.AADtN. The length of tokens belonging to the same type was not always the same.

```
Master token (GLS):
oauth2rt_1/406mny5jZzr7hLLS8RMwYUX0M0okCkeBLRvA1c1ag3M

GA token:
ya29.aQAQCcMZ2yQR_E0AAAB_MQ3GI5_1GpBSLDsFS-venMX6K3TI3HjpQJRjF49JG1wpiLMYMFAxK
6ugQRK7pM5AGfWGgDwiIqZ1BBeRzJfYpmWOLu5Iavi88c6R807eZQ

GLS token:
DQAAAPsAAABpHeBTHq9gB-vyoONAG_3zIcFUwjAcKTRBLuaEr6XcGza83zwIDDMzWuXheRvLMUYNE2
A0t2bWv_6U2mY_LuvkQ8QlFsYuYS81mPDWPcGpmMvk53U9Nu2-3drSDYOw6Lsoz29Zhm_uerHRZIrt
CU5kmYSbLsO1_yon89OaflF3XGJDw6hw5fhAWqqweGum-rHnJh8eF9QjPgFjEAfsgZo0NHvdkgTeQe
MFx6pq9S_G2-7IPOihP2qrom1ngPURApP35ZbOmAJHzHGahu8a-3x9BtWOfBz4TvOz_W5WdhiTb2H_
lZYj_qn3dBKM1ouFyyWIgS3isUpjSVghO-Ti9-49
```

```
Ubertoken (GA, weblogin):
APh-3FyrN6yRUf6XiAWT1KSM98lvFsLBq0Jrejhz3PZHpA_o3SZjddboqy6aXm_2Q0RtT8ZwQP3Dhu
_vKmu2IEBYG8_TGSMeX5rj7kpm7mbJATj205ye9qh9mdhQiBCvDCJcloxEjjV-loVW62VG1TBDPCnd
KiuvgvOYou5U6TaTjffU9wkFw_D6aZBRWyt2M7DfdECE4EFkliqLj9rpj5mhropHqBWy2iLweZK7Hh
aspaaQdL29CSAPlJEXTJmdBdkE2khl-SXjmIMQtsJxasF6fDh1ZvhvoMAm5bUpHHr-gWdTlmhsIFnr
wUV7JDRCWSb8hlZtOUwIn1RcRv_rDQ0sDf11BsNtObW836IIQL2iMHjeIgAPU6q-FQytyH5KTFQd_s
_yoBYyuEKcCGgW9xq5H0YbTu1Ir4plO67M_V34Gvceeiu9pClVLilgjCXJwDIIMXc-sbAxm3reg8N5
2GxuFVVZD0uEwzQ6Fo3OwzwyWgyBmxMBTy_wgQaf7GbuB4rw-40u13NaseNUT9LmBTEYKY6WVPOXmG
5WxjYUX4SxNbJbHw6AIadIQYTMSDg1wim6cd9hLqPDjW_gyc-6m0Y3WdQSDxQ2xFRHv8uwaY1YR_Wq
IXK55y0

Play Store application download token:
AOTCm0RaEzu5Of11gVPwCxV0eWL_T1E5fqoxesTQUveP5s6j3pjf1B8r9KIzBWeiYNvdzl7Pd4O_s6
XT5La9hVBfrdYqqdCiwREn4YUsHrqh2tKMbg
```

Listing 5.1 *Observed example tokens: Master token, GA, GLS, Ubertoken and application download token.*

Tokens were observed to be used only for authorization and it is not known whether tokens had other information included within them in some way. For example, GLS and Ubertoken are long strings, which can easily contain encoded information.

In general, the acquirement and usage of security tokens for Google's service happens in three phases. At first a "master token" is acquired, and then in the second phase the master token is used to acquire another token for a mobile application or its service. And finally, the mobile application or service sends its request to Google along with the second acquired token included in the message. There is also one exception to the three phased token usage, namely Weblogin, which also uses the master token, but the application specific token is acquired by exchanging HTTP cookies.

All tokens were observed to be requested by sending a HTTP message to *clients.android.google.com/auth* address. And it was noticed that in token acquirement two different user-agent attributes were used in the HTTP header, Google Login Service (GLS) and Google Auth (GA). Both of these user-agents worked in principle in the same 3 phases. Differences were in parameters used in token acquirement, applications that used either of the user-agents, and in message flow.

5.1.1 Master token

Master token was observed to differ from other tokens in two ways. It is acquired only when a user's Google account is registered to an Android device (Figure 5.1) and when the user starts to use Google's 2-step verification for authentication. The master token was observed to be used only for acquiring new tokens for Android device's Google applications or services, when they were in need to access Google's resources and user's data.

Google does not use the "master token" name, instead in the observations the master token was in a parameter called token or EncryptedPasswd. Latter name is used for two different things and therefore the name "master token" is used in this thesis to clarify naming and emphasize the master token’s special role. The name master token is to the author’s best knowledge first used by Nikolay Elenkov [28].

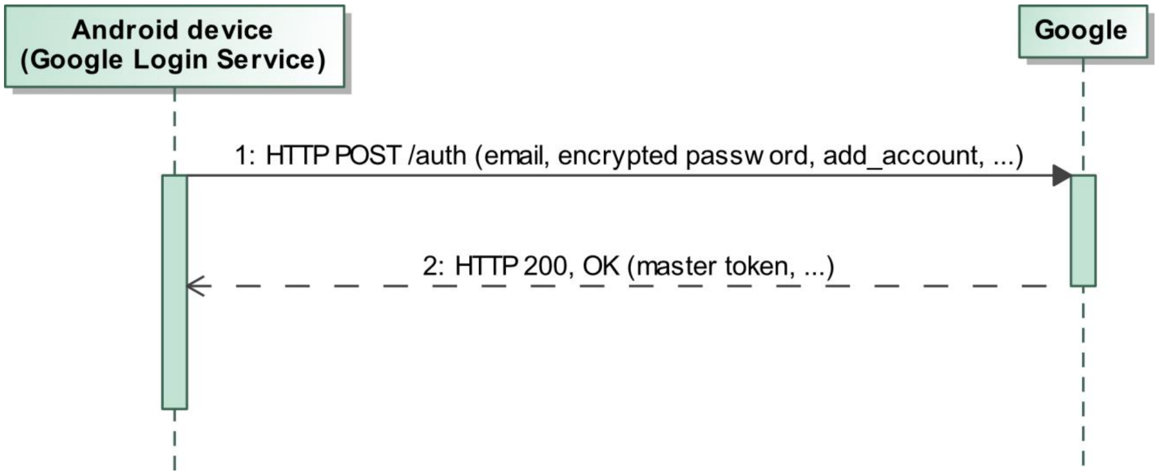

Android device
(Google Login Service)
Google
1: HTTP POST /auth (email, encrypted passw ord, add_account, ...)
2: HTTP 200, OK (master token, ...)

Figure 5.1 *Master token acquirement, relayed parameters are in brackets.*

Master token acquirement started with registering a user’s Google account to the Android device. GLS sent HTTP POST /auth message (Listing 5.2) to Google (address *android.clients.google.com*) containing, among other things, user’s Google email address, and presumably user’s encrypted password (parameter EncryptedPasswd) and parameter *add_account*. Email address was thought to be used as login identification and parameter add_account indicating that the user’s account is to be added to the device. The EncryptedPasswd parameter’s value was observed to change every time the user registered to the device. According to Elenkov [28] Google very likely uses 1024-bit RSA key and the optimal asymmetric encryption padding (OAEP) to encrypt the user’s password.

```
POST /auth HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: android.clients.google.com
User-Agent: GoogleLoginService/1.3 (m3 JSS15J)

accountType=HOSTED_OR_GOOGLE&Email=*****@gmail.com&has_permission=1&add_accoun
t=1&EncryptedPasswd=AFcb4KRHuI22tP-Yd3ILuXt1PsGWXM2ZaMkHVSgUyMxwQkLTr3YUExnI7Q
vh2HT1kZuXbPhZggs4ZQv0ruwsxPruv7NhdcYy9qylXQ6dhnYKiyC9FgIWf67i8gSAYwSYeJXsnQsl
8aA61mTXgybGOTOAn0ypiyXlowgE4n7cy7U1YB-RNQ==&service=ac2dm&source=android
&androidId=32f8ab24222c9535&device_country=fi&operatorCountry=fi&lang=en&sdk_v
ersion=18
```

Listing 5.2 GLS master token request message (Message 1, Figure 5.1).

```
HTTP/1.1 200 OK
Content-Type: text/plain; charset=utf-8

SID=DQAAAMsAAADmPD ...
LSID=DQAAAM0AAAC2O ...
Auth=DQAAAM0AAAAcO ...
services=mail,doritos,googleme,lh2,talk,android,cl,friendview,
chromiumsync,multilogin
Email=****@gmail.com
Token=oauth2rt_1/406mny5jZzr7hLLS8RMwYUX0M0okCkeBLRvA1c1ag3M
GooglePlusUpgrade=1
PicasaUser=Tuomo Tutkija
RopText=
RopRevision=1
firstName=Tuomo
lastName=Tutkija
```

Listing 5.3 GLS reply message for Master token request (Message 2, Figure 5.1).

The most important part of the reply message for the master token request was parameter *Token*. The Token field contained the requested master token. The rest of the token parameters in the message (*LSID*, *SID* and *Auth*) were never observed to be used. In fact, LSID and SID parameters in any GLS token replies were never observed to be used. The reply message also contained the user’s name and information regarding to other Google services.

5.1.2 GLS token

The GLS token acquirement process is presented in Figure 5.2. The first message exchange shows how the master token is used to acquire Auth token for a mobile application.

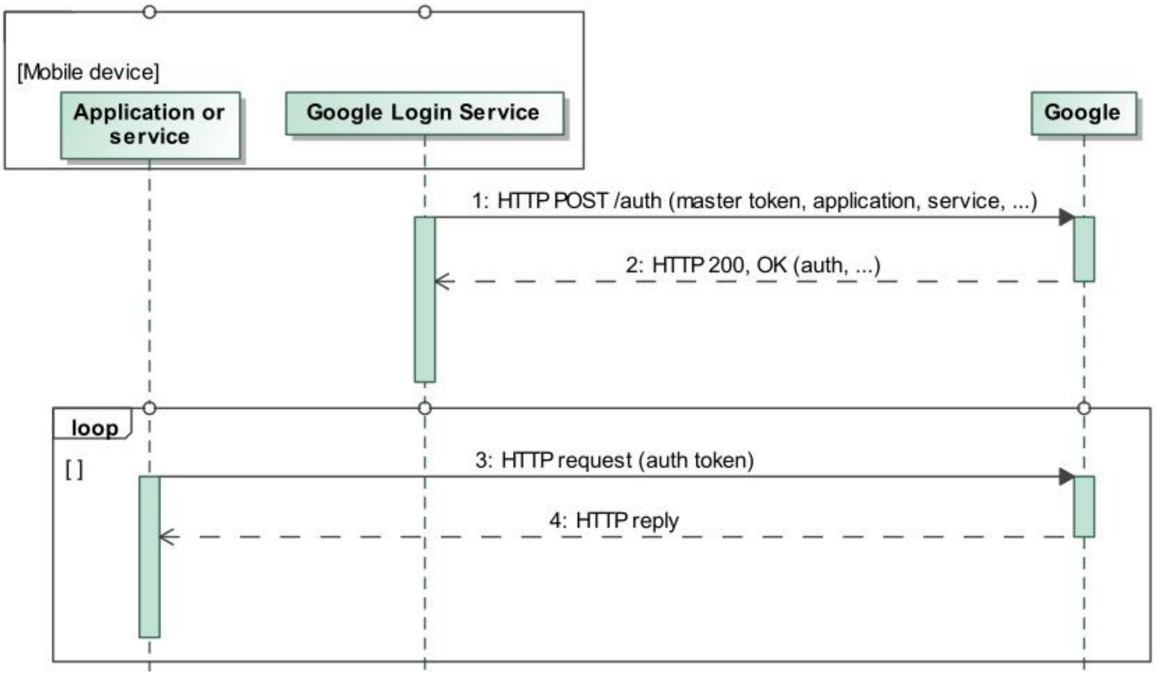

[Mobile device]
Application or service
Google Login Service
Google
1: HTTP POST /auth (master token, application, service, ...)
2: HTTP 200, OK (auth, ...)
loop
[]
3: HTTP request (auth token)
4: HTTP reply

Figure 5.2 *Sequence diagram of token acquirement with Google Login Service.*

The Auth token is application or service specific (depends on which one the token was acquired for). For example, the application com.google.android.gsf.login was observed in token requests together with services such as ac2dm, mail and chromiumsync. Newly acquired Auth token is then passed on in the second message exchange, when the application sends it request to Google (Message 3, Figure 5.2). The token is acquired usually only once for the application and the same token is then always used by the same application.

```
POST /auth HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 339
Host: android.clients.google.com
User-Agent: GoogleLoginService/1.3 (m3 JSS15J)

accountType=HOSTED_OR_GOOGLE&Email=****@gmail.com&has_permission=1
&EncryptedPasswd=oauth2rt_1/406mny5jZzr7hLLS8RMwYUX0M0okCkeBLRvA1c1ag3M
&service=sierra&source=android&androidId=3c5a5c3bcd5b2d7f
&app=com.android.vending&client_sig=38918a453d07199354f8b19af05ec6562ced5788
&device_country=fi&operatorCountry=fi&lang=en&sdk_version=18
```

Listing 5.4 GLS token request message (Message 1, Figure 5.2).

```
HTTP/1.1 200 OK
Content-Type: text/plain; charset=UTF-8

SID=DQAAAPgAAAB461lU_C32QIyonhi5oxcDsG0jRS--pHSb9EncQxsYOArHghnBmVbBoMEVdr2W_d
niueIIVWYzh_2ym6DSW00KwUmGpNAanWOATIA3_sM3JIUzMs-AkYyK8zkhp4iIPmSuF9jTcA5ZiYiC
6rSumbChMbH5Y3pDdmZ-edLEzq7XHbR39g8L_l6ZDjqKyODB0XSy_hr1lkoLVeNraBB0RAsgKYiJ04
AKv2m1hofhMazn5anssfHnc3v2XL7lyIyE5H9WsCE0b1iZzH_1p1Fm-4pbM5zEOC4Hh7PWwotxa_a-
CMtig4ahsxhFmW6FJQKOYt98g3dlXknqDBcSdzXvygMo
LSID=DQAAAPwAAAC29nFq6fprKTxuKhzOaguRpSAta0gIb55ryrrvEMAm-BtM5qSjBc5VQLM3y1hl0
xZ7SNH2Yu9miVnyJ5eL8kUVzP8cbOQGIp6r7-KbDmlg134QPwMx9T_VwamZT31jB6uxc6lMCabeawO
Z4vQ-otofjYwiQdkAWsB706G-gktTntuOP40-VRqv2BIM6qsCVwA_nHx688513TmHEF_Xh8L9uENGs
CW44dsmWov_Fqz5XoL-w35qzZUXxGHIf6GeRBDV_-insoOJCW8FxSpDjEJCjc-E2yReK3rt5TWg6mx
wO65qCsQQK3xIjdFHuZxXVS3BgjgL4FQeGvslFcc3c7Bm
Auth=DQAAAPsAAAC29nFq6fprKTxuKhzOaguRpSAta0gIb55ryrrvEMAm-BtM5qSjBc5VQLM3y1hl0
xZ7SNH2Yu9miVnyJ5eL8kUVICvKtImm_YPuAyJK78sOcKChd_Aylv8whDWI8Ygu-ZxSi9G8CIrSOi9
qQHyMTFN6nYJH_oGA6GYQTz3k541NRIcL6gHyXKeTVeoLWUqcVvVmamHPnII9MFTlHP28Mw6DHoBQv
cx-98U-oGzc7UdwZsEj9bzll1p43FiiUnWdxJHXn5QBydsPa70hagAgaEIPnsFcKRcQI_ngmNehAWV
d4Cdz5fyyoTGmQRJpFkHo_RVhYJg7JsGaeMV4amVHr838
issueAdvice=auto
services=mail,doritos,googleme,lh2,talk,android,cl,friendview,chromiumsync,
multilogin,sierra,esmobile
```

Listing 5.5 GLS token reply message (Message 2, Figure 5.2).

GLS token request message (Listing 5.4) contained in the message body parameters, such as, user’s email address (account name), EncryptedPasswd (master token) and the name of the application requesting the token. The EncryptedPasswd parameter value in Listing 5.4 contains the master token and it is different from parameter used in Listing 5.2, which presumably contains the user’s encrypted password.

The reply message contained the requested token in Auth parameter. As stated earlier, other tokens (SID and LSID) in the reply message were never observed to be used.

5.1.3 GA token

GA method to acquire the application specific token is almost identical to the GLS method, differences were on request parameters, token expiry and usage. The process is presented in Figure 5.3.

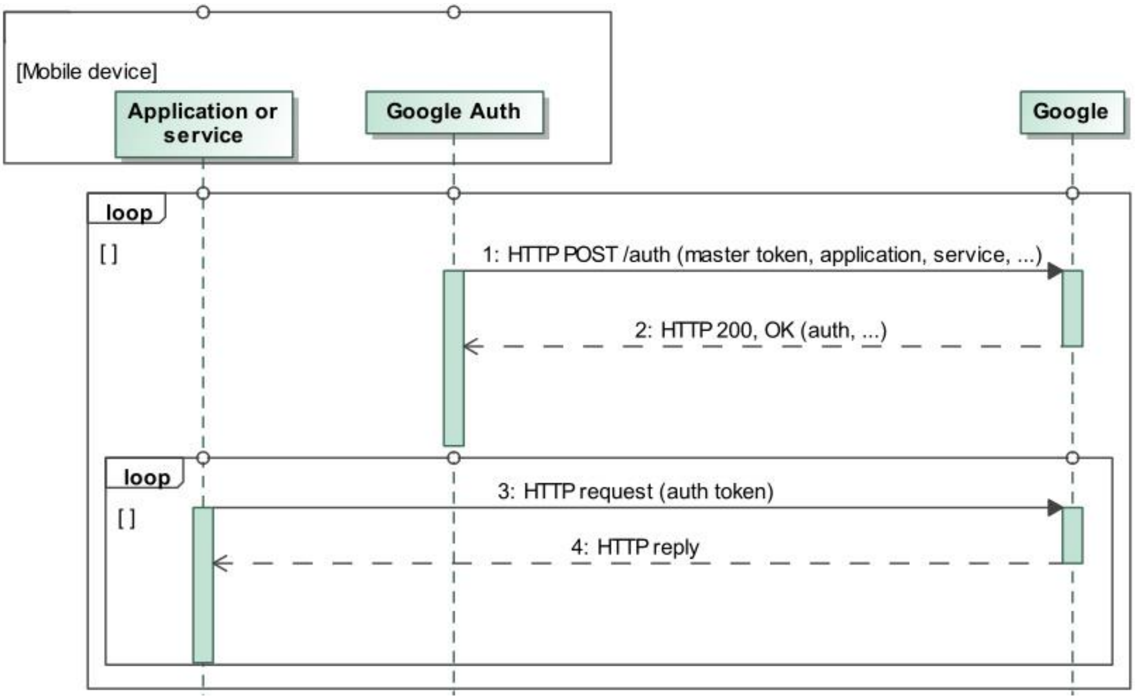

[Mobile device]
Application or service
Google Auth
Google
loop
[ ]
1: HTTP POST /auth (master token, application, service, ...)
2: HTTP 200, OK (auth, ...)
loop
[ ]
3: HTTP request (auth token)
4: HTTP reply

Figure 5.3 *Sequence diagram of Auth token acquirement with GoogleAuth.*

Figure 5.3 presents the GA token acquisition process and usage. The main difference between the GLS and the GA is that GA tokens have expiry time and therefore applications need to request new tokens.

```
POST /auth HTTP/1.1
device:
app: com.google.android.gms
Content-Type: application/x-www-form-urlencoded
Content-Length: 468
Host: android.clients.google.com
User-Agent: GoogleAuth/1.4 (m3 JSS15J)

device_country=fi&operatorCountry=fi&lang=en_GB&sdk_version=18
&google_play_services_version=3225130&accountType=HOSTED_OR_GOOGLE
&system_partition=1&Email=****@gmail.com&has_permission=1
&service=oauth2:https://www.googleapis.com/auth/calendar&source=android
&androidId=3c5a5c3bcd5b2d7f&app=com.google.android.syncadapters.calendar
&client_sig=38918a453d07199354f8b19af05ec6562ced5788
&EncryptedPasswd=oauth2rt_1/406mny5jZzr7hLLS8RMwYUX0M0okCkeBLRvA1c1ag3M
```

Listing 5.6 GA HTTP token request (Message 1, Figure 5.3).

The GA request message contained all the same parameters as GLS token request message and in addition two to four other parameters depending on the message. New parameters are google_play_services_version, system_partition, callerPkg and callerSig. Parameters callerPkg and callerSig were not always present in the request. As with the GLS token request message, EncryptedPasswd contained the master token.

```
HTTP/1.1 200 OK
Content-Type: text/plain; charset=UTF-8

issueAdvice=auto
Auth=ya29.PgBVooZERI8NA1AAAABnFIMaNWZcbuZS-0hyiBtsU1P2kEK3l0CUnG3se9bzqOk95ksJ
ialWljLSQk4krt4RjRmdE8MgUj7dwc1GIGjhfDwV048X1o5Uogm8foNwTQ
Expiry=1405060390
storeConsentRemotely=0
```

Listing 5.7 GA HTTP token reply (Message 2, Figure 5.3).

```
SID=DQAAAPgAAABhfx1sPUmm ...
LSID=DQAAAPoAAABpk84aGPG ...
Auth=DQAAAPsAAACgMO2Jn35 ...
issueAdvice=auto
services=mail,doritos,googleme,lh2,talk,android,cl,friendview,chromiumsync,
multilogin,sierra,esmobile
```

Listing 5.8 Alternative GA token reply message (Message 2, Figure 5.3).

The message body of the GA token reply message (Listing 5.7) contained the requested token and the token expiry time in unix time. GA tokens had expiry time, which suggested that GA tokens were valid only up to a certain time and after that applications needed to acquire a new GA token. However, the expiry time was not followed all the time. Token requests for the same application and service were observed to be made before the previously set token expiry time.

The GA reply message was not always the same. In cases where the GA reply message was different (Listing 5.8) it was noticed to be identical with GLS reply message (Listing 5.5), where the requested token is given in Auth parameter. These alternative reply messages were also noticed to be given when the service parameter value in the GA request message (Listing 5.6) did not begin with “oauth2:”.

5.1.4 Weblogin

Weblogin was observed to be a method for acquiring cookie tokens. The process is presented in Figure 5.4.

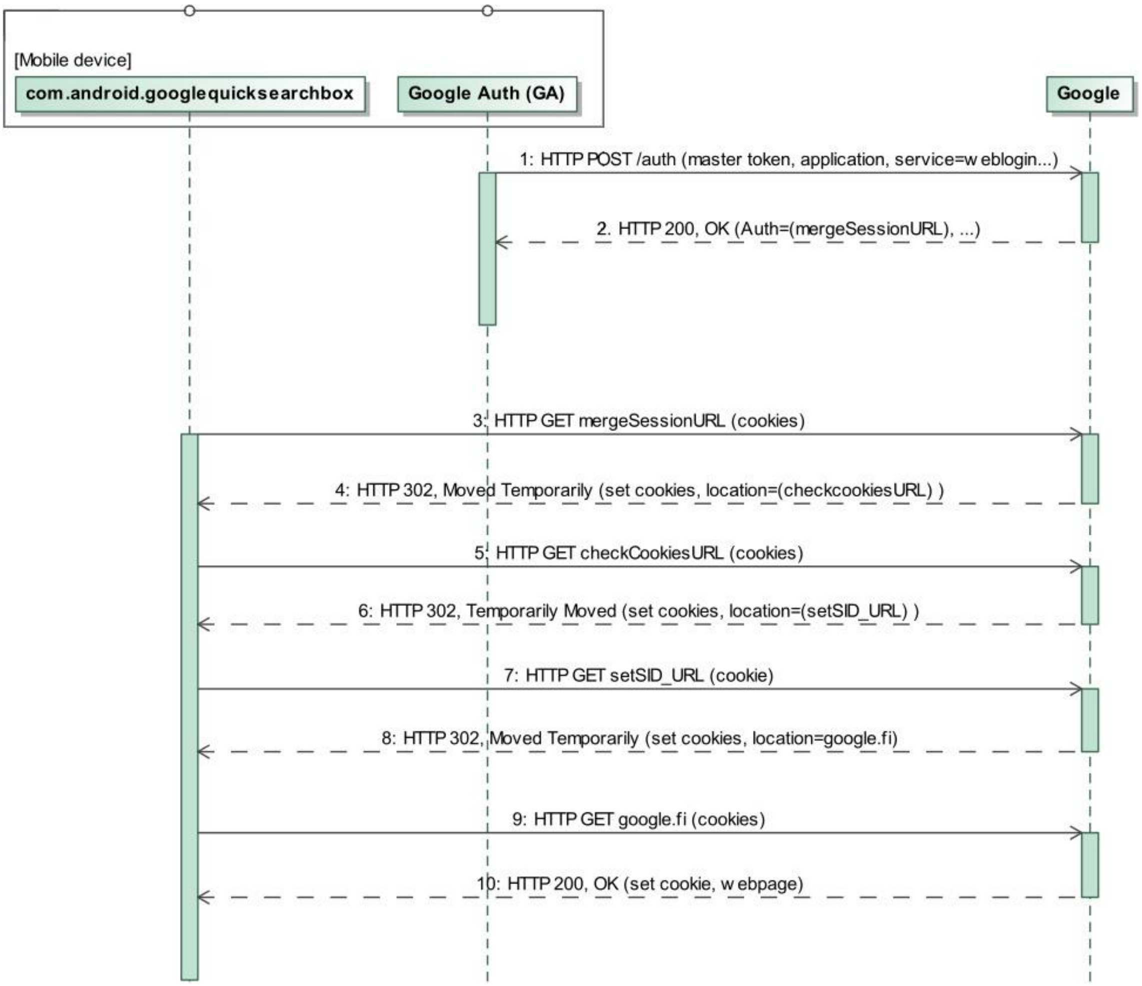

[Mobile device]
com.android.googlequicksearchbox
Google Auth (GA)
Google
1: HTTP POST /auth (master token, application, service=w eblogin...)
2. HTTP 200, OK (Auth=(mergeSessionURL), ...)
3: HTTP GET mergeSessionURL (cookies)
4: HTTP 302, Moved Temporarily (set cookies, location=(checkcookiesURL))
5: HTTP GET checkCookiesURL (cookies)
6: HTTP 302, Temporarily Moved (set cookies, location=(setSID_URL))
7: HTTP GET setSID_URL (cookie)
8: HTTP 302, Moved Temporarily (set cookies, location=google.fi)
9: HTTP GET google.fi (cookies)
10: HTTP 200, OK (set cookie, w ebpage)

Figure 5.4. *Sequence diagram of Weblogin method of acquiring token.*

The Weblogin method for acquiring token was observed to happen rarely and the token was acquired for the Google quicksearchbox application. The first message resembles the previously described GA token acquirement method, but there is a difference in *service* parameter (see below).

```
...&service=weblogin: service=hist&continue=https://www.google.fi...
&app=com.google.android.googlequicksearchbox ...
```

The reply (Message 2, Figure 5.4) from Google did not contain auth token. Instead it contained following the URL (including new token, uberauth) and expiry with value of zero.

```
Auth=https://accounts.google.com/MergeSession?args=service=hist&
continue=https://www.google.fi&de=1&uberauth=APh-3FzU8H02fj1MkZT3ew28t
ju0s0cpvZ0ws0jyLi3YKNFoEWsZ0G0BZ0W3lIPQfWXoDOnVCps3u80kkY7fk2mrCYrv6y9
ufg3MxOs-rDo5U5Zz_JuoQR9yvpxG_i12_Y6n5reeNizC2wBAEH2ZCAI9RlYjISIdyRsWx
LqsIoxCGDc9ajOsBursFqKbzRuf0OY3yg3iSy-rGqsdQ_BA8Igvm9Q25ONlQmJM-uvaXo5
YNXIFDBo82jHBO0VF13sVdpytRYucerJgkaLdrz6K9IhcU1Dmw4zT7bJdtd8LSYUac1ZXQ
u4hoLp7hYOO1otrQLhJOt4kwJKdkTTX-7dGiOWxowSCooCZCUVc9yq0QnQ_rhf8sMpSdko
RYTnlHMHwTmEKANwNF-eT&source=AndroidWebLogin
Expiry=0
```

Listing 5.9 Weblogin HTTP token reply (Message 2, Figure 5.4)

In the rest of message flow most of the information exchanges happened in cookies, but the exchanged information was not understood. According to Google [38], they use different types of cookies, such as preferences, security, processes, advertising, session state, and analytics cookies. Security cookies are used to authenticate user and to protect user's data from unauthorized parties. For example cookies named SID and HSID contain digitally signed and encrypted records of a user's most recent sign-in time and Google account ID. [38]

5.1.5 Token renewal

One experiment was developed in order to find out the token renew process. In the experiment the mobile device's time was manually set to different future time (from one week to 12 months), from the Android device's own settings, that were usually past the acquired GA token's expiry times. Google's applications were observed always to send tokens to Google. This hinted that the application did not check the token expiry and that the responsibility was in Google's server side. GA tokens were nearly always observed to be used immediately and with a few exceptions, token's usage was not observed; meaning that the token delivery method could have not been understood, found or the token was simply not used.

All tokens provided by the GLS (Auth and Master token) were not observed to expire during the experiment. In other words, the same token was always used, when the application sent new requests to Google. However, tokens were invalidated, for example, when an application was updated; master token was revoked manually by the user and when the user started to use 2-step verification. The user is able revoke the master token manually at Google's account settings web page. One updated application that required new tokens was com.android.google.gms, which is according to Shiram the Google Play Services application [39]. Listings 5.10 and 5.11 present the reply messages Android device got from Google during the application update. The reply presented in Listing 5.10 was observed to happen before the update and Listing 5.11 after the update.

```
HTTP/1.1 401 Unauthorized
Content-Type: text/html; charset=UTF-8
WWW-Authenticate: GoogleLogin realm="https://accounts.google
.com/ClientLogin", service="androidmarket"
```

Listing 5.10 Reply for replicateLibrary message.

```
HTTP/1.1 403 Forbidden
Content-Type: text/html; charset=UTF-8
```

Listing 5.11 Reply for ApiRequest message.

When a Google’s application send a request to Google and the token was invalid, Google replied with a HTTP 401 Unauthorized message. The same request with the same token was usually sent 2 – 3 times before the application stopped sending requests and then tried to acquire a new token for the application. When the master token in the token request was invalid, revoked, etc. Google replied with HTTP 403 Forbidden. Depending on the case, when the master token is invalid the user is usually required to authenticate. The only observed exception was the com.android.google.com application update. The user is kept oblivious to the problems and notices problems only if the service she gets is slow and when the master token is invalidated and she needs to authenticate.

5.1.6 Token request parameters

Table 5.1 presents parameters seen in the captured communications. GLS has two entries, because it uses different set of parameters when acquiring a master token. Also parameters marked in parenthesis are not always present.

Table 5.1: *Observed parameters in GA, GLS and master token acquirement.*

Parameter:	**GA**	**GLS**	**GLS** (Master token)
device_country	x	x	x
operatorCountry	x	x	x
lang	x	x	x
sdk_version	x	x	x
google_play_services_version	x	-	-
accountType	x	x	x
system_partition	x	-	-
Email	x	x	x
has_permission	x	x	x
service	x	x	x
source	x	x	x
androidId	x	x	x
app	x	x	-

client_sig	x	x	-
callerPkg	(x)	-	-
callerSig	(x)	-	-
add_account	-	-	x
EncryptedPasswd	x	x	x

AccountType parameter defines the type of the account. Only one value was observed: HOSTED_OR_GOOGLE. According to ClientLogin's documentation [34] the value in question refers to an action where the user's Google account is first attempted to authorize for hosted account and if failed then attempt for Google account [34]. System_partition parameter was left unknown. It was observed to always have the same value: 1.

Service parameter contained information regarding what service requested a token. The value was different depending on which user-agent made the request. Values in GLS requests were either codenames (e.g. sierra, sj, etc.) or more self-explanatory (e.g. androidmarket). GA request's values were either for specific service (e.g. oauth2:https://www.googleapis.com/auth/calendar) or contained multiple values for a service and defining specific rights.

```
oauth2:https://www.googleapis.com/auth/plus.circles.read
https://www.googleapis.com/auth/plus.circles.write
https://www.googleapis.com/auth/plus.media.upload
https://www.googleapis.com/auth/plus.pages.manage
...
```

Source parameter value was though to represent the origin of the request. The value for the source parameter was always: android. AndroidId parameter is according to Shiram [39] a unique device id. The AndroidId is probably the ANDROID_ID value described in the Android OS reference [40]. The ANDROID_ID is a 64-bit hex string, which is randomly generated when a user's Google account is added to the device. The value changes every time the device had reset performed. [40] This behaviour was in line with the author's observations.

According to Shiram [39] the client_sig parameter contains a signature value of the Google Play services application (com.google.android.gms) [39]. Parameters client_sig and callerSig (when present) were observed to have the same value (38918a453d07199354f8b19af05ec6562ced5788), which was not observed to change during the observations.

5.2 Practical security in Play Store

This section first presents the message flow when downloading a free application. Then a paid application is considered, which includes new security mechanisms. The section concludes with presenting miscellaneous findings.

5.2.1 Free application case

The most essential messages when downloading a free application from Play Store are presented in Figure 5.5.

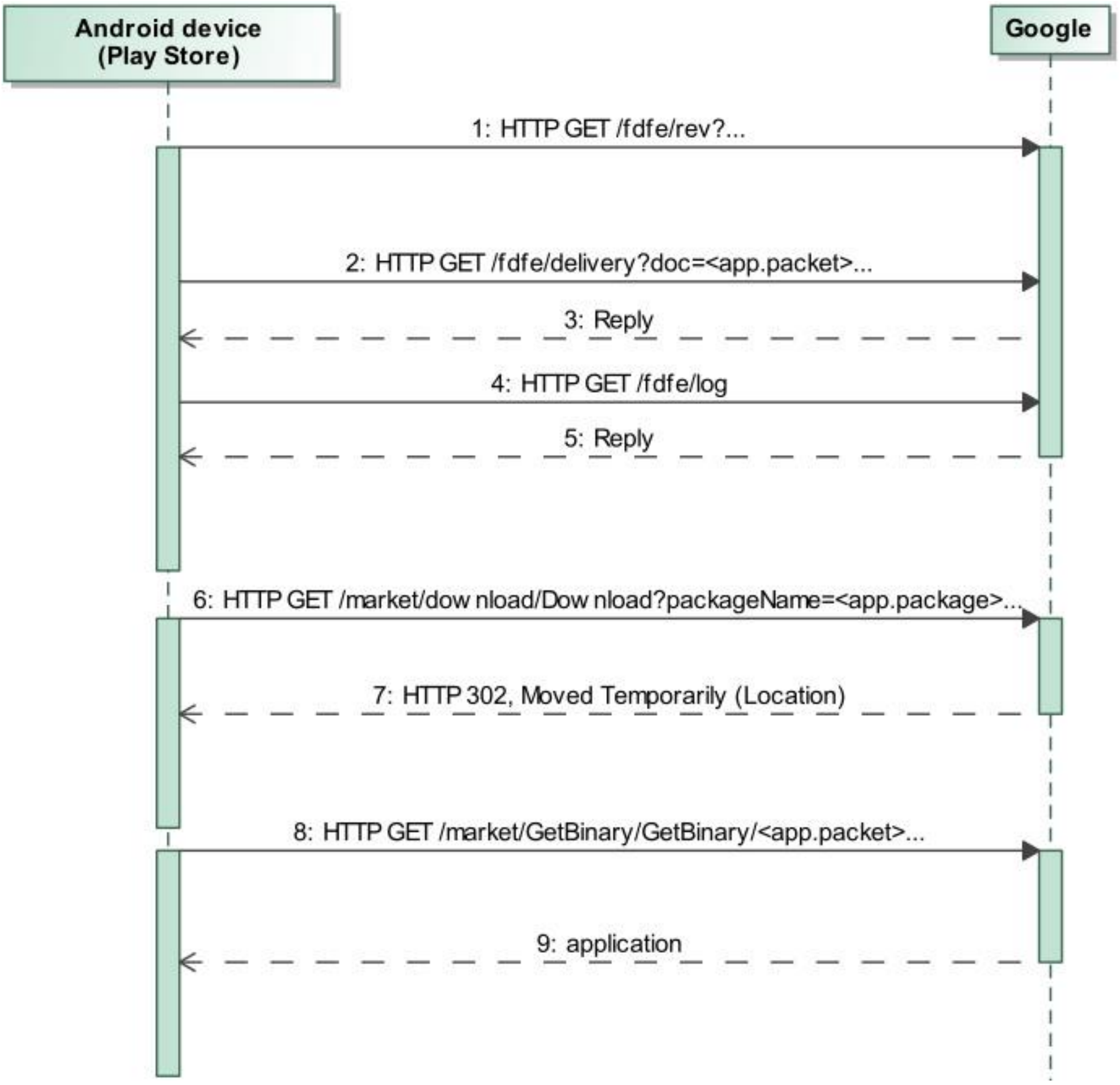

Android device
(Play Store)
Google
1: HTTP GET /fdfe/rev?...
2: HTTP GET /fdfe/delivery?doc=<app.packet>...
3: Reply
4: HTTP GET /fdfe/log
5: Reply
6: HTTP GET /market/dow nload/Dow nload?packageName=<app.package>...
7: HTTP 302, Moved Temporarily (Location)
8: HTTP GET /market/GetBinary/GetBinary/<app.packet>...
9: application

Figure 5.5 *Message flow of free application case in Play Store.*

The message flow of the free application case was conducted securely with SSL/TLS up to the log message (Message 4, Figure 5.5) and the rest of the flow was done through an insecure channel, including the actual delivery of the free application package (Messages 6 – 9). The application was never downloaded from the first given URL, which was given in the reply (Message 3) for delivery message (Message 2). Instead the user was given in the reply a *HTTP 302 - Temporarily moved* status and a new URL where the download would be completed (Message 7).

```
GET /fdfe/delivery?doc=com.rechild.advancedtaskkiller&ot=1&st=EKbsppkF%0A
&vc=10203
```

Listing 5.12 Download URL request message (Message 2, Figure 5.5)

```
HTTP/1.1 200 OK
Content-Type: application/x-gzip

KLzvvob5inySFDD_Z9onEk0PK9s
http://android.clients.google.com/market/download/Download?
packageName=com.rechild.advancedtaskkiller&versionCode=10203
&token=AOTCm0R0yTrLIiV_2m0-x_VX7ax9c_wiFqjo0jcrJZJ25KVM3WYffMG9qo8l4hCy1YmTCW
7tdDL79taYZMJuIQSiBNzUNeVN_cCIJUqR2iY
&downloadId=77927525110214843* MarketDA05261065972045272208@*
```

Listing 5.13 Reply for download URL request message (Message 3, Figure 5.5)

```
POST /fdfe/log HTTP/1.1
Content-Type: application/x-protobuf
Authorization: GoogleLogin auth=DQAAANQAAADZhuyFo7zr_MWeljM ...
Host: android.clients.google.com

?(6confirmFreeDownload?doc=com.rechild.advancedtaskkiller
```

Listing 5.14 Log message of free software case (Message 4, Figure 5.5).

Message 2 was sent after the user had decided to install the desired application. The reply in Message 3 contained a URL (which included a token) and a string value beginning with MarketDA. The next message sent from the user's device after the delivery message was Log-message containing confirmation of the free application download.

```
GET /market/download/Download?packageName=com.rechild.advanced
taskkiller&versionCode=10203&token=AOTCm0R0yTrLIiV_2m0-x_VX7ax9c_wiFqjo0jcrJZJ
25KVM3WYffMG9qo8l4hCy1YmTCW7tdDL79taYZMJuIQSiBNzUNeVN_cCIJUqR2iY
&downloadId=77927525110214843
Cookie: MarketDA=05261065972045272208
Host: android.clients.google.com
```

Listing 5.15 Download message (Message 6, Figure 5.5)

```
HTTP/1.1 302 Moved Temporarily
Location: http://r3---sn-ovgq0oxu-5goe.c.android.clients.google.com/market/
GetBinary/GetBinary/com.rechild.advancedtaskkiller/10203?ms=au&mt=1396861410
&mv=m&mws=yes&expire=1397034280&ipbits=0&ip=0.0.0.0&cp=Snp3bmFzRFk6ODQzMzY4Nzc
5MzA5NTI3NTM0MDQ&sparams=expire,ipbits,ip,q:,cp&signature=754A53FE5D8B59630EDA
7DDDA3A445C64FF79FD8.B54C0E3A652A09DC1F320773FE6948364BBA1B5F&key=am3
```

Listing 5.16 Reply for download message (Message 7, Figure 5.5)

The string value mentioned earlier in Message 3 proved to represent a HTTP cookie in Message 6, but it was not known where it was used for (e.g. authentication or authorization). Token in the URL was thought to be comparable to Auth tokens, since HTTP header did not contain Authorization field. The reply (Message 7) contained the final download URL, which included a signature parameter. Signature parameters usage was also not known. The author suspected it to have something do to with Android packet (APK) verification.

```
GET /market/GetBinary/GetBinary/com.rechild.advancedtaskkiller/10203?ms=au
&mt=1396861410&mv=m&mws=yes&expire=1397034280&ipbits=0&ip=0.0.0.0&cp=Snp3bmFzR
Fk6ODQzMzY4Nzc5MzA5NTI3NTM0MDQ&sparams=expire,ipbits,ip,q:,cp&signature=754A53
FE5D8B59630EDA7DDDA3A445C64FF79FD8.B54C0E3A652A09DC1F320773FE6948364BBA1B5F&ke
y=am3 HTTP/1.1
```

Listing 5.17 Final download message (Message 8, Figure 5.5)

Message 8 might be one of the few exceptions where no token was provided to Google in some form along the message. However, Message 8 also contained many parameters (e.g. *cp* and *mt*), whose usage were left unknown.

5.2.2 Paid application case

Paid application case introduced new messages and security mechanisms. The most essential messages are presented in Figure 5.6.

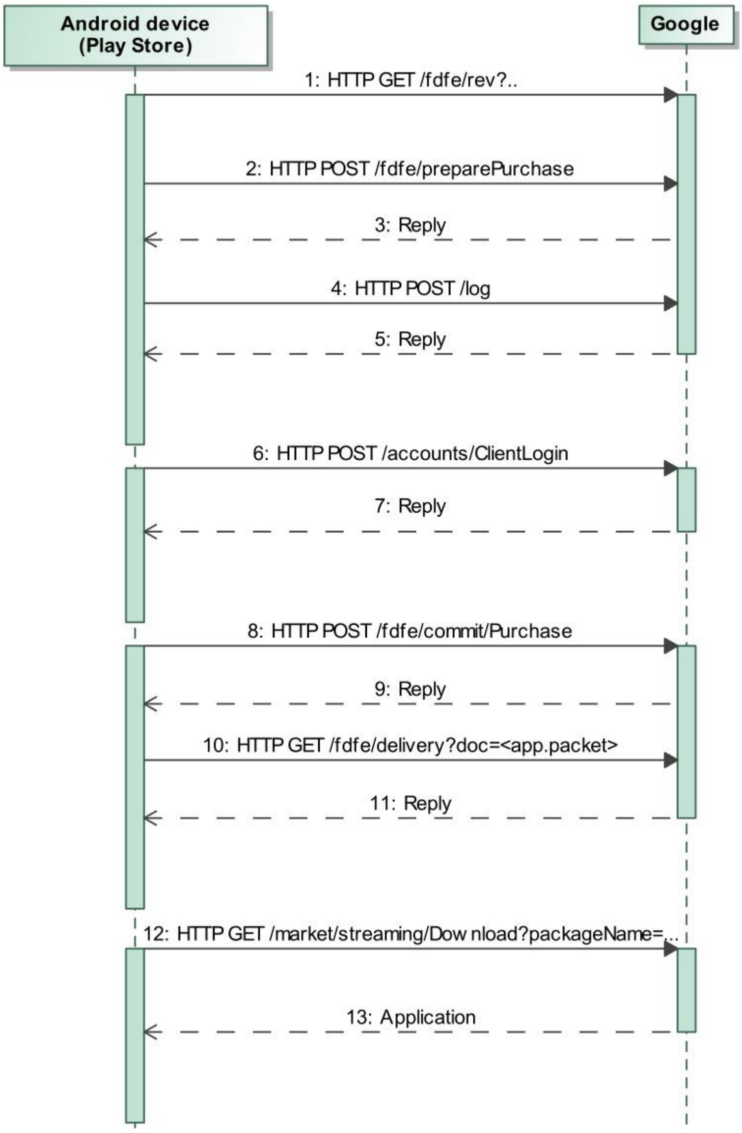

Android device
(Play Store)
Google
1: HTTP GET /fdfe/rev?..
2: HTTP POST /fdfe/preparePurchase
3: Reply
4: HTTP POST /log
5: Reply
6: HTTP POST /accounts/ClientLogin
7: Reply
8: HTTP POST /fdfe/commit/Purchase
9: Reply
10: HTTP GET /fdfe/delivery?doc=<app.packet>
11: Reply
12: HTTP GET /market/streaming/Dow nload?packageName=...
13: Application

Figure 5.6. *Message flow of paid application case in Play Store.*

The new messages in the paid application case are preparePurchase (Message 2, Figure 5.6), ClientLogin (Message 6) and commitPurchase (Message 8). New introduced security mechanisms are digital signature and authentication.

```
POST /fdfe/preparePurchase HTTP/1.1
Authorization: GoogleLogin auth=DQAAANQAAADZhuyFo7zr_...
Host: android.clients.google.com

ot=1&doc=radiotime.player&pcauth=4&vc=134&ct=dummy-token
&dcbch=NxFayPdjWuQoiozgycXNzFWZnEs&
```

Listing 5.18 PreparePuchase, (Message 2, Figure 5.6).

```
HTTP/1.1 200 OK
X-DFE-Content-Length: 4882

TuneIn Radio Pro2.88
* ACKDxPPf5Ck7idZZFguFxR/hdHaeAEvajB5RzFqbNeWNhIkcT+3Yc1npgwyy5WwbfFmtvZVCQdq+
DNeKM60rA4oZnpaTxuY3nHyIofXjYWTh/lD+dH9IR/1CcY7DIBbA4bNw/K3EzENMd79bvCqKHmsX/v
M8Lo7VssAfys3p8b6qaq3oqbXMKHVl0kVt2lzmWbNXDlUKMvWbCvNogzlWrN67Xz8EWNWiXm6JjXs+
... Bnn0=":
'15218670163581205722.D.15377127500743620B     Visa-4670
*SCurrency fluctuations, bank fees and applicable taxes may change your final
amount.2By tapping "Buy", you agree to the Google Wallet Terms of Service …
```

Listing 5.19 Reply message for preparePurchase (Message 3, Figure 5.6).

```
POST /fdfe/log HTTP/1.1
Content-Type: application/x-protobuf
Authorization: GoogleLogin auth=DQAAANQAAADZhuyFo7zr_MWeljMFno8 ...
Host: android.clients.google.com

.(%completePurchase?doc=radiotime.player
```

Listing 5.20 Log (Message 4, Figure 5.6)

The preparePurchase message was sent right after the user decided to install the desired application. On the surface the message did not contain anything new or particularly interesting, but it did contain parameters in the message body that were not understood. The reply (Message 3, Figure 5.6) contained among other things in one capture a 5432 character long base64 encoded string. An attempt to decode the string did not reveal anything understandable text, which hinted that the contents might be encrypted or binary and therefore the content was left unknown. The next message sent from the user's device after the preparePurchase was Log-message containing confirmation of the application purchase.

```
POST /accounts/ClientLogin HTTP/1.1
Content-Length: 125
Host: www.google.com

service=apps&accountType=HOSTED_OR_GOOGLE&source=Google-GooglePlay-80260017
&Email=****@gmail.com&Passwd=***** &
```

Listing 5.21 ClientLogin authentication request (Message 6, Figure 5.6)

```
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 881

SID=DQAAAMsAAACoBG ...
LSID=DQAAAM0AAADfT ...
Auth=DQAAAMwAAADfTnvm ...
```

Listing 5.22 ClientLogin authentication reply (Message 7, Figure 5.6)

After deciding to install the application, Play Store presented a UI interface, where the user was requested to authenticate with her Google email address and password. Play Store used mechanism called ClientLogin to authenticate the user. One notable feature of the ClientLogin mechanism is that the user's password is communicated in plain text.

Play Store's usage of ClientLogin was observed to be more straightforward than depicted in chapter 4.2.2, it comprised only steps 1 – 2 and 7 presented in Figure 4.2. In a successful authentication the reply's (Message 7) message body looked similar to a GLS token reply (Listing 5.5), which contained SID, LSID and Auth tokens. These tokens were not observed to be used at all during the communication, which hinted that even though ClientLogin is designed mainly as an authorization mechanism [34], Play Store used it for only authentication. ClientLogin authentication request message (Message 6) is one of the few exceptions that did not contain a token within the message.

It should be noted that when buying application from Play Store the message flow included messages and parts of messages, that were not fully understood. Therefore it is possible that token provided in Message 7 could have been used and such ClientLogin could have been used more than just for authentication.

```
POST /fdfe/commitPurchase HTTP/1.1
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Authorization: GoogleLogin auth=DQAAANQAAADZhuyFo7zr_...
X-DFE-Signature-Request: nonce=jBha_12piAPP4w…
X-DFE-Request-Params: timeoutMs=35000
Content-Length: 5848

pcauth=2&pcarc=0&pct=ACKDxPP   ...   Bnn0=&ct=dummy-token
&chdi=qIHI3OWg3mesmCxcMsK75RKu7F4&
```

Listing 5.23 Commit purchase message (Message 8, Figure 5.6)

```
HTTP/1.1 200 OK
X-DFE-Signature-Response: signature=ABG4_WXBR93mozUv…
X-DFE-Content-Length: 538

radiotime.player0*(QtDMGv49qQg-UWD2tNhCmW8F00M(         623"cSpa8cWPXSx2cPY7u-
DzpeSktnQ
https://android.clients.google.com/market/streaming/Download?packageName=radio
time.player&versionCode=134&token=AOTCm0RjPvi90PsYWXQ-FQZVHXpeIysIdSILJLJYjrA_
PA5bw5mo4oltsdfKshNv1huHScCTo9_IXqsIdbM52dw7W7xpUYW9r9d1AnR5dPQ&ev=1
```

```
&downloadId=6890805265695851619*MarketDA05261065972045272208O@bJ9GYLSn7DRCHRWp
nLWQ1Ohg==,mEz0TaKr6bILmoIr30cTKCAKPF+YTPRNoqvpsguagis=*...
```

Listing 5.24 Reply for commit purchase message (Message 9, Figure 5.6)

Right after the reply for the authentication (Message 7) was received, a commitPurchase message (Message 8) is sent. It includes a signature request and in the message body the same base64 encoded string (parameter pct in Listing 5.23), which was provided in the reply (Message 3) for the preparePurchase message (Listing 5.19). The message body also included parameters whose usage was not understood, such as, *pcauth*, *pcarc*, *ct* and *chdi*. Signature request's purpose is presumed to authenticate the other communication party to be genuine Google service.

The reply (Message 9) for commitPurchase message contained the response for the signature request and download URL (with token) similar the one provided in the free application delivery message (Message 2, Figure 5.5).

```
GET /fdfe/delivery?doc=radiotime.player&ot=1&st=EKbsppkF%0A&vc=134 HTTP/1.1
Authorization: GoogleLogin auth=DQAAANQAAADZhuyFo7zr_MWelj...
Host: android.clients.google.com
```

Listing 5.25 Download URL request message (Message 10, Figure 5.6)

```
HTTP/1.1 200 OK
X-DFE-Content-Length: 406

...
cSpa8cWPXSx2cPY7u-DzpeSktnQ
https://android.clients.google.com/market/streaming/Download?
packageName=radiotime.player&versionCode=134&token=AOTCm0TIRIXQofTyyCc7EC2IE2C
a8FGeb15iyfb8iK7eSmci77l8ZYXrywRuNjhofg1Lp5bvO8xGHhBylE9p2nXQYC0fOD_cQOYoZQWUf
RA&ev=1&downloadId=-2412412061604833543*
MarketDA05261065972045272208O@bJibg7Wl2WjHJs+hGvBpiBxg==,
CC/W5vfHdY4dQMS8Wgw88RrMgNcIL9bm98d1jh1AxLw=*...
```

Listing 5.26 Reply for download URL request message (Message 11, Figure 5.6)

Even though the URL given in Message 9 looked like the application download URL, it was not later on used to download the application. The final download URL was delivered in the next message exchange (Messages 10 – 11, Figure 5.6). The reply for the delivery message (Listing 5.26) looked almost same as in Listing 5.13 in the free application case, except in this case the MarketDA string had next to it two base64 encoded strings, whose usage was left unknown.

5.2.3 Miscellaneous findings

There are a few interesting messages sent by the Play Store's client application from the user's device during the Play Store's start up. Messages (not shown in any figure) are: POST /fdfe/replicateLibrary and POST /fdfe/bulkDetails. The interesting part of replicateLibrary message contained, in addition to the authorization, also a signature request in HTTP header. Finally the time-out values are considered.

```
POST /fdfe/replicateLibrary HTTP/1.1
Authorization:GoogleLogin auth=DQAAANQAAADZhuyFo7zr_MWeljMFno...
X-DFE-Signature-Request: nonce=_hMNviWlui56GEmdvIscAnv-EB2wanBgG 0P7_w-1dDQWe4
uaVNVtzF0nuOUrJTJA3VxtLe_AEgwSpIry7tVzS0wbBUxFLrS8LMLnlT9HZg5e_6ZBhdSvLP768LYP
15Kpi8v0eMh1nVuHQK6Pi2K3woqTMD8MwQxHXaXAIh7vAf-5VU3FVT8cfyDGKk0THoc19M_zf6s5eq
qXFWyZ6-Jwf4fGWEUICkACk6sQ17AphrAx0McsKVklkxGzel6WapHqeKVU8ugQ88C2YLlQnOLEGWqS
IWwFWzcela2Q-Jz0CCYuRPyrXUKJ-cyWpjZbVyw4_ZJS8PDFE5hYOoAbZrlFTA
X-DFE-Request-Params: timeoutMs=30000
Host: android.clients.google.com
```

Listing 5.27 replicateLibrary message.

```
HTTP/1.1 200 OK
X-DFE-Signature-Response: signature=ABG4_WVWLK2s4PZa5UgRIgciiNGO0bi0RedinvOUkM
6FuCrghwt-TAXTnmwtlYUHas03LXr44B4ILkpno3v4EQg2PcUHDmRqL736MbbUGW_qmAOPujEZXVB-
JcWen8t1iBc0gJmo132AinmewYNwgrUZ6kAEkYa4xcVSP9DBEpfhiMwRJ7LrTRh9e2EIz8_zNpPCjg
Txikuh_PyP12Ukk_q-hNrRonqe9pitWLLsRXgXobgE0m7n24iz0YkSfoG1vQGW5_756gPvzaaU7vxp
yaMALVt8FM9a5HGGrkFSuFNa2Va8bOrUazHJdJRdZYmhP8MqLPk41BZfa44897V4eUeSoTCB
```

Listing 5.28 Reply for replicateLibrary message.

The replicateLibrary message's HTTP header X-DFE-Signature-Request contained a long nonce value encoded with base64URL and the reponse (Listing 5.28) was also encoded in base64URL. Since digital signatures are used for authentication it was thought that the signature request is one of the Play Store client application's security mechanisms used to authenticate the other communication party, in this case Google's server.

The bulkDetails message (Listing 5.29) contained in the HTTP header, among other things, the authorization token. The interesting part was in the message body, it contained a list of applications found on the mobile device.

```
POST /fdfe/bulkDetails HTTP/1.1
Authorization: GoogleLogin auth=DQAAANQAAADZhuyFo7zr_MWeljMFn...
X-DFE-Request-Params: timeoutMs=30000
Content-Length: 717
Host: android.clients.google.com

com.android.chrome
com.google.android.apps.plus
com.google.android.gm
com.google.android.gms
com.google.android.googlequicksearchbox
...
```

Listing 5.29 bulkDetails message.

```
HTTP/1.1 200 OK
X-DFE-Soft-TTL: 7200000
X-DFE-Hard-TTL: 7200000
```

```
com.android.chromecom.android.chrome    *Chrome    Browser    -    Google2Google
Inc.BUSDFree(@J(KQ5,"$2X`pLhR`*Zhttps://lh5.ggpht.com/NBr8wEdoQqZxBMblJWDPN6HI
C3xcEGYM_eEgNbk2iL_GTIUWBVfNRl97bLQFjPBb9FmEHR^*Xhttps://lh3.ggpht.com/_eS2pfW
StKFMufgCbeIAyWmTbhpu_iploEyRdoffHtpdMVtOImS_MMrhAyu949FoVQHj

pHR)android.permission.ACCESS_COARSE_LOCATIONR'android.permission.ACCESS_FINE_
LOCATIONR'android.permission.ACCESS_NETWORK_STATER$android.permission.
ACCESS_WIFI_STATERandroid.permission.CAMERARandroid.permission.GET_ACCOUNTSR
android.permission.INTERNETR"android.permission.MANAGE_ACCOUNTSR
(android.permission.MODIFY_AUDIO_SETTINGSRandroid.permission.NFCR%
... (rest of the permissions omitted)
4 Apr 2014APPLICATIONr(t@i (08@Ca)@zB
Top DeveloperL *B
http://www.gstatic.com/android/market_images/badges/topdev_ann.pngM *C
http://www.gstatic.com/android/market_images/badges/topdev_list.pngL !*B
http://www.gstatic.com/android/market_images/badges/topdev_hdr.png
"Top Developersdetails?doc=com.android.chrome@
https://play.google.com/store/apps/details?id=com.android.chromedetails?
doc=com.android.chrome
...(rest of the message body omitted)
```

Listing 5.30 Reply for bulkDetails message.

The reply for the bulkDetails contained such HTTP header fields as *X-DFE-Hard-TTL*, which was thought to be some kind of time-to-live value for the information provided in the message body. The reply message's message body contained, for example, permissions for every requested application. Listing 5.30 shows only the information provided for one application.

Play Store client application was also observed and presumed to use time-out values in the HTTP header to inform the server, how much time it had to answer to the request. Play Store client application conveyed this information in the HTTP header field named X-DFE-Request-Params. For example, with a value timeoutMs=30000. Play Store was observed to use values 2500, 30000 and 35000, that were thought to represent 2.5 – 35 seconds.

5.3 Discussion

This section first covers the relationship with the GLS, GA and GPS. Then it moves telling the odd finding during the observations, which hindered at first analysis of Google's system. Section concludes with describing the previous work done in the subject.

5.3.1 Relationship with GPS, GLS & GA

The two observed user-agents in HTTP header hinted that Android has two services or at least different codes bases that provide tokens for mobile applications for the Google's resources. There is very little or none reliable information in public how Google's own applications authentication and authorization works. Google provides documentation for several methods [5], but these are meant for third-party developers and Google's applications were observed to use these differently or none at all.

From the author's own observations the HTTP user-agent GA in the token acquirement is probably actually a integrate part of GPS, because GPS is the only publicly known Google Android mobile application service that provides OAuth 2.0 tokens, although it provides for third-party applications. Shiram's [39] work also supports this speculation. Also from the Ars Technica article [33] it is known Google's tight integration of its applications to GPS and from observations it is known many of Google's own applications to use OAuth 2.0 tokens provided by GA HTTP user-agent.

According to Elenkov [28] both GLS and GPS provide authentication. For example, GLS is responsible when user's Google account is added to Android device and GPS for OAuth 2.0 authorization and Google+ social media service sign-in [28]. Then again according to Ars Technica [33] GPS was held responsible of e.g. initial account setup, account authentication and account syncing. This contradiction in GLS and GPS responsibilities between Elenkov and Ars Technica is probably due the fact that Nelenkov's blog post was done in November 2012, a month after GPS was announced [41]. And Ars Technica's article was done a year later in September 2013 [33], when Google had already integrated many its services and applications under GMS [31, 33], which consists of GPS and Play Store [31].

Author's own observations support neither Elenkov nor Ars Tecnica. In the observations the master token request was always the user-agent GLS, but in the reply message (Listing 5.3) the master token value begins with "oauth2", which suggests that the GA had made the request.

Shiram's [39] work might shed some light in the matter. According to him all the critical code related to Google OAuth flow runs only within the com.google.android.gms application (.auth.GetToken service), which is signed by Google. The service approves applications locally by their signature and package name and uses the master to obtain access tokens. [39] The OAuth flow in a Google Android device is presented in the Figure 5.7.

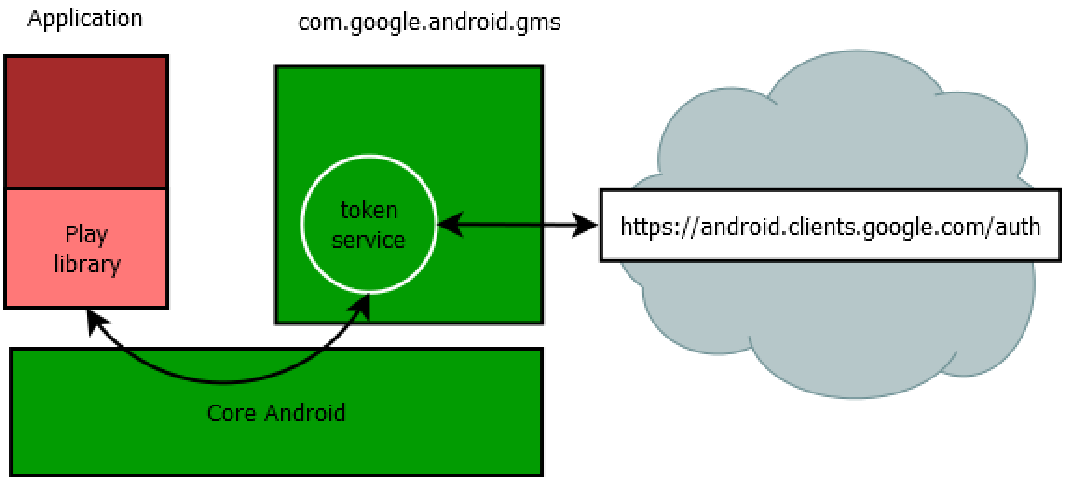


Figure 5.7 *Google OAuth token flow in a Google Android device* [39].

In the Figure 5.7 the green areas run trusted code. Shiram also says that the com.google.android.gms application is actually the “Google Play Services” application, which holds the token service. The application painted in red in the Figure 5.x is a third-party application, which uses a Google Play Services library. The library does not run critical code and it just forwards call to services and activities running within a GPS application. [39] Although the Figure 5.7 depicts a third party application, author personally believes this flow is applicable with Google’s own applications.

Shiram’s [39] work describes the master token request and two OAuth token requests with their respective parameters, but Shiram had not included the user-agent in any of token requests he presented. The master token request was identical to authror’s observations. One of the OAuth token request Shiram presented was identical to author’s observations of the GLS token request and the second OAuth token request resembles very closely to a GA token request. Author believes the differences observed in between the GA token request and Shiram’s example stems from the fact that Shiram describes a request for a third-party application.

This hints that the GLS and GA both are now fully integrated in the GPS. This is the only logical explanation that author can think of and which explains author’s observations and is still in-line with Shiram’s work, which is more recent than Elenkov’s and Ars Technica’s.

5.3.2 Legacy names

One oddity found during the observations was in the HTTP header Authorization, which value contained a token and a name for a security mechanism. This was first thought to be confusing, because the observed behaviour of authentication and authorization from the captured communications was compared to the publicly available documentation (e.g. AuthSub [42]) of the protocol and these two did not match.

```
POST /gcm/groups HTTP/1.1
Authorization: AuthSub token=ya29.1.AADtN_X-7HE6...
```

Later on it was understood that the name of the mechanism the header value did not always tell how the token was acquired. It was thought that at some part of time Google integrated authorization mechanisms and mechanism names were left as a legacy from the past.

5.3.3 Previous work

Previous work on how Google uses tokens in Android has been done by Nikolay Elenkov [28, 43], KB Shiram [39] and Korean Android community has reverse engineered Google GMS [44, 45]. Compared to this thesis Elenkov’s work is broader. He mainly concentrates on the Android device, which was considered out of scope in this thesis, but also covers communications security. For example, Elenkov describes how GLS has been implemented in different Android versions and discusses related security aspects (e.g. how user's password has been encrypted in the device, etc.) [28, 43]. Elenkov's work has also been written from different point of view as he has been concentrating to the third party developer's point of view (e.g. how to use weblogin mechanism for single sign-on). This thesis has been written solely from the vendor's (Google and Amazon) point of view. For these reasons Elenkov's work and this thesis have only a few common findings: the master token and weblogin mechanism.

Elenkov reported Android OS 4.0 and newer versions get during the user's account registration a master token, which is then used to obtain new tokens [28]. This finding agrees with authors own observations with Android OS 4.3 version device. Elenkov also describes how the new tokens are acquired in devices using operating system older than Android 4.0, but this thesis cannot confirm this particular finding, because only a single mobile device with Android OS version 4.3 was used during observations.

Elenkov presented in his work how weblogin mechanism [43] that could be used to single sign-on and essentially to authenticate a user in a third party applications. In author's own observations weblogin mechanism was observed, but it was used in different context and its working (especially HTTP cookies) was not fully understood. Therefore it is not sensible to compare Elenkov's finding to author's observations.

Shiram's work also concentrates more on the Android device than what happens in the communication channel, but his work shed light on questions that are not possible to find out by merely looking at the communication channel. Especially Shiram's work on how an application in a Google Android device gets access tokens was useful in order to understand the relationship with GPS, GLS and GA. Shiram had also described the master token request and OAuth token requests, which were very similar to author's own findings.

Korean Android community has done reverse engineering, especially, on Google GMS [44, 45], which provides some hints on how Google's applications work. However, the documentations are outdated, since the most recent available works are from year 2011. For example, the author's observations regarding on how Play Store works did not match with the documentation.

6 AMAZON RESULTS

Chapter starts by presenting how Amazon acquires, uses and renews tokens. After that Amazon's Market is studied and the chapter concludes by discussing on findings. All listings in this chapter hava been modified (e.g. parts omitted, bolded, etc.) for readability.

6.1 Tokens, acquisition and signatures

Amazon's applications also used tokens heavily for security. And like with Google, it can be assumed that a token was sent in along every message unless otherwise explicitly stated.

During the observations 3 different tokens were observed: refresh, access and x-adp token. These are presented in Listing 6.1 together with three different HTTP cookies, which were observed to be used as tokens. As with Google's HTTP cookie tokens, they are not addressed in depth, because their usage and roles were not understood.

Tokens were observed to be delivered in 2 different methods. The first is a HTTP header with two different field names: *Authorization* and a custom header field *x-adp-token* (or X-ADP-Authentication-Token). The second method is passing the token in a HTML Form.

From "normal" tokens the refresh and access token's appearances are almost identical. The x-adp-token appearance was different from the others and also it was the only token, whose structure and appearance hinted at the usage of encryption

```
refresh token:
Atnr|EQEBLjAsAhQVKZUM7BpmmE5Ypqifd1_KuookkgIUKjyZHNDDM2YbA_wTAv3duSX0Yj1n0M0Vq
lVRzIU-gWe1dm1KuT4o-Ku2B1bqlJNhg6MpOA5ZR5sH0PqUUr0kMok1tixedfvw5q0Nz9FWEqnOZru
FkmYH9I7LFB5XXHULzvjXTiHYYLaA6V-1Syx8sd0ZRG8CpZY0B_M_KuFwYvYxmZHF1A

access token:
Atna|EQEBLjAsAhQBvlzt4dgbdCS729ZuPMujFv86TwIUQZKdFCFxdKgZ8ew_ViydoMZ_-J5k14c4L
ie-IDRG_uc7iKWsWZkL2O_30N050_BdNZKR-9ne7T9dYYrfcLT71FG7Hw4j71aobE8XVyiITD6RB1f
Zvdu5ZHk4PVpTCzFEE0cirxO__UHNYhlLX44SMx_79dcImbVHmv29tqEO12AOABguQg

x-adp-token:
{enc:O5R4tVC8 ...}
{key:CprPB3Ja ...}
{iv:aS/kZh79x+F2V9rsYPLtBQ==}
{name:QURQVG9rZW5FbmNyeXB0aW9uS2V5}
{serial:Mg==}

x-main:
EUboCL9TgRR8dCZw375umy2Cddx?sf5RoNQw0rOSwi@EXlCGKlHjctz?6w1q0Nd8
```

```
at-main:
5|BbQsv+cew0+OHJDbQUuMhSqsiXHWmmCwKjGjHyJFGMqgIrfu7pZ+2EWlRbjtJmc3leZnLb5HAuvR
R3aRAqvuFa0vfzwMP0aCToa+9XfiXbkBSKvKlIz+TPInosuHN1Z5LgKkoBhCQNccIbP/gMjL2qTYkU
rl3xyRjPPZ0avesUr1AtF175poQqTfUvOxyyxVXC+w7DG8uxUzJMw4AlPseu/uOqinOh7u

sess-at-main:
u0Po/R0HYUDcqzBYiOLbcfL84dfkuk3bBWsER4AiIxk=
```

Listing 6.1 Examples of observed tokens.

As with Google's token, Amazon's tokens were also observed to be used only for authorization and it is not known whether tokens had information included within them in some way, except the x-adp-token.

Refresh token is used in acquiring new access tokens. The token acquirement processes are covered in Sections 6.1.4 and 6.1.5. Access token and x-adp token are similar in the way that both are used just for authorization and are pointed to specific applications. The difference between these two is in expiry. Access tokens have a defined expiry time, which is given when the token is acquired. X-adp-token on the other hand was observed to change only when the user registered again to the device.

In general, the acquirement and usage of tokens of Amazon's client applications happens in two to three phases. Two phase usage concerns x-adp-tokens and third phase for the rest. X-adp-tokens are first acquired during the user registration to the device and then then the token is used. For the rest, refresh token is acquired during the registration, then it is used to acquire access tokens and finally the application uses the access tokens.

The token acquirement and usage, especially, during the registration is a very confusing process. For example, tokens are acquired in many different messages, the same messages are sent multiple times, certain messages have exceptions and the given tokens might be even used only once, etc. In general, the final x-adp-token is given in the reply message for the /Firsproxy/registerDevice message. Access tokens and HTTP cookie tokens are acquired in messages /ap/exchangetoken and /ap/exchangetoken/cookies respectively. Refresh token is acquired during the registration in an exception case of /ap/exchangetoken message.

The rest of this section presents x-adp-token in depth, and then covers signatures, which were nearly always present together with x-adp-tokens. Next, a part of token acquirement during the registration is covered and it is continued in the following sections regarding token and cookie token acquirement. Finally, the token request parameters are presented.

6.1.1 x-adp-token

X-adp-token, which is sometimes also named as X-ADP-Authentication-Token (Listing 6.1), is the only observed token where the structure and appearance clearly indicated the usage of encryption. The token's name is taken straight from the custom HTTP header field name. The name of the HTTP header field did not matter to outward appearance of the token. The token consisted of five parts separated by curly brackets: *enc*, *key*, *iv*, *name* and *serial*. In all parts the values were base64 encoded. *Enc*, *key* and *iv* base64 decoded values were not successfully decoded and decrypted, so their contents were left unknown, but *name* and *serial* parts were. Name decoded into *ADPTokenEncryptionKey* and *serial* into *2*.

The length of each part of the encrypted tokens was observed to be always the same. Enc part contained 960 character long base64 string which decoded into 720 bytes of data. Key and iv were respectively 344 and 24 bytes long base64 strings and decoded into 256 and 16 bytes of data.

The same encrypted token values were observed to be used multiple times, which made it possible to follow the token usage and acquirement. This does not confirm, but suggests that tokens were encrypted only once before usage and that the same token was used until it was expired. The author's own speculation regarding this token's encryption is discussed in Section 6.3.1.

6.1.2 Signature

X-adp-tokens were every time (with one exception) observed to appear with possibly two other security methods: digital signature and cryptographic hash function (Listings 6.2. and 6.3). Digital signature appeared in a custom HTTP header: x-adp-signature. Another custom HTTP header x-adp-alg was thought to contain the information regarding to what signature algorithm was used, since value SHA256WithRSA refers to a known algorithm defined in PKCS #1 standard [46].

```
x-adp-signature: Ilro59XAPWWPeFip7s1cqsaDG0AatUie+p7oVKvomG4o1LLLazeXqFUl9j91B
ZwnumMEg0f5E4tedYglWWRei+2MrKVMPfpgV2IlFZzws4Tw2f23qp+zyaeCVFkMHnVAdRsUBOwIsL0
M92QGvmHUK1NObNg3HpRDn0a1GEwGGjcpPs1yxR3/PIImuJWgKPxJxv1WOEz5UKiEAYuqxZU4eJhya
h5LylqDMV1qZmcMgDFZBZcG6S7fj4WHZ5aWF5Gq:2014-04-24T15:52:37Z
x-adp-alg: SHA256WithRSA:1.0
```

Listing 6.2 x-adp-signature example.

Signature consists of timestamp and 256 character long base64 encoded string, which decodes into 192 bytes of data. It is not known what parts of the message are included in the calculation of signature, nor the keys used in the algorithm.

```
X-ADP-Request-Digest:
Hl6bik/S4im03elWAWIo+Fxj/jQjb8Gv4KUe7KYkmGH9kF5iBxZ1/GfOU/B35uoWkb4tM41Zf/bNI1
zt/738M7gYFSRiHGRbgaLhU1K6WWTPfurJWvVdDQzLlh54sPK+MHnFYDbykEGd2VFbPmVZ0rO09PaN
tXMrzR3Y4ZtE+WdE18XHExNEHqsRuMy2LHjTHIMYw8CS91g26L+RKPUtjCYKY/mKGM9nj+nb5lspX9
xSQx31WsHyCmIuul+ees8Q:2014-04-24T15:52:58Z
```

Listing 6.3 x-adp-request-digest example.

The outward appearance of the HTTP field X-ADP-Request-Digest is similar to x-adp-signature. Both fields have similar timestamp and base64 encoded string with equal length. Only two things suggest that there are differences between these two fields. First the HTTP field named x-adp-alg does not appear with X-ADP-Request-Digest. Second a part of HTTP field name (digest) refers to a cryptographic hash function's output. It is not known which hash function was used, nor what parts of the HTTP message are included in the calculation of the hash function. The author's own speculation regarding this signature and digest value is discussed more in Section 6.3.2.

6.1.3 Initial token acquisition

When user's account was added to the tablet during a registration all tokens were acquired in five phases (Figure 6.1). Figure 6.1 shows only one message of each phase, but in practice the same message was sent 2 – 4 times (registerDevice -message being an exception) and each time new tokens were issued.

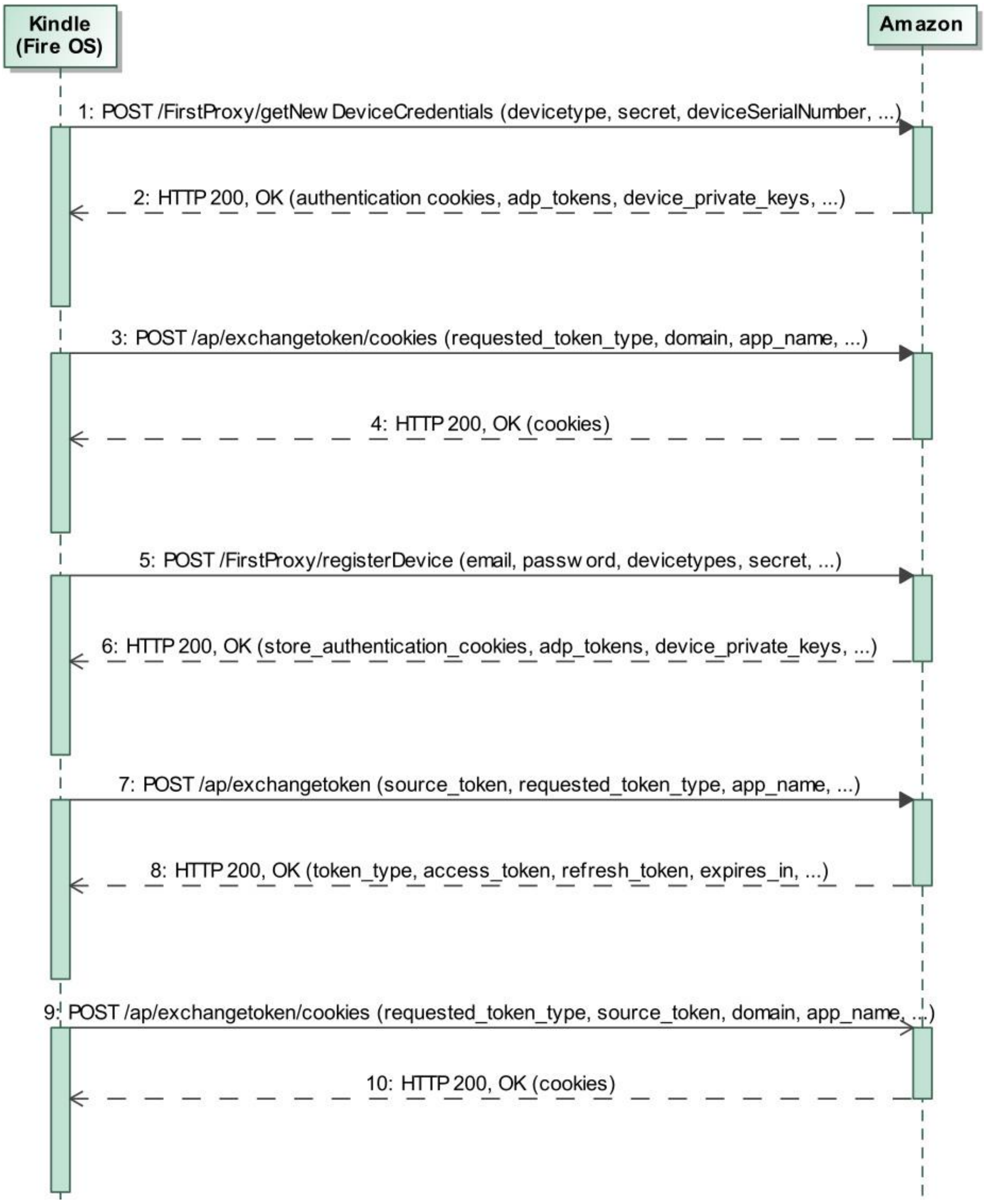

Kindle (Fire OS)
Amazon
1: POST /FirstProxy/getNew DeviceCredentials (devicetype, secret, deviceSerialNumber, ...)
2: HTTP 200, OK (authentication cookies, adp_tokens, device_private_keys, ...)
3: POST /ap/exchangetoken/cookies (requested_token_type, domain, app_name, ...)
4: HTTP 200, OK (cookies)
5: POST /FirstProxy/registerDevice (email, passw ord, devicetypes, secret, ...)
6: HTTP 200, OK (store_authentication_cookies, adp_tokens, device_private_keys, ...)
7: POST /ap/exchangetoken (source_token, requested_token_type, app_name, ...)
8: HTTP 200, OK (token_type, access_token, refresh_token, expires_in, ...)
9: POST /ap/exchangetoken/cookies (requested_token_type, source_token, domain, app_name, ...)
10: HTTP 200, OK (cookies)

Figure 6.1 *Amazon token exchanges when adding a user account.*

This subsection considers only getNewDeviceCredentials and registerDevice messages. Exchangetoken and exchangetoken/cookies messages are examined in the next subsection, because the messages used during the registration are exception cases.

```
POST /FirsProxy/getNewDeviceCredentials?deviceType=A2VZ790DVVI91K&deviceSeria
lNumber=D0FBA0A034530WWC&secret=B3KXAQO3AJNZGB7Y170D&radioId=00bb3ab377ce&reas
on=NoState&softwareVersion=323001720&softwareComponentId=com.amazon.thor.andro
id.os HTTP/1.1
X-Amzn-RequestId: e972d391-52ee-40c2-a242-ac2b3e6ecd1c
Content-Type: text/xml
User-Agent: Dalvik/1.6.0 (Linux; U; Android 4.2.2; KFTHWI Build/JDQ39)
Host: firs-ta-g7g.amazon.com
Content-Length: 612

<?xml
    <request>
        <deviceTypeSoftwareVersionMap>
            <entry
                deviceType="AZ9LILQBO9I6H"
                version="38038210"
                softwareComponentId="com.amazon.cloud9"/>
            <entry
                deviceType="A225F6K82YR2UO"
                version="1710016310"
                softwareComponentId="com.audible.application.kindle"/>
            <entry
                deviceType="AYNDLAEFR9H1C"
                version="4021210"
                softwareComponentId="com.amazon.mp3"/>
            <entry
                deviceType="A2VZ790DVVI91K"
                version="323001720"
                softwareComponentId="com.amazon.thor.android.os"/>
            <entry
                deviceType="AXRZR9ASDFH6P"
                version="30208810"
                softwareComponentId="com.amazon.ags.app"/>
        </deviceTypeSoftwareVersionMap>
    </request>
```

Listing 6.4 getNewDeviceCredentials request message.

```
HTTP/1.1 200 OK
Server: Amazon Web Server
Content-Type: text/xml;charset=UTF-8

<?xml
    <response>
        <deviceCredentials>
            <deviceCredential
                deviceType="A2VZ790DVVI91K">
                <store_authentication_cookie>
            0rLs0KcpMz4VcIsfGsFXHEdScn0K4oowDhvZ6vSuIofDqdtITGRCgk3pND
            vB4YDLPJVjr9NG5Jnq8Nx+HIMe4udMmWZJINsxhQ7kEDprykkSZRF9MABD
            v1vQCqX6rihtpKMoDTuJXzs=
            </store_authentication_cookie>
        <device_private_key>
            MIIDmAIBADANBgkqhkiG9w0BAQEFAASCA4IwggN+AgEAAoHBANKd/Jbn3P
            1nH2N8rb8RfnSFVXXj7p/H10vchArbWAEBDEJAWwrYE6cFMXWDED1Cx5G5
            xxq1yciki7ueaGC4hEWqc+9HLundW/WZKvzq9BQj4OcDUKDjb5xtgep4R
            Tp78PKzsPOwCj2XOY/o03/baL3Kocjjz//wExmBfttgle4eY1rB3U...
            </device_private_key>
     <adp_token>
            {enc:XCUbd2MwQEjucIIUTH3SNx6Qgf...
            </adp_token>
        </deviceCredential>
    </deviceCredentials>
<adp_token>
  {enc:XCUbd2MwQEjucIIUTH3SNx6Qgf...
    </adp_token>
<store_authentication_cookie>
  0rLs0KcpMz4VcIsfGsFXHEdScn0K4oo...
    </store_authentication_cookie>
<device_private_key>
     MIIDmAIBADANBgkqhkiG9w0BAQEF...
    </device_private_key>
<cookies>
    <cookie>
        <url>
            .amazon.ru
            </url>
        <value>
            x-fsn=0rLs0KcpMz4VcIsfGsFXHEdScn0K4oowDhvZ6vSuIofDqdtITGRC
            gk3pNDvB4YDLPJVjr9NG5Jnq8Nx+HIMe4udMmWZJINsxhQ7kEDprykkSZR
            F9MABDv1vQCqX6rihtpKMoDTuJXzs=;  expires="Mon,  05-Jun-2034
            11:47:37 GMT"; domain=.amazon.ru; path=/; secure
            </value>
        </cookie>
       ... (rest of cookies omitted)
       </cookies>
</response>
```

Listing 6.5 getNewDeviceCredentials reply message

In Listing 6.4 getNewDeviceCredientials request message was one of the first messages sent during the registration. The message included information such as: device serial number and device type that referred to a specific Amazon application in the device. Even though the request message's payload contained more than one device type, the reply message (Listing 6.5) contained tokens for only one device type specified in the request message's URL. The Reply message contained the following tokens: store authentication cookies, device private keys and adp-tokens, and cookies for different Amazon top level domains (e.g. amazon.co.uk, amazon.com, amazon.de, etc).

```
POST /FirsProxy/registerDevice HTTP/1.1
X-Amzn-RequestId: d9d1b83c-0f55-45d3-ae1b-7ae952e61c0e
```

```
Content-Type: text/xml
User-Agent: Dalvik/1.6.0 (Linux; U; Android 4.2.2; KFTHWI Build/JDQ39)
Host: firs-ta-g7g.amazon.com
Content-Length: 970

    <?xml
        <request>
            <parameters>
                <deviceType>
                    A2VZ790DVVI91K
                    </deviceType>
                <deviceSerialNumber>
                    D0FBA0A034530WWC
                    </deviceSerialNumber>
                <pid>
                    CD411901
                    </pid>
                <email>
                    *****@gmail.com
                    </email>
                <password>
                    *****
                    </password>
                <secret>
                    B3KXAQO3AJNZGB7Y170D
                    </secret>
                <softwareVersion>
                    323001720
                    </softwareVersion>
                <softwareComponentId>
                    com.amazon.thor.android.os
                    </softwareComponentId>
                </parameters>
            <deviceTypeSoftwareVersionMap>
            (omitted, same deviceTypeSoftwareVersionMap as in Listing 6.1)
        </request>
```

Listing 6.6 registerDevice request message.

```
HTTP/1.1 200 OK
Server: Amazon Web Server
Content-Type: text/xml;charset=UTF-8

<?xml
    <response>
        <deviceCredentials>
            <deviceCredential
                deviceType="AYNDLAEFR9H1C">
                <store_authentication_cookie>
                    SbkHfZXYg8/a69aEEVe5766+uda...
                    </store_authentication_cookie>
                <device_private_key
                    refDeviceType="A2VZ790DVVI91K"/>
                <adp_token>
                    {enc:yAmMyEhitIhownq3P2udr...
                    </adp_token>
                </deviceCredential>
```

```
   (rest of the deviceCredentials entries omitted)
        </deviceCredentials>
    <adp_token>
          (enc:O5R4tVC8+FMNe61lpl3e2...
           </adp_token>
    <store_authentication_cookie>
      vbM4lme+gdD3tPYe7G4rr6648zXTp...
        </store_authentication_cookie>
    <device_private_key>
          MIIDlQIBADANBgkqh...
        </device_private_key>
    <given_name>
        Tuomo
        </given_name>
    <name>
        Tuomo Tutkija
        </name>
    <account_pool>
        Amazon
        </account_pool>
    <country_of_residence>
        FI
        <source_of_cor>
            CUSTOMER_COUNTRY_OF_RESIDENCE
            </source_of_cor>
        </country_of_residence>
    <preferred_marketplace>
        ATVPDKIKX0DER
        </preferred_marketplace>
    <alias>
        ******
        </alias>
    <kindle_email_address>
        ******@kindle.com
        </kindle_email_address>
    <user_directed_id>
        amzn1.account.AGUZKC7EKGQVARWZV3Z3ZIVOPOMA
        </user_directed_id>
    <user_device_name>
        Tuomo's Kindle
        </user_device_name>
    <cookies>
  (cookies omitted)
       </cookies>
    </response>
```

Listing 6.7 registerDevice reply message

In Listing 6.6 registerDevice request message contained mainly the same parameters as getNewDeviceCredentials message (Listing 6.4). Parameters in registerDevice message were listed in XML at the message body instead of in the URL, as was the case with getNewDeviceCredentials message. The registerDevice request message contained more parameters such as user’s *email* and *password* and values for both of these parameters were presented in plaintext.

The reply message (Listing 6.7) included tokens for every entry presented under the request message's deviceTypeSoftwareVersionMap tag. The deviceTypeSoftwareVersionMap tag in registerDevice request message contains the same entries as in getNewDeviceCredentials message (Listing 6.4). The highlighted adp_token entry in the registerDevice reply message was observed to be used frequently. The same adp_token entry is mentioned twice in the Listing 6.7. First time it is mentioned under the deviceCredentials tag and was pointed to the deviceType A2VZ790DVVI91K in the Listing 6.7 (omitted from the listing). This deviceType was mapped in Listing 6.1 to a softwareComponenId named com.amazon.thor.android.os, which was thought to represent the user's operating system in the kindle device. Second time the token is mentioned immediately following the deviceCredentials XML closing tag (highlighted in Listing 6.7).

6.1.4 Token acquisition - Exchangetoken

Amazon uses exchangetoken messages to acquire, refresh and access tokens. Refresh token's role and usage is similar to Google's master token as both are used to acquire new tokens. Applications use access tokens to access Amazon's services and resources.

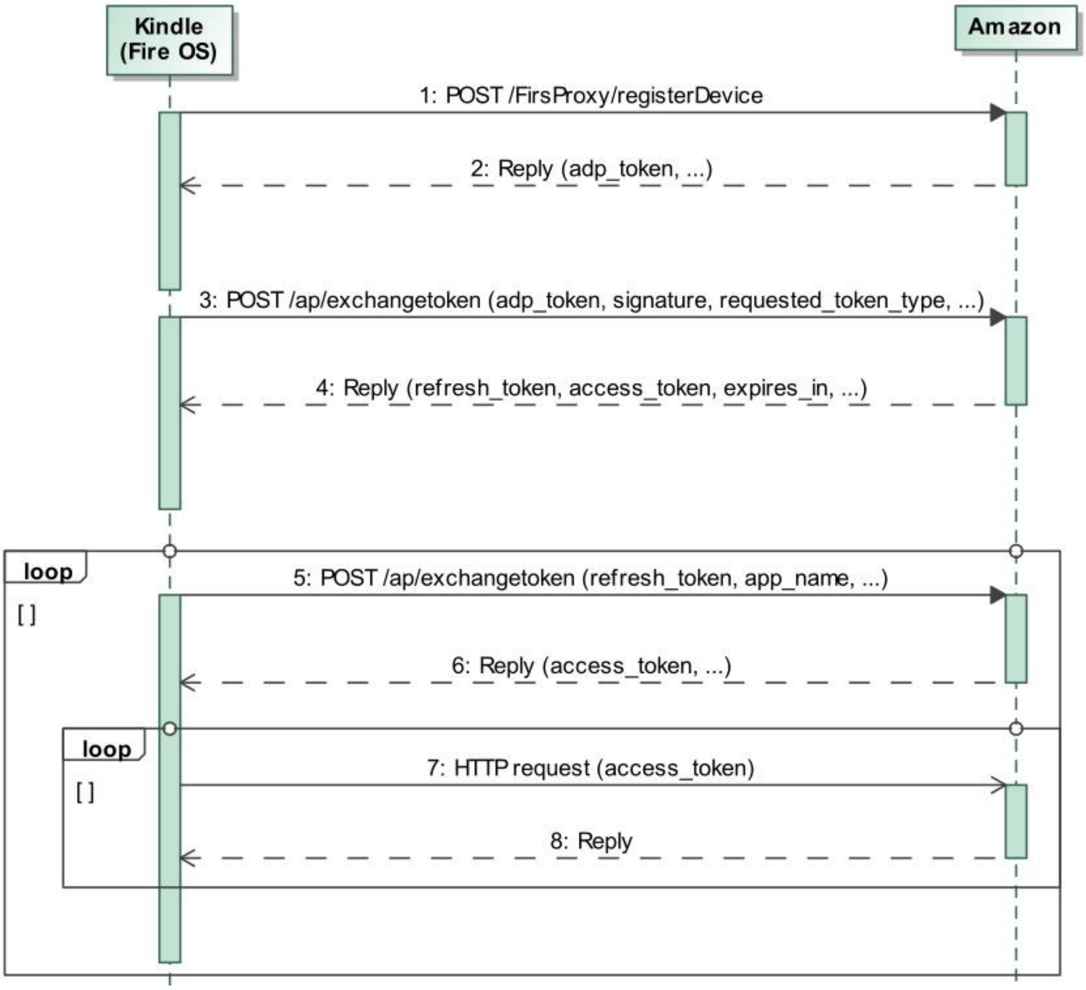

Kindle (Fire OS)
Amazon
1: POST /FirsProxy/registerDevice
2: Reply (adp_token, ...)
3: POST /ap/exchangetoken (adp_token, signature, requested_token_type, ...)
4: Reply (refresh_token, access_token, expires_in, ...)
loop
[ ]
5: POST /ap/exchangetoken (refresh_token, app_name, ...)
6: Reply (access_token, ...)
loop
[ ]
7: HTTP request (access_token)
8: Reply

Figure 6.2. *Amazon token acquirement and reacquisition process.*

Figure 6.2 presents both the token acquirement and the renewal process. First the mobile device gets an adp_token meant for com.amazon.thor.android.os application during the registration. This is represented in the registerDevice request and reply messages (Messages 1 and 2, Figure 6.2; Messages 5 and 6 Figure 6.1; Listings 6.6 and 6.7) In the second message pair (Messages 3 and 4, Figure 6.2) the mobile device acquires a refresh token and a temporary access token. The refresh token and the access token are acquired by sending a HTTP POST message (along the adp_token) to address www.amazon.com/ap/exchangetoken. Exchangetoken message is normally used to acquire new temporary access token or cookie tokens for applications, but this case (Message 3, Figure 6.2; Message 7, Figure 6.1) is an exception, because both refresh token and temporary access token are acquired at the same time. After this the device can use the temporary access token as long as it is valid (Messages 7 and 8, Figure 6.2). In the first time (during the registration) Messages 5 and 6 are skipped over.

After the access token has expired a new token is acquired (Messages 5 and 6, Figure 6.2). The exchangetoken message is used with the refresh token to acquire new access token or cookie tokens for an application to use (Messages 7 and 8, Figure 6.2). Refresh token was not observed to expire during the observations.

```
POST /ap/exchangetoken HTTP/1.1
User-Agent: AmazonWebView/MAPClientLib/120069810/Android/4.2.2/KFTHWI
Host: www.amazon.com

app_name=com.amazon.imp
&app_version=120039810
&source_token_type=refresh_token
&source_token=Atnr|EQEBLjAsAhQ5XRXMV...
&requested_token_type=access_token
```

Listing 6.8 Exchangetoken (access token) request message.

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Set-Cookie:  session-id=177-3291618-2853634;  Domain=.amazon.com;  Expires=Wed,
19-Apr-2034 12:52:36 GMT; Path=/
Set-Cookie: session-id-time=2029063956l; Domain=.amazon.com; Expires=Wed, 19-
Apr-2034 12:52:36 GMT; Path=/

{
   "response": {
      "token":"Atna|EQEBLjAsAhRAto0C ... ",
      "token_type":"bearer_token",
      "token_expires_in":"3600"
   },
   "request_id":"06K8ERKMAVP4GVSZNSEK"
}
```

Listing 6.9 Exchangetoken (access token) reply message.

Access token reacquisition request message contained five parameters in the message body: app_name, app_version, source_token_type, source_token and requested_token_type. Reply message contained a JSON object (Listing 6.9) in the message body, which had the requested token value and expiry time.

```
POST /ap/exchangetoken HTTP/1.1
x-adp-alg: SHA256WithRSA:1.0
x-adp-signature: VmleK1FFS2ymNj/8uD0VKp ...
x-adp-token: {enc:O5R4tVC8+FMNe61lpl3e ...

source_token_type=dms_token&source_token=source_token&requested_token_type=ref
resh_token&app_name=com.amazon.imp&app_version=120039810
```

Listing 6.10 Exchangetoken (refresh token) request message.

The exchangetoken message during the initial token acquirement was an exception. The message was used to request a refresh token and access token during the registration. The message had signature and adp_token in the HTTP header and no source token was used in the message body (Listing 6.10). The reply message was otherwise same as in Listing 6.9, but it contained one more JSON object, refresh token, in the message body.

6.1.5 Cookie token acquisition - Exchangetoken/cookies

Exchangetoken/cookies is a message used to acquire cookie tokens for applications. Exchangetoken/cookies request and reply message are very similar to the exchangetoken message in structure and also have the one exception case, which happens during the registration. Also the token acquisition and renewal happens in the same way as with exchangetoken messages.

```
POST /ap/exchangetoken/cookies HTTP/1.1
X-Amzn-RequestId: 70993150-bbec-493b-8bf0-b232ab9ed2f5
User-Agent: AmazonWebView/MAPClientLib/120069810/Android/4.2.2/KFTHWI
Cookie: session-id=190-2410784-9443562
Cookie: ubid-main=177-4988256-9367335
Cookie: x-main="oYD@eSq5r6YrZ9ikjnrWdVwF98b..."
Cookie:at-main="5|LDjqyGDFz90dCSsdnDjKbLbRj..."
Cookie: sess-at-main="WutkQPllmrWqw8unKcFsfhBFLOBbcTbiRGC6/knWAUA="
Host: www.amazon.com

source_token_type=refresh_token
&source_token=Atnr|EQEBLjAsAhQ5XRXMVIHcCu3Fd9...
&requested_token_type=auth_cookies
&domain=www.amazon.com
&app_name=com.amazon.imp
&app_version=120039810
```

Listing 6.11 Exchangetoken (auth cookie) request message.

```
HTTP/1.1 200 OK
Date: Mon, 04 Aug 2014 11:55:19 GMT
Content-Type: application/json;charset=UTF-8
Set-Cookie:  session-id=184-2273602-6255638;  Domain=.amazon.com;  Expires=Sun,
30-Jul-2034 11:55:19 GMT; Path=/
Set-Cookie: session-id-time=2037873319l; Domain=.amazon.com; Expires=Sun, 30-
Jul-2034 11:55:19 GMT; Path=/
Set-Cookie:  ubid-main=182-7107327-1967538;  Domain=.amazon.com;  Expires=Sun,
30-Jul-2034 11:55:19 GMT; Path=/
Set-Cookie: session-token="U0HxLsnqkMh0q7wWuJB3k+YtqdYzEeqJtefLdQKOpt9AfFBFxpW
IQV/omwmyFiDYITEMK/aZ4miLTZXWn6qTdkeLBrjp4jcGWbvIoyWR0swDlHXXJBUDXH9XlKjlm2B/G
2CFm6KCyH5YWV2ATDZL5MOMHB7CmbViOqDb/FN6SwQT4NJT5Gq+9jxH/Ug2R++fBJbEFPOC6tEVKTC
MKJC/IU9i06PRu3k1sJdN/BbZ8vE=";     Version=1;     Domain=.amazon.com;     Max-
Age=630720000; Expires=Sun, 30-Jul-2034 11:55:19 GMT; Path=/
Content-Encoding: gzip

{"response":{
      "tokens":{
         "cookies":{
            ".amazon.com":[
               {"Name":"session-id",
                  "HttpOnly":false,
                  "Value":"184-2273602-6255638",
                  "Expires":"30 Jul 2034 11:55:19 GMT",
                  "Secure":false,
                  "Path":"/"
               {"Name":"ubid-main",
                  "HttpOnly":false,
                  "Value":"182-7107327-1967538",
                  "Expires":"30 Jul 2034 11:55:19 GMT",
                  "Secure":false,
                  "Path":"/"),
               {"Name":"x-main",
                  "HttpOnly":false,
                  "Value":"\"KTb@txAuqA1gJwuoLlCF?Zje3IDraRzwi@qxXPROVYl?5BRHJ
mG3Q6mcLqkbhiSb\"",
                  "Expires":"30 Jul 2034 11:55:19 GMT",
                  "Secure":false,
                  "Path":"/"),
               {"Name":"at-main",
                  "HttpOnly":false,
                  "Value":"\"5|zMxne86heAPMFpj8vUSatJEVekgpwchVLZbzWNLuK6Zekn9
5OrAqH+okB2d/U7sJokY+l+OY2LO9YbZfuVONS3MFSlZsF/QzevwGoiObA39czy/RxeYD0CNujj2wf
RVehIwvITRLGEUd8kc8I6d6lfF4cD1SRt8uDvosP93lZcLp2OO5uVex3jXnScxBTRgteo3XMo//x+C
Bia69lU/rb3a+uGY9xym8\"",
                  "Expires":"4 Aug 2014 12:55:19 GMT",
                  "Secure":true,
                  "Path":"/"),
               {"Name":"sess-at-main",
                  "HttpOnly":false,
                  "Value":"\"IF2JC9dGVffus4ZzZKXaCAjDPeXCv/xBb0VrD5jy+lY=\"",
                  "Expires":"4 Aug 2014 12:55:19 GMT",
                  "Secure":true,
                  "Path":"/}
            ]}}},
   "request_id":"030C9TSQRHGTGHW78KC6}
```

Listing 6.12 Exchangetoken (auth cookie) reply message.

The exchangetoken/cookies message was observed to acquire five to six cookie tokens: session-id, ubid-main, x-main, at-main, sess-at-main and session-token. Session-token cookie was not always assigned and the logic behind this was not understood. Session-token, when assigned, was always given in HTTP header and not in the JSON object at HTTP message body as the rest of the cookies. Session-token cookie was also given in at least one other message, which was not related to token acquisition. Cookies at-main and sess-at-main are given one hour expiry time and the rest of the cookies were given 20 years minus 5 days.

```
POST /ap/exchangetoken/cookies HTTP/1.1
X-Amzn-RequestId: ad763a6c-0c1c-4b37-852c-0bd75957cdfe
Content-Type: application/x-www-form-urlencoded
User-Agent: AmazonWebView/MAPClientLib/120069810/Android/4.2.2/KFTHWI
Host: www.amazon.co.uk
Content-Length: 103

requested_token_type=auth_cookies
&domain=www.amazon.co.uk
&app_name=com.amazon.imp
&app_version=120039810
```

Listing 6.13 Exchangetoken/cookies (initial token) request message.

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Set-Cookie: session-id=276-9389280-3838434; Domain=.amazon.co.uk; Expires=Mon,
05-Jun-2034 11:47:39 GMT; Path=/
Set-Cookie:  session-id-time=2033120859l;  Domain=.amazon.co.uk;  Expires=Mon,
05-Jun-2034 11:47:39 GMT; Path=/

{"response":{
      "tokens":{
         "cookies":{
            ".amazon.co.uk":[
               {"Name":"session-id",
                  "HttpOnly":false,
                  "Value":"276-9389280-3838434",
                  "Expires":"5 Jun 2034 11:47:39 GMT",
                  "Secure":false,
                  "Path":"/"),
               {"Name":"ubid-acbuk",
                  "HttpOnly":false,
                  "Value":"276-3404217-4919469",
                  "Expires":"5 Jun 2034 11:47:39 GMT",
                  "Secure":false,
                  "Path":"/"}
            ]}}},
   "request_id":"0HHJDB1386HCE0VESJPA"}
```

Listing 6.14 Exchangetoken/cookies (initial token) reply message.

Listing 6.13 and Listing 6.14 present exchangetoken/cookies request and reply messages, which were used only when the user's account was registered to the device. Messages are slightly different from the messages presented in Listings 6.11 and 6.12. The request message (Listing 6.13) is otherwise the same, but it is missing source_token and source_token_type parameters from the message body. Missing parameters are explained by the fact that when this request message is sent, no refresh token has yet been acquired.

The reply message (Listing 6.14) differentiates from the other message (Listing 6.12) by having only 3 cookies: session-id, session-id-time and ubid-acbuk. The ubid-acbuk cookie name is thought to depend on the domain mentioned in the request message, since Amazon has shops and domains customized for different countries, for example, UK and Germany.

6.1.6 Token request parameters

Tables 6.1 and 6.2 present parameters used in the token related request messages. Table 6.1 concentrates on messages concerning token acquisition, including exception cases in initial token acquisition while adding a user to a device. Table 6.2 presents other messages related to token acquirement when a user is added to a device.

Table 6.1: *Token exchange request parameters*.

Parameter	Exchangetoken/ cookies (initial)	Exchangetoken/ cookies	Exchangetoken
requested_token_type	x	x	x
domain	x	x	-
app_name	x	x	x
app_version	x	x	x
source_token_type	-	x	x
source_token	-	x	x

Requested_token_type parameter defined what kind of token was requested. Observed values were: auth_cookies, refresh_token and access_token. Domain parameter was thought to define which domain address the request is supposed to go. For example, www.amazon.com and www.amazon.co.uk were observed. Source_token_type parameter defines the type of the source token (e.g. refresh_token).

Table 6.2: *Device register and credentials request message parameters.*

Parameter	**getNewDevice Credentials (url parameters)**	**getNewDevice Credentials (xml in HTTP message body)**	**registerDevice (xml in HTTP message body)**
deviceType	x	-	x
deviceSerialNumber	x	-	x
pid	-	-	x
email	-	-	x
password	-	-	x
secret	x	-	x
softwareVersion	x	-	x
softwareComponentId	x	-	x
deviceTypeSoftwareVersionMap (entry: deviceType, version, softwareComponentId)		x	x
reason	x	-	-
radioId	x	-	-

DeviceType parameter contained a string from capital alphabets and numbers, which was suspected to be the unique identification of a specific application (e.g. com.amazon.thor.android.os) and its version number. SoftwareComponentId contained the name of the application file (e.g. com.audible,application.kindle). DeviceTypeSoftwareVersionMap parameter contained entries, which hold deviceType, softwareComponenId and softwareVersion.

RadioId parameter contained the device's MAC-address. Reason parameter was always observed to contain value "NoState" and was suspected to tell the reason why the request is made.

6.2 Practical security in Amazon Shop

Amazon Shop's free and paid software cases were made simple. A user starts Amazon Shop application, installs free or paid software and then leaves the shop. The message flow presented in Figure 6.3 is identical with free and paid software cases. The Figure 6.3 presents only the most relevant messages from the message flow. Omitted messages were related to information and metadata requests, one redundant token request, image downloads for the shop and one partial application download.

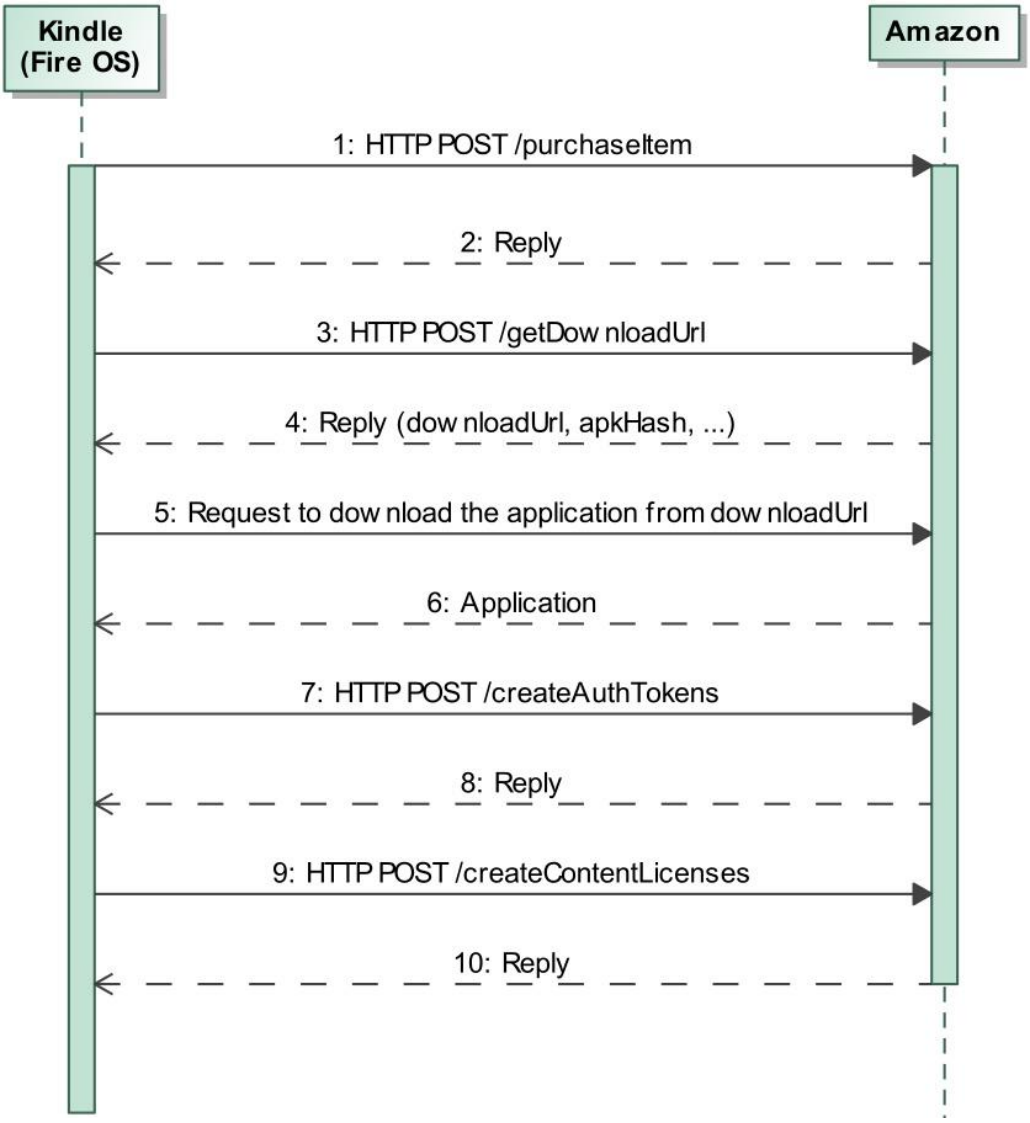

Kindle (Fire OS)
Amazon
1: HTTP POST /purchaseItem
2: Reply
3: HTTP POST /getDownloadUrl
4: Reply (downloadUrl, apkHash, ...)
5: Request to download the application from downloadUrl
6: Application
7: HTTP POST /createAuthTokens
8: Reply
9: HTTP POST /createContentLicenses
10: Reply

Figure 6.3 *Message flow when buying and downloading software from Amazon shop.*

```
POST /purchaseItem HTTP/1.1
x-venezia-pfm: ATVPDKIKX0DER
User-Agent: VeneziaAndroid/release-7.1017
x-venezia-cor: FI
Session-ID: 191-0368151-6201814
X-ADP-Request-Digest: Qi4216+DgNegys2Fb7ft...
X-ADP-Authentication-Token: {enc:O5R4tVC8+...
Content-Type: text/plain; charset=UTF-8
Host: mas-ext.amazon.com
```

```
{"zeroesPaymentActive":false,
   "currentVersion":"0",
   "searchAnalytics":{
      "refMarker":"apps_th_gd_gm_4"},
   "currentPrice":{
      "amount":"0.00",
      "unit":"USD"},
   "deviceInfo":{
      "ref":"unknown",
      "model":"KFTHWI",
      "deviceDescriptorId":"MDD-S-3F6CWH2R03YPF",
      "osVersion":"17",
      "deviceType":"A2VZ790DVVI91K",
      "manufacturer":"Amazon",
      "carrier":"unknown",
      "build_fingerprint":"Amazon\/thor\/thor:4.2.2\/JDQ39\/13.3.2.3.2_user_
323001720:user\/release-keys",
      "build_product":"thor"},
   "asin":"B00JX66AV0"}
```

Listing 6.15 PurchaseItem request message.

```
HTTP/1.1 200 OK
Content-Type: application/json

{"displayMessageKey":"mas.device.purchase.success.no_error",
   "orderId":"D01-8181186-4443362",
   "purchaseErrors":"NoError",
   "stateToken":"---===**{{{[VeNeZiA]}}}**===----"}
```

Listing 6.16 PurchaseItem reply message.

PurchaseItem message (Message 1, Figure 6.3) contained two HTTP header fields: X-ADP-Request-Digest and X-ADP-Authentication-Token. These two HTTP header fields were used in nearly every message sent from Kindle Fire device to Amazon when downloading a free application. Exceptions were Messages 3 – 4 from Figure 6.3 and certain messages (omitted from Figure 6.3) related to content metadata.

The same authentication token was used in every message. The used token was observed to be obtained during the registration (Message 6, Figure 6.1) and the acquired token was pointed to the deviceType A2VZ790DVVI91K. As mentioned earlier this deviceType was mapped to a softwareComponenId named com.amazon.thor.android.os, which was thought to represent the user’s operating system in the kindle device. This same deviceType is also mentioned in the message body of purchaseItem request message (Listing 6.15).

The reply message for puchaseItem contained four fields: displayMessageKey, orderId, purchaseErrors and stateToken. Listing 6.16 presents the case when purchase was done successfully. In the case of failure orderId and stateToken fields were left empty and other fields had information regarding the error. During observations one purchase error was captured, it was due the fact that the author had not selected default payment type from Amazon account settings. During this error the author was directed to sign-in with the browser to Amazon and change the setting for payment. Afterwards purchases were made successfully.

One notable observation was that the user was never asked to authenticate when buying an application or downloading a free application. Also the user was required to add credit card details to user’s Amazon account before the user was allowed to install any software, free or paid.

```
HTTP/1.1 200 OK
Date: Mon, 23 Jun 2014 08:50:56 GMT
Content-Type: application/json
Content-Length: 351

{  "apkHash":"R6YV3ht6kFugGE347TmXZg==",
   "downloadUrl":"https://amznadsi-a.akamaihd.net/US/prod/B008K38DXK-10-
844a3a45-ce68-492e-a70e-34f18e342882.bin?AWSAccessKeyId=AKIAJNJFQESGNKQ7THPA
&Expires=1404118256&Signature=aqqyoBmv7/L94arzViPTg79aAso=&__h__=1404118256_4a
dc6f2706a2de23f2b9c9f81d8f219a",
   "latestContentId":"MC-S-39P8Q6KEBTN92",
   "packageName":"com.rovio.angrybirdsseasonsHD",
   "stateToken":null}
```

Listing 6.17 getDownloadUrl reply message.

The reply message for Message 3 (Figure 6.3) contained in the message body a hash value for the requested application file and URL from where to download the application. The download URL (for the requested application) contained the following parameters: filename, AWSAccessKeyId, expiry time, signature and one unknown parameter __h__.

AWSAccessKeyId's role in this message is not fully understood. Amazon's AWS (Amazon Web Services) documentations might give some hints. AWSAccessKeyId is an identification value distributed by AWS (Amazon Web Services), when a user signs up for an AWS account [47]. During the same time when AWS access key is obtained, the user also gets a secret key. These two values are used to sign requests made by applications to AWS. [48] Based on the AWS documentation, AWSAccessKeyId could identify the user to whom the device has been registered to. AWSAccessKeyId was observed to change only when the user registers herself to the device, but observations do not rule out the possibility of AWSAccessKeyId to change after a time, since the observations were done in two sets separated by only three months.

The signature in the download URL is thought to be the signature of the download request and not of the requested application, because the signature resembles the AWS signature version 2 [47]. The AWS documentation specifies that the signature in the query URL must be base64 encoded and then URI encoded (the URL in the Listing 6.17 is decoded). The documentation also specifies that the signature has to be calculated with either HMAC-SHA1 or HMAC-SHA256 protocols. [47] The signature given in the Listing 6.17 meets the first criteria and the signature is 160 bit long after URI decoding, which is also the length of the HMAC-SHA1 output [13]. The expiry time given in the URL was unix epoch time format and the time was set to expire exactly in one week (168 hours). The same expiry time was also a part of the unknown parameter __h__ value.

```
POST /createAuthTokens HTTP/1.1
User-Agent: VeneziaAndroid/release-7.1017
Content-Length: 86
Content-Type: text/plain; charset=UTF-8
Host: mas-ext.amazon.com
```

```
{  "stateToken":"",
   "clientVersion":"release-7.1017",
   "contentIds":["MC-S-39P8Q6KEBTN92"]}
```

Listing 6.18 createAuthTokens request message.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 370

{ "authTokens":{
      "MC-S-39P8Q6KEBTN92":"eiTaNCoAaLUOGGHrzssKCJpArWKclVLcAJM4j3RTpWigua5cg4
fBSCd57OSaK5dsDHKlLkg37uc8XL+E4l1lHH4tlWNY0K/mmPQhm0hpWMD958VpnuauKYIshKo7SFG2
Xh3bYIwE9zmt4Aqa6Uuxj9vfAAn1aQLpXGjvIyReJVHsJJbjOVIDTbjYgjfvjhPB0baskvX5n7iMzF
Q5T55yZkqUVIHEdKu11ax/2X883INGxIqCSV9jZZVfJRvCkId+faimdhw542OC2SHfzXrhUHNZbKpn
ApiHliXqvtKBW1eXcezec0Tju9/aYd9j9Y1DREesIOV8IJ0pPjz3Et2ZYQ=="
   },"errors":{},
   "stateToken":null}
```

Listing 6.19 createAuthTokens reply message.

The createAuthTokens request message is one the few exceptions that did not contain a digest and token pair. The request message contained contentId parameter in the message body to refer to the desired application. The contendId was given in the reply message (Listing 6.17) for the getDownloadUrl message. The reply message in Listing 6.19 provided the requested token in the message body. The token was base64 encoded and decoding it did not produce anything comprehensible and its usage was left unknown.

```
POST /createContentLicenses HTTP/1.1
x-venezia-pfm: ATVPDKIKX0DER
User-Agent: VeneziaAndroid/release-7.1017
x-venezia-cor: FI
X-ADP-Request-Digest: VABIXC16jBw3kBYccXYcR ...
X-ADP-Authentication-Token: {enc:O5R4tVC8+FMNe61lpl3e2 ...
Content-Length: 358
Content-Type: text/plain; charset=UTF-8
Host: mas-ext.amazon.com

{  "stateToken":"",
   "contentIds":["MC-S-39P8Q6KEBTN92"],
   "deviceInfo":{
      "ref":"unknown",
      "model":"KFTHWI",
      "deviceDescriptorId":"MDD-S-3F6CWH2R03YPF",
      "osVersion":"17",
      "deviceType":"A2VZ790DVVI91K",
      "manufacturer":"Amazon",
      "carrier":"unknown",
      "build_fingerprint":"Amazon\/thor\/thor:4.2.2\/JDQ39\/13.3.2.3.2_user_32
3001720:user\/release-keys",
      "build_product":"thor"
}}
```

Listing 6.20 createContentLicenses request message.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 663

{"errors":{},
   "licenses":{
      "MC-S-39P8Q6KEBTN92":{
         "contentId":"MC-S-39P8Q6KEBTN92",
         "expirationDate":1.408698591739E9,
         "token":"ZXhwaXJhdGlvbj0xNDA4Njk4NTkxNzM5LGRybU1vZGU9MSxkZXZpY2VJZD1E
MEZCQTBBMDM0NTMwV1dDLHBhY2thZ2VOYW1lPWNvbS5yb3Zpby5hbmdyeWJpcmRzc2Vhc29uc0hELG
N1c3RvbWVySWQ9QTFUVDcwMFExUjFCMzcsY2hlY2tzdW09UjZZVjNodDZrRnVnR0UzNDdUbVhaZz09
LHRzPToxNDAzNTE0NTkxNzQ0fGU0UTB4aHp4dlpzWWZjK2EzM0U1b0lHeW9pOGt0NktjQXhDbjR3RD
QydnlLRHJpRE5BSkEwQzdyOS9pMk5OR0FTQ0FpUlZ0aW80YjNYTTNHUCt3akVNL0xiVG1vWEtZNDRW
V1RjMk01bWdXdjM1WHNtcVk3WFBRY0I4SWVENU5PTm5IMXFuOWsrTlhPQVB1SHF5eFVBQWEwVnFHWX
JJdUY0WERET2lFWXYydnY4ZFNFZVBNS0Y4c3R1cG8vdldhMG9XT1RXdlFKaGdIQmZVTkhSSlUyaEkx
NEpmeUkxQ3RwY3JHUER4RzTQWdVYUFxY0xqRlorRlZrY0NoS20rbXRJQVlZbnp4RWdYb0MxblovSF
hGQ05KWjVwdVVlZXkyOHJSL1BOSzdkcnFNTVRkQ2ZoQ1NoWml4anlFVGF3aGtnN0xwc0k1OGpETmZJ
b1lFbENkT2hUdz09"
      }
   },"stateToken":null}
```

Listing 6.21 createContentLicenses reply message.

The createContentLicenses request body contained *contentId* referring to the desired application and information regarding the user's device. The reply message in Listing 6.21 contained among other things *expirationDate* and a *token* that were base64 encoded. The token was decoded as follows:

```
expiration=1408698591739,drmMode=1,deviceId=D0FBA0A034530WWC,
packageName=com.rovio.angrybirdsseasonsHD,customerId=A1TT700Q1R1B37,checksum=R
6YV3ht6kFugGE347TmXZg==,ts=:1403514591744|e4Q0xhzxvZsYfc+a33E5oIGyoi8kt6KcAxCn
4wD42vyKDriDNAJA0C7r9/i2NNGASCAiRVtio4b3XM3GP+wjEM/LbTmoXKY44VWTc2M5mgWv35Xsmq
Y7XPQcB8IeD5NONnH1qn9k+NXOAPuHqyxUAAa0VqGYrIuF4XDDOiEYv2vv8dSEePMKF8stupo/vWa0
oWOTWvQJhgHBfUNHRJU2hI14JfyI1CtpcrGPDxRzSAgUaAqcLjFZ+FVkcChKm+mtIAYYnzxEgXoC1n
Z/HXFCNJZ5puUeey28rR/PNK7drqMMTdCfhCShZixjyETawhkg7LpsI58jDNfIoYElCdOhTw==
```

Listing 6.22 Base64 decoded createContentLicenses message reply token.

The token contained among other things expiration time, device Id, package name, timestamp (parameter ts) and base 64 encoded checksum. The token also contained 344 character long base64 string, which was not understood after a decode attempt. The string contained after decoding 256 bytes (2048 bits) of data. From the length the author suspects it to be either some sort of key for asymmetric encryption algorithm or digital signature. The expiration time here is the same as expirationDate parameter presented in Listing 6.21, it is presented in another format.

6.3 Discussion

The x-adp-tokens and signatures/digest values addressed in Sections 6.1.1 and 6.1.2 left a few open questions. What is encrypted in the x-adp-token and how it is encrypted? Other questions are: what cryptographic hash function is used and where the key for signatures is gotten from? Subsections 6.3.1 and 6.3.2 try to answer to these questions. Chapter concludes in discussing what problem was encountered with the fake CA SSL certificate.

6.3.1 Speculation of encrypted tokens

Regarding x-adp-tokens, the enc part is thought to contain encrypted information and key and iv parts are thought to contain the encryption key and initialization vector for the used encryption algorithm. Encryption key and initialization vector together suggested that the used encryption algorithm is a symmetric algorithm, but on the other hand the length of the possible encryption key is unusually long (2048 bits). For example, AES-256 algorithm has 8 times shorter key. The length of the key actually hints that it is encrypted with asymmetric encryption. The idea is presented in Listing 6.23.

{enc: $\{content\}_{K_S}$}
{key: $\{K_S\}_{K_A^-}$}
{iv: ... }

Listing 6.23 Example speculation.

In the Listing 6.23 the key K_A^- is thought to be Amazon’s private key of asymmetric encryption algorithm used to wrap (encrypt) symmetric encryption key K_S. The symmetric key thought to be used together with iv to encrypt the content in the enc field. The encrypted contents were left unknown.

6.3.2 Digest function and origin of signing key

X-adp-request-digest‘s cryptographic hash function is one mystery, because the length of the hash value seemed to be unusually long (1536 bit). This limits the possible list of hash algorithms, but the length of the x-adp-request-digest value also suggests another possibility. The digest could actually be a digital signature and the name x-adp-request-digest is just a legacy name just as discussed in the Google’s case in Section 5.3.2.

```
MIIDlQIBADANBgkqhkiG9w0BAQEFAASCA38wggN7AgEAAoHBAMq4bDtSriLvJNCbewFjK94Hv5Nyo1
8+hf1o0Tj5CSTNygNd7Rla9eKCcrJS06qIdJ0IE0lHeOWd/dYAxd4U4CAzZqvOX5mKrrc1bGRuiizM
GyaBKrHwcbFFIU17ptBzui2AFYP9QrTiMjzj+vjXe32G6I5wXKy8XhTl+PeghGk/I8KODaoccCnNQj
Ln6xSB2ZgGeCUk6veCGRIHrKjI03a0D5yZixFELQMQUberyNTmQDuZcRmfuFnljgoyBbA7twIDAQAB
AoHAPn4Jx2PekKBeHfzAN5ZF0KVc1mxxlovkrFDipoiG3BSgYNdUnwteX1xNVVGZNKnqIPp7T4y75T
```

```
UGKjGEzchrqyGNVgtBh0isQYgziOZWrCajfAQ3kDEKK1afRI6IxxvVTO0IqZ/XdjZUjoKL/XSN7vsF
5sLtH//D2frv7jImVmtCiEWA8WX33eibcND2e4/Mp01uj7ow13RdLdILV5hCd9RnZWHHqhjc9RHIkc
3mNbiWBm1k4z/RzRxpi+9+0HnZAmEA+P0MRA03DULOekzCyk2fPQs4UzLwkzKsqRh+o+wdjRUvj67v
ItmIg2C9QJYWWu67VwTx2J7Xj8JhSjq8/Okz3ngd3TzLN9nm4O5DraxMDv421mRjhIJH+ZPITOkLbS
ZVAmEA0G3UVqAweiXMSdTCDyHFzdh+wJvDHXemoNkFyVuEl7FcZsXpxBXKmJ5m8rHWl/dyjixxvnFQ
F9CwhHZDCn7tYhdIfGEOnb0oJBmyQLUEecq+YW0EwUqJKZBMILEJxa3bAmAUCnIhTBsBQz5RU7peBC
9r/2oyMChzAKIrHiCbWxbp0ym32/G9kVrOkEvLVglImmyTzX8V23soRBIGooGxraeSIYLt2sXUogJW
WncyFGsuzcFmMkBcWwwmB4IYxq03xN0CYFqhNzCHrSyf3OnJsqxDwjAU0GDOGhRpH7JXS9XXjpsrSw
OBUXmrBBJC5n7nZ2li9pPtrXPi77G7U9X1Wodo063Qc/tpdKKQSrEP31uVmqGe0BOrBONLlsLU9NVO
RfKH+wJgM8B7XbPYTQ/xeU2vxMnBNGbbQif9PF4KBSAeFb7vmW2D3Jte7VNh3vIpJb7BcipdRmjQgB
remvXEkxEiqH3e9QkOUyyjtXW5oWN2eYply2ByXe4F6suEKfGI5/2UPp6z
```

Listing 6.24 Example of given base64 encoded device private key.

Another problem was the origin of the private key that Amazon’s client applications in user’s device use to sign messages. During the registration a *device private key* is given, which might have something to do with signatures (Listing 6.7, subsection 6.1.3), but the author personally doubts this. One given device private key is presented in Listing 6.24. The key is 1228 character long and base64 encoded, which decodes into 7368 bit of data. The length of the given key (7368 bit) is too long for the key used in observed SHA256WithRSA signatures (1536 bit), because according to the standard the length of the signature is the same as the length of the private key [46]. No other potential key candidate was noticed in the desired length and therefore the origin of the used signing key is left unknown.

6.3.3 Problems with a fake CA SSL certificate

During the observations a second fake CA SSL certificate was created, but when it was deployed all SSL/TLS communications failed during the handshake. At the same time, when the new certificate was deployed the Amazon’s device got a new update. It was first thought that the new behaviour was a result of the update and Amazon had implemented new mechanism to prevent MITM attacks.

Later on an older certificate was tried and noticed that SSL/TLS communications worked again. It was then understood that the problem lied in the new certificate and in closer inspection it was noticed that the new certificate had only one field (common name) filled during its creation with OpenSSL. This was due the author’s own fault. The actual reason why the new certificate did not work with Amazon’s client applications was left unknown, but it was understood that all the fields needed to be filled.

7 WHEN SSL FAILS

The MITM attack used in this thesis for observing and capturing encrypted SSL/TLS communications is not an easy attack to do in a real life due to its requirements. However, the attack is more problematic than it first appears for Google Android users. A MITM attack was performed in a laboratory environment to demonstrate what an attacker could gain and achieve in a real attack against victim with a Google Android device.

The chapter is divided in three parts. First the MITM attack is described, especially, how it was done in practice. Then it moves to describe what information is needed for the exploitation, how to exploit and what information the attacker can gain. Chapter ends in discussing what the core problems that enabled attacks are, what has been done to prevent MITM attacks and finally the author presents a proposal how to prevent the described attacks. The findings have been informed to Google in 23th September 2014.

7.1 MITM attack in practice

A MITM attack is normally used to spy the victim’s communication, but the attack reveals only what the user has conducted during the communication. In Google’s Android case this is problematic for the user, because the information leaked in a first successful MITM attack could be used to spy the victim remotely after the attack and interact with the victim’s data stored in Google’s servers.

7.1.1 MITM attack preparation

Figures 7.1 and 7.2 presents the attack flow and Figure 7.4 describes what information the attacker is after during the attack and for what she can use that information. The attack can be divided in three phases: preparation, MITM attack and exploitation.

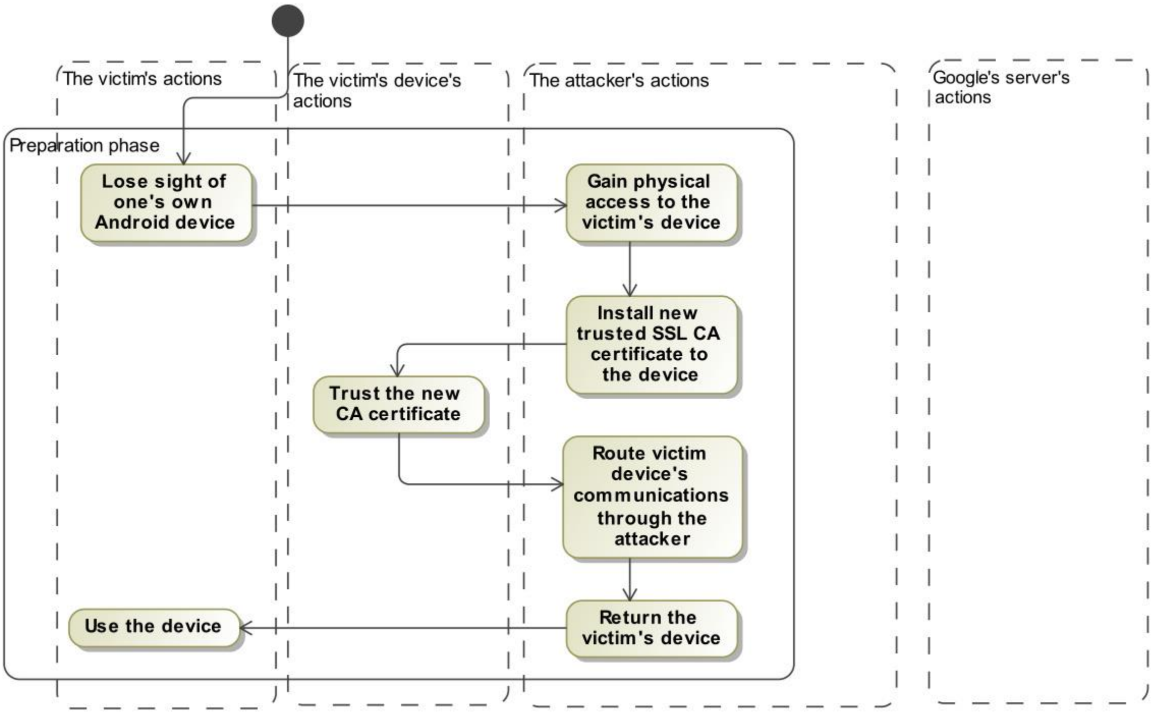

The victim's actions
The victim's device's actions
The attacker's actions
Google's server's actions
Preparation phase
Lose sight of one's own Android device
Gain physical access to the victim's device
Install new trusted SSL CA certificate to the device
Trust the new CA certificate
Route victim device's communications through the attacker
Return the victim's device
Use the device

Figure 7.1 *The MITM attack preparation phase.*

In the preparation phase the attacker first needs to gain physical access to the victim’s device in order to install a CA SSL certificate into the device’s trusted credential storage. And the attacker also needs to somehow route victim’s communications through the attacker. In practice the attacker can access to the victim’s device with a help of social engineering or simply steal the device. The attacker needs only circa 30 seconds to install a CA SSL certificate if the attacker has proper tools and scripts. As for tools the attacker needs a device with an USB port, an USB cable and a script which automatically copies a specific certificate to attached USB device’s root folder.

The victim’s communication can be routed through the attacker’s device in the same way as it was done in the laboratory, by making a WiFi-hotspot. The victim could be lured in by performing an “evil-twin” attack, which works creating a WiFi-hotspot with same SSID as a real hotspot and moving the rogue hotspot with stronger signal near to the victim [49]. Another way to route traffic could be for the attacker to insert new credential for her own WiFi-hotspot (which is near the victim) at same time when she is accessing the victim’s device and installing the CA certificate. Figure 5.8 presumes the attacker using the second option. Finally the attacker returns the device to the victim preferably without her knowledge. When the victim connects to the attacker’s WiFi-hotspot and starts to use the device then the MITM attack starts.

7.1.2 The attack phase

When the MITM attack starts the attacker has basically two choices: she can either just passively observe or actively modify (and observe) the communications. With passive observation the attacker can only gain authorization tokens present in requests made during the attack. And if the attacker is lucky she might intercept a token request, from which she gains the victim's master token (Listings 5.4 and 5.6).

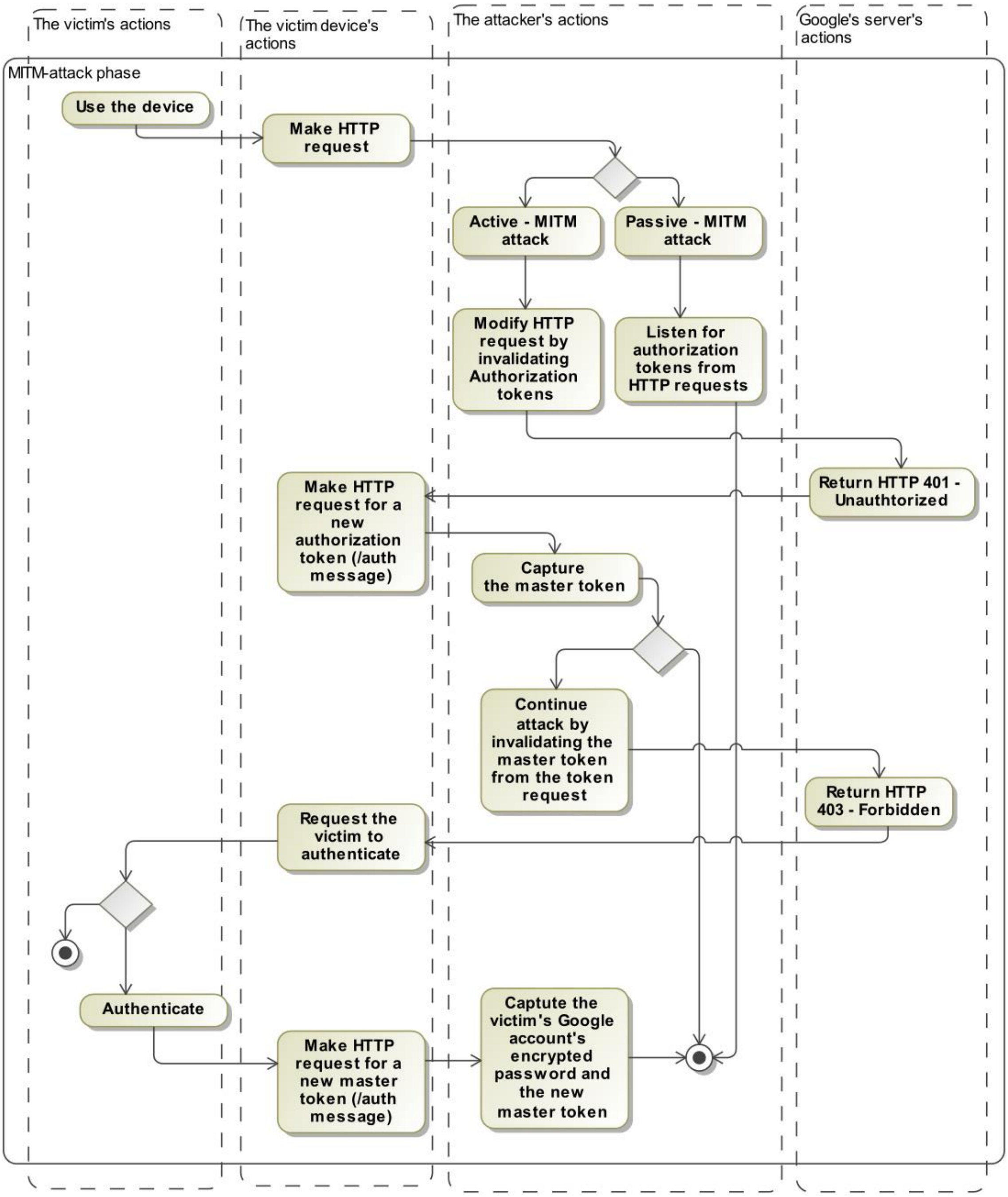
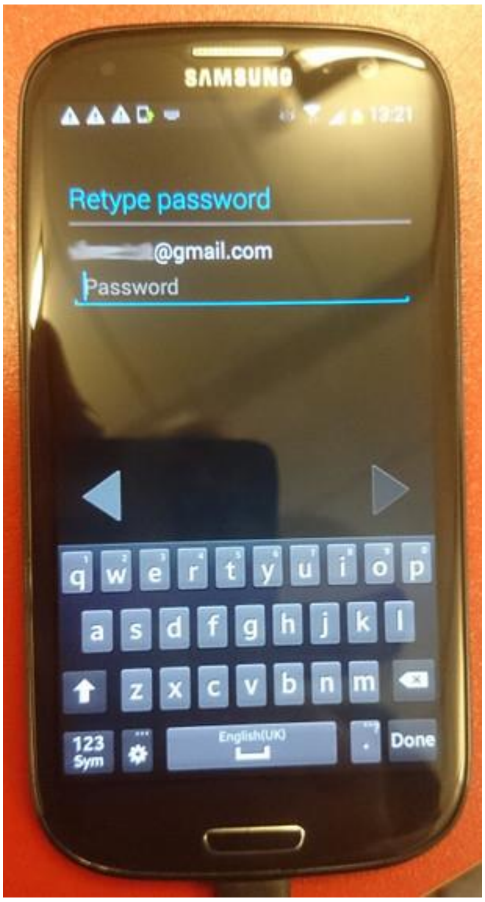

The victim's actions
The victim device's actions
The attacker's actions
Google's server's actions
MITM-attack phase
Use the device
Make HTTP request
Active - MITM attack
Passive - MITM attack
Modify HTTP request by invalidating Authorization tokens
Listen for authorization tokens from HTTP requests
Return HTTP 401 - Unauthtorized
Make HTTP request for a new authorization token (/auth message)
Capture the master token
Continue attack by invalidating the master token from the token request
Return HTTP 403 - Forbidden
Request the victim to authenticate
Authenticate
Make HTTP request for a new master token (/auth message)
Captute the victim's Google account's encrypted password and the new master token

Figure 7.2 *MITM attack phase.*

With an active MITM attack the attacker can "force" the victim's device to reveal information, which is useful in a situation where no token request has been seen. This is done by capturing HTTP requests victim's device makes and modifying the authorization tokens to be invalid, before they are passed on to the Google's servers. Because of the modification, Google's servers return *HTTP 401 – Unauthorized* reply to the device. The victim’s application gives up after a few failed attempts, presumably believing that the problem is in the authorization token, and tries to acquire a new token for itself. The token request (Listings 5.4 and 5.6) is interesting for the attacker because it contains the desired master token.

When the attacker gains the master token she is also faced with a decision: she can now either stop the active attack or continue the attack to gain even more information. If the attacker continues the attack, she can do it by modifying the master token from the captured token request to be invalid and pass the request on to the Google’s server. In this case the Google’s server return *HTTP 403 – Forbidden* reply to the device. At this point the device presumably believes the problem must be in the user’s account, since replies from Google and behaviour so far is the same what happens when the master token access has been revoked. Figure 5.9 presents the view the Android device used in the demonstration presented to the user after continued attack:


Retype password

[redacted]@gmail.com

Password

Figure 7.3 *View presented to the user when the attacker continues the attack.*

The device requests from the victim to authenticate. This is a critical point in the attack, because the user can opt not to authenticate right away (during the MITM attack), but the author personally believes the majority of the victims would authenticate at this point without a second thought. Once the victim authenticates the device sends a master token request (Listing 5.2), which is captured by the attacker. The interesting part from this message is the victim's encrypted Google account password and the new master token from Google's reply message for the request.

7.2 Gained information and its usage

After the MITM attack, starts the exploitation of the gained information, which is presented in Figure 5.10. The attacker has gained during the MITM attack: applications' authorization tokens, the victim Google account's master token and victim's encrypted Google account password. During the MITM attack the attacker has gained more information than just the previously mentioned, but she cannot trust that she has got what she wants from the victim during the MITM attack. Therefore the attacker is after tokens and encrypted password, which can be used after the attack to access remotely victim's data in the Google's servers.

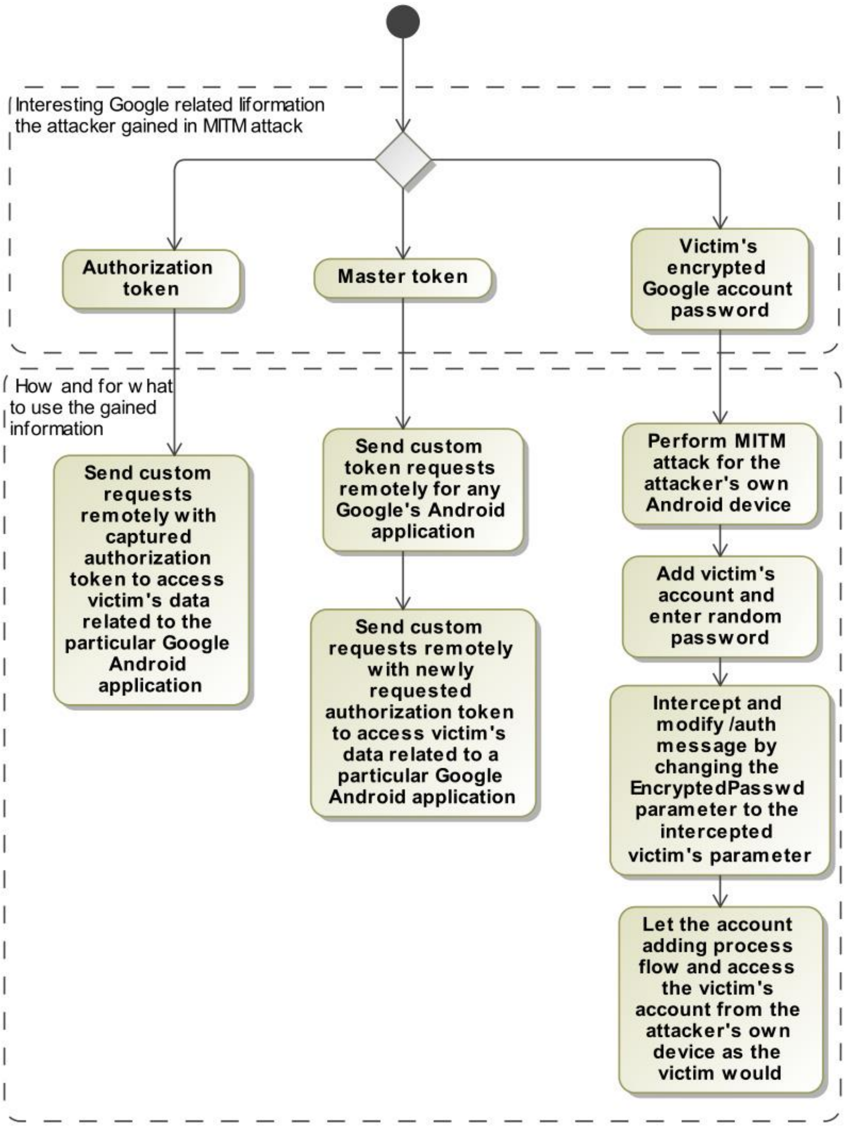

Interesting Google related lformation the attacker gained in MITM attack
Authorization token
Master token
Victim's encrypted Google account password
How and for w hat to use the gained information
Send custom requests remotely with captured authorization token to access victim's data related to the particular Google Android application
Send custom token requests remotely for any Google's Android application
Send custom requests remotely with newly requested authorization token to access victim's data related to a particular Google Android application
Perform MITM attack for the attacker's own Android device
Add victim's account and enter random password
Intercept and modify /auth message by changing the EncryptedPasswd parameter to the intercepted victim's parameter
Let the account adding process flow and access the victim's account from the attacker's own device as the victim would

Figure 7.4 *Information gained in the attack and how to use it.*

Generally authorization tokens have a limited access to the user’s data. Tokens are presumably restricted only to the information that the specific application (which the token belongs to) needs. A master token on the other hand is used to acquire tokens for all Google’s Android applications and victim’s encrypted password is present when master token is acquired during the account registration process.

In general, Google uses only SSL and authorization token to protect user’s data, when a Google Android application accesses it. An attacker can, by looking at the previously made request and slightly modifying them to handcraft own request and access victim’s data. In practice the messages can be handcrafted in a text editor and sent manually using OpenSSL, which forms an SSL/TLS tunnel to Google’s servers and is capable of sending messages.

The spying, if done right in practice, stops when the victim changes her Google account password and before that the victim could be spied upon for years. In the case of phones, according to Roger Entner [50, 51] handset replacement cycle in 2010 and 2012 in United States was 21.7 months (roughly 2 years), but it also heavily depends on the country. For example, in Finland the replacement cycle in 2010 was 74.5 months (roughly 6 years). [50]

7.2.1 Exploiting the gained information - Custom requests

During the observations two tests were made, where information acquired in the MITM attack was used to reveal more information of the victim than was communicated during the MITM attack. The victim in tests was the author’s test Google account. Before tests, two contacts and one calendar event was added to the victims Google account for the attacker to target.

In the first test, after the MITM attack the attacker’s computer changed its IP-address, made an SSL/TLS tunnel connection with OpenSSL to Google’s servers and send handcrafted requests, with victim’s authorization tokens. First request was used to test whether a replay attack would work. However, the request the attacker made was not completely identical with the original message. The attacker’s message was slightly modified in order to get reply in a plain text and retrieve more interesting information. The first request was used to obtain all of the victim’s contacts (e.g. phone numbers, email addresses, etc.) stored in the Google servers. The request made by the attacker used the same token as the victim had used.

Second request was used to test whether the attacker could obtain new tokens for a Google’s application or service. The captured GLS and GA token request messages contained all the necessary information for the attacker to make modified requests. The second request was used to obtain a new token for Google calendar application.

Third and final request tested whether the attacker could use the newly acquired token to make request and obtain more information of the user. The sent request was used to obtain all the calendar events the victim had made. The second test was identical to the first test, but this time the victim used Google’s 2-step verification, where the login information came from the Google Authenticator. Nevertheless all the requests were made successfully in both tests. The requests made in the first test are presented in Appendix A.

While, not every Google's Android application was tested, these tests proved that once a user has been once a victim of a MITM attack capable of decrypting SSL/TLS communications the victim could then be spied upon remotely. If the attacker knows the victim's master token, the author believes that then in practice, the attacker has access to victim's all Google data (same data as the victim's own device has access to), since the attacker is able to acquire new tokens for victim's Google Android applications. On the other hand the attacker has to familiarize herself with how Google's applications communicate with Google's servers in order to be able to make custom requests.

This attack does not leave traces for the victim to notice (except the new trusted CA certificate in the device). No indication was noticed in the Google account's security related pages that another device had made queries or used the victim's account, for example, from another IP-address. Also the Google Android application's behaviour helps to hide the possible attack, since presumably one token is only valid per application, and application's default behaviour is to acquire new token when it has problem with the old one and user is not informed of these new requests or invalid tokens.

7.2.2 Exploiting the gained information – Victim's encrypted password

With the knowledge of the victim's encrypted password the author was able to add the victim's Google account to another device and use the Google's applications with victim's credentials without actually knowing the victim's password. The attack was done as follows.

The attacker starts by adding the victim's account to her own device as the victim would add her own account. The attacker adds the victim's Google account name, which she got from any of the captured token requests (Listings 5.2, 5.4 and 5.6, parameter Email) and types a random string for the password field (the typed password does not matter). After accepting the terms of service, etc. the device starts the registration process by sending a master token request (Listing 5.2). At the same time the attacker performs a MITM attack to her own device and stops the master token request message. She changes the EncryptedPasswd parameter (which holds the user's encrypted password) value to the one she got earlier in the MITM attack from the victim. After changing the value she lets the message go to the Google's servers and lets the process flow in its own weight. A few minutes later the attacker is able to use the victim's account as she would.

This attack makes it easier for the attacker to spy or do other malicious acts, because she does not have to familiarize herself to how Google's Android applications communicate and make handcrafted requests. On the other hand, this attack leaves much bigger traces of the attack. The attacker's device gets listed to the victims Google accounts – apps and permissions list, where the victim can also revoke the attacker's device's rights. However, merely revoking the attacker's device's right to access victim's account does not prevent the attacker from adding the device again, since the attacker knows the victim's encrypted password and is therefore able to add victim's account again.

Also this attack does not work when the victim uses 2-step verification. Namely, in order to complete the registration process the attacker needs the one time password (OTP) given by Google, which is usually given to the phone number found in the victim's Google account details.

7.3 The weaknesses and their prevention

Google's Android applications and Android itself have the following problems that made the previously described attacks possible: messages are lacking authentication and integrity checking, Android applications in the device blindly trust every CA certificate, possibility to use encrypted Google account password more than once.

7.3.1 The weaknesses

Messages lacking authentication and integrity checks mean that the party sending HTTP requests to Google's servers is not authenticated nor are messages' integrity secured during the transit. The lack of these measures makes it possible for the attacker to send her own requests and modify all the communications between the victim and Google's servers without either of the parties knowing.

Another issue was that a user-added and trusted SSL CA certificate is also trusted by default nearly every Android application (not just Google's) in the device. This makes it possible for the attacker to perform MITM attack capable of decrypting all SSL communications.

The last problem is that a user's encrypted password sent during the registration is valid more than once and possibly for as long the password stays the same. For example, the author used a 7 months old encrypted password in a demonstration attack.

7.3.2 What a user can do to prevent or stop a MITM attack?

For a user to stop an ongoing spying she needs to change the master token and preferably her password, since the attacker can spy the victim as long as the master token and the password are valid. The master token can be changed by at least three ways: revoking the device's right to access the user's account, starting to use 2-step authentication and changing the Google account password.

The device's right to access the user's Google account can be revoked from Google account security settings regarding applications and websites permissions [52]. Also the 2-step verification is deployed from the Google account security settings. Both of these mechanisms helps only *after* the user has been a victim of the MITM attack, because the user is given a new master token and the older token, which the attacker has, is invalidated.

The easiest and the most effective way to protect a user temporarily from being spied are to change the user’s Google account password, because it automatically invalidates all permissions that among other things, any device has to the user’s account. This means the attacker cannot use the captured master token and the victim’s encrypted password anymore. Changing the password also works until the victim gets attacked by a MITM attack again.

For a user to protect herself from a MITM attack, she can: keep the device under a close watch, check the trusted CA certificates time to time or use a device with an Android OS 4.4 version or newer. It is not feasible to require for a user to keep a close watch on her device all the time. Social engineering and exploiting a trust with the victim are also possible means to take victim’s attention away. Also for the user to check trusted CA certificates requires a lot of understanding of SSL and Android security, which is not a reasonable expectation from an everyday user. And even in the case when a victim finds a rogue trusted CA certificate, the attack might have already happened. In practice using a newer Android OS is the best solution, because Google has made security improvements to OS 4.4 version, which should prevent the MITM attack. These improvements are discussed in depth in next section.

7.3.3 What steps has Google taken to protect the user?

At the time of writing Google has already taken steps to protect its users. In Android version 4.2 Google introduced feature a called Certificate pinning [53]. Certificate pinning is according to OWASP [54] a process of associating hosts with their expected X.509 certificate or public key. This means that it is used to give an application the ability to trust only certain determined CA SSL certificates [55], which prevents MITM attack because the rogue CA certificate will not be trusted by applications in the Android device. In Android the certificate pinning is implemented by maintaining a list of SHA1 hashes of trusted certificate’s public keys [55]. In practice the hash is calculated from certificate’s SubjectPublicKeyInfo (SPKI) field and is stored in the device in the following way. Enforcing is either true or false and is followed by SHA512 hashes separated with comma. [55]

```
Hostname=enforcing|SPKI SHA512 hash, SPKI SHA512 hash,...
```

Google added more security features in Android OS 4.4 version and the security enhancements included notification related to certificate pinning: "Android 4.4 detects and prevents the use of fraudulent Google certificates used in secure SSL/TLS communications." Another important security enhancement in Android 4.4 was device monitoring warnings, which "provides users with a warning if any certificate has been added to the device certificate store that could allow monitoring of encrypted network traffic." [56] In general, author believes the certificate pinning and the system warning message are good features. Certificate pinning prevents the MITM attack against certain host before it even starts. And system warning messages clearly warn the user of a potential MITM attack.

However, certificate pinning feature and warning messages themselves are not enough to protect users, if they are not used. For example, apparently Google does not use certificate pinning feature in its own applications except in Android version 4.4. Otherwise author would not have been able do this work with Android OS 4.3 and 4.2 devices. Also at the time of writing only one third-party application in the tested Google Android OS 4.3 device was noticed not to work in the test environment and this behaviour was presumed to be a result from the certificate pinning. On the other hand the device was stripped from unnecessary applications when possible in order to keep capture files "as clean as possible".

Also at the moment of writing only the newest Android OS 4.4 version has these features. This is an issue for older Android OS version users, because only the newest Android devices get updates to new versions of the OS. This has led to a problem called Android OS fragmentation. Even now after a year Google released Android OS 4.4 [57] it is still used roughly only in 21% of all of the Android devices [58]. So it will take time until the majority of the Android users are protected.

7.3.4 What a service provider can do to prevent a MITM attack?

The author presents the following proposal, which might or might not be appropriate for a service provider, but it prevents the described attacks. In-depth analysis of the proposal is out-of-scope of this thesis. The author proposes for a service provider to use digital signatures in messages from both the client and the server to protect users from the observed aftereffects of the MITM attack. Digital signature is useful because it authenticates the party sending a message and gives integrity protection for the message. For the attacker to make her own request she needs to get the victim’s private key used for signing messages, which does not need to be present in the communications at all. Hence this would prevent the attacker from making her own requests successfully to Google’s servers.

As stated earlier, Amazon uses digital signatures in nearly every request made to Amazon’s servers, but their implementation should not be completely copied, because the key distribution the author presumes Amazon using is not secure enough to protect from a MITM attack. The problem with Amazon’s key distribution is that Amazon’s server gives the private key used for signing during the registration to the user’s device and it is given in plaintext. Hence the key would leak to the attacker, if registration was done under a MITM attack. Author personally believes the MITM attack happening during the first registration is highly unlikely, but apparently it is not impossible for the attacker to gain the same information. The attacker can deceive victim’s device to "force" the victim to register again, as it was demonstrated, and gain all tokens and the key used for signing.

Instead of service provider giving the private key for signing to the user's device let the user's device create the keys for signing ($K_A^+$and K_A^-) and give the public key K_A^+ in encrypted form during the registration process when the master token is acquired. Using Google Android as an example, Google's Android devices already have Google's public key K_G^+, which is used to encrypt the user's password [28]. The process to deliver the user's device signing key K_A^+ could be the following:

`...&EncryptedPasswd=`$\{K_S\}_{K_G^+}$`&EncryptedContent=`$\{password, K_A^+\}_{K_S}$`...`

According to Elenkov [28] the Google's public key resembles 1024-bit RSA public key and Google uses OAEP for padding and therefore can encrypt at maximum 86 bytes of data [28]. Due the limitation of how much data can be encrypted with Google's RSA private key, the master token request requires few minor changes. First changes should take the user's password away from EncryptedPasswd parameter and replace it with a symmetric encryption key K_S, which is encrypted with Google's public key to ensure only Google can retrieve it. Then a new parameter EncryptedContent is added to the master token request. The new parameter contains user's password for authentication and her public key K_A^+, which Google can use to verify the user's devices signatures. The content of the new parameter is encrypted with a symmetric encryption algorithm using the key K_S.

Using signatures and giving the signing key as described would fix two other problems: encrypted password reusability and deceiving the victim's device to do registration again. The password reusability is prevented in the case where the attacker copies the parameters EncryptedPasswd and EncryptedContent to her own request (in order to use victim's Google account in her own Android device). Because the attacker does not have the user device's private key K_A^-, she cannot make valid signatures for subsequent messages. And deceiving the victim's device would not be possible if Google would also sign sent messages, because the attacker does not know the Google's private key K_G^- and therefore is not able to make valid replies. Google already has the capability to sign requests, as seen in Google Play Store Listings 5.23 and 5.27.

Digital signature is not however a silver bullet in fixing these problems. Signatures require extra processing power from both the user's Android device and Google's servers, which might be an issue for Google who has over 1 billion active Android users at the time of writing in 2014 [59].

8 GOOGLE AND AMAZON COMPARISON

The chapter is divided in three parts. First and second part summarise the found security features in Google's and Amazon's cases. In the third section Google's and Amazon's security features are compared and discussed.

8.1 Google security features summary

Table 5.1 summarizes the security features that Play Store was observed to use when buying and downloading free and paid software. Presented features are restricted to those features that were directly observable in the communication channel. Main differences were in the usage of SSL/TLS and in the additional authentication of the communicating parties.

Table 5.1: Observed security features of free and paid application cases in Google.

Security feature	Free application case	Paid application case
Confidentiality	SSL/TLS used in the communication, except in the final delivery of the application.	SSL/TLS used during the whole communication. And probably some kind of encryption is used with prepare- and commitPurchase messages.
Authentication: Server	Client presumably authenticates servers with signature requests at the beginning of the message flow.	Same as in free application case, but also additional server authentication when user committed to purchase of the application.
Authentication: User	The user is never authenticated.	The user is authenticated, when he/she decides to buy the application.
Authorization:	Every message sent by user's device except the last one used to download the application has a token included.	Every message sent by user's device except ClientLogin authentication had a token included.
Availability: Time-out value	Every message sent by user's device except application delivery messages (Messages 6 and 8,	Every message sent by user's device except ClientLogin authentication

	Figure 5.5) had time-out value in HTTP header.	and final application delivery message (Messages 6 and 12, Figure 5.6) sent from client to Google had time-out value in HTTP header.
Integrity: Message	Message integrity is not protected.	Message integrity is not protected.
Integrity: Application	Possibly signature was used. (Discussed below in detail.)	No protection was observed, but messages contained parts which were not fully understood. (Discussed below in detail.)

SSL/TLS was used to secure the whole communication between the user's device and Google, when the user buys an application from Play Store. In free application case the SSL/TLS was also used, but not in the last two messages, which were part of the application download.

In free and paid application cases the client application in user's device authenticated the Google server it communicated with by sending a signature request during the Play Store start-up. In paid application case Play Store makes one additional signature request, when the user has committed to purchase an application.

In the paid application case the user was authenticated when the decision to buy the application was made. Play Store used Google's own proprietary mechanism ClientLogin for this purpose. In the free application case the user is not authenticated at all.

Authorization was done with tokens, which are used between free and paid cases similarly. In paid case every message except the ClientLogin message sent by the user's device included a token. In free application case every message sent by the user's device included a token except the final message requesting the application packet download.

Also time-out values were sent along similarly in nearly every message sent from the user's device in free and paid cases. The exceptions were in free application case the final two messages used to download the application. In paid case also the final application download request and the ClientLogin message used for user authentication did not have time-out value in the message.

Google was not observed to use any means to protect the message integrity in the application layer in free and paid application cases. However, determining whether any application integrity protection is used was not clear. In paid application case no protection was observed, but messages contained parts which were not understood. For example, reply message for preparePurchase message contained over 5400 character long base64 encoded string, which could easily hold signatures or hash values of the application. The free application case is also unclear on the application's integrity protection. The last message used to request and download the application contained a signature parameter in the URL. However it was left unknown what the signature was for (e.g. URL or the packet) and what parts the signature covered.

8.2 Amazon security features summary

In Amazon the observed security features were identical with free and paid software cases. Table 5.2 presents the findings.

Table 5.2: Observed security features in Amazon case.

Security feature	**Free and paid application cases**
Confidentiality	SSL/TLS used during the whole communication.
Authentication: Server	Client does not authenticate server it communicates with.
Authentication: User	The user is never authenticated, when the decision to buy or download an application is made.
Authorization:	Every message except the createAuthtoken and certain metadata requests has a token included.
Availability: Time out value	Time out values are not used.
Integrity: Message	Nearly every message that has an x-adp-token included contains also a digest or signature of the message.
Integrity: Application	The application's hash value is provided before the download.

Amazon uses the SSL/TLS to secure confidentiality of the whole communication between a client (user's device) and a server (Amazon). Certain image downloads and reachability test URLs were not protected by the SSL/TLS.

The Amazon shop client application does not authenticate the server it communicates with. The shop application also does not authenticate the user, when the decision to download a free application or purchase an application is made.

Nearly every message has a token included, presumably just for client applications authorization to use Amazon services. X-ADP-token is also always (with one exception) accompanied with message digest or signature value, which protects the message from tampering and corruption during the transit from user's device to Amazon. However, it was not known what part of the message the hash or signature covers. Application's integrity was protected by telling the application file's hash value at the same time when the download URL was given.

8.3 Comparison of Google and Amazon

The section is divided in two parts. The first part compares the security in general between Google and Amazon. The second part concentrates on security of the market places.

8.3.1 Security in general

In general, Google and Amazon used tokens in the same way for authorization. First, a master token or refresh token was acquired, when the user's account was registered to the Android device. Then the master token or refresh token was used to acquire tokens for applications. One difference between Google's and Amazon's token acquirement process was that certain Amazon's applications got tokens during the registration process and the tokens were not acquired with the refresh token.

Amazon's client applications were the only ones to use signatures (and possibly hash functions) to protect the integrity of the client's messages to Amazon and in the case of signatures, also to authenticate the sender. There might be the issue of how the key used for signing is delivered to the device during the registration (see Section 6.3.2). However, the delivery method for the key was not determined conclusively. The author believes that the key distribution is not a big problem, unless the attacker can force the user to register herself again to the device in order to intercept the signing key. However, this possibility was not tested.

The usage of the signature in a message is a clear security advantage for Amazon and for their users, if the SSL/TLS fails. As it was demonstrated in the case of Google's device (see Chapter 7), when SSL/TLS protection fails the attacker can get access to nearly all the information the victim has in her Google account. From the perspective of the communication security, it can be said that Google lacks defence in depth.

8.3.2 Market place specifics

Biggest differences between Google's and Amazon's market places were due to the fact that Google Play Store worked differently depending on whether the user was acquiring a paid or a free application. Both Google's and Amazon's market places used heavily SSL/TLS and authorization tokens. Amazon used both the whole time and Google used it also the whole time except in the free application case, when the application was downloaded. Other exceptions in SSL/TLS and token usage were in certain image downloads and reachability tests.

Amazon Shop did not authenticate the user at any point, when free or paid application was bought and downloaded. In Google’s case it depended on the application’s price. If the application was bought, then the user was authenticated in Google’s Play Store, but not in the free application’s case. The Google’s authentication mechanism (ClientLogin) in Play Store had one oddity: the user’s password was sent in plain text inside the SSL/TLS connection. The author felt this was odd, because during the registration, which happens rarely compared to buying applications from Play Store, the user’s password is sent encrypted in the SSL/TLS connection.

The Amazon Shop’s lack of authentication might be a problem when, for example, a user’s device has been stolen. The thief can buy, for example, applications from the shop in the name of the victim. However, this case requires two things: first the victim has to have given her credit card details to her Amazon’s account and the thief has to get past the device’s screen lock, if it has been enabled (by default it is off).

Google Play Store client application was the only one to check with whom it was communicating by making a signature request, and the only one to use time-out values for messages. However, the signature request was made only once and only for the requested nonce. This means that, in practice, it does not prevent the attacker from modifying the message as long as she does not touch the nonce that the client application sends and the signature that the Google’s server sends in response.

The application integrity comparison is not clear. Amazon clearly gives the application’s hash value. Google was observed only in free application case to give a signature, but it was not determined whether the signature was for the application or the URL. In paid application case no integrity protection mechanism was observed, but then again certain parts of the messages were not understood.

9 CONCLUSIONS

The goal of this thesis was to identify and analyse, what security mechanisms Android vendors, Amazon and Google, use during the communication when their own Android applications communicate with their services. The adequacy of the found security mechanisms for the observed applications was out of the scope of this thesis.

In general, both Google and Amazon rely on authorization tokens and SSL/TLS protocol to protect the communicated information. Amazon's client applications in the Android device were noticed to use signatures to provide message integrity protection and authenticate the sender.

The security and general operation of the market applications were similar. The SSL/TLS was used all the time, except in Google Play Store it depended on whether the application was free or not. Other differences were on the requirement of user authentication during the purchase. Amazon does not require the user to authenticate on purchase, which means that, security wise, the user of Amazon's device has to take greater care of her own device than Google's user.

During the authentication, when the user buys an application from Play Store, the client application sends the user's password in plaintext inside the secured SSL/TLS connection. During the less frequently happening registration of the user's Google account to the device the user's password is sent instead encrypted inside the SSL/TLS connection.

The Google's device and the authors Google's account was attacked to demonstrate in practice, what the attacker can do and achieve, when the SSL/TLS protection fails. Security weaknesses were found and informed to Google. The weaknesses give the attacker ability to remotely access to nearly all the victim's Google data, for now. Also the attacker can "force" the victim to register again to the Android device, and the attacker can use the victim's intercepted encrypted password to add the victim's Google account to her own Android device. However, it should be noted that the attack used in the demonstration requires for the attacker to have physical access to the device. The author proposes using digital signatures and usage of Android OS 4.4 version or newer to counter the threat.

The author believes that the goal set for the thesis was reached and the work went well, even though there were some issues. The challenges and critiques of the work method in making this thesis were lack of documentation and manual labour. There is, in practice, no documentation available (only bits and pieces) of Google's and Amazon's applications and security mechanisms. Hence, a lot of the time was just spent to understand how things worked and what message was related to what application, etc. Google’s habit to change the way how things worked did not help either. These are also the reasons why so many things were left unknown.

The lack of documentation is where the manual labour came in. For example, in a very simple case where Amazon Android device is booted, waited 30 seconds on the home screen, and then shut down, the device sends circa 3000 packets, which can be filtered to circa 110 interesting messages that all have to be examined manually. The longest capture files had over 800 interesting packets. Over 60 communications were captured and examined, and number is quite big, because the author seemed to find all the time something that had to be verified in a new capture, before it could be written in the thesis. Also, when the understanding of how the system and applications worked was achieved, then the captured communications were not studied as closely, and because of this, a few exception cases were found later on by accident. Hence, it is possible that some things might have been missed.

Future work on the subject could be done in determining whether Amazon’s devices can be forced to authenticate the user again and what the attacker can achieve with manipulating the packets during the MITM attack. For the Google Play Store, the free application download security could be examined more closely, since the application download does not use the SSL/TLS protocol, which might leave the download vulnerable for poisoning attacks.

REFERENCES

[1] W. Stallings, Cryptography and Network security - Principles and practice, London: Pearson Education, Inc., 2011.

[2] Canalys, “Google's Android becomes the world's leading smart phone platform,” Canalys, 31 January 2011. [Online]. Available: http://www.canalys.com/static/press_release/2011/r2011013.pdf. [Accessed 24 October 2014].

[3] Phonearena, “Android forks are now 20% of the ecosystem. What is Google's plan?,” Phonearena, 5 August 2014. [Online]. Available: http://www.phonearena.com/news/Android-forks-are-now-20-of-the-ecosystem.-What-is-Googles-plan_id59003. [Accessed 24 October 2014].

[4] R. Shirey, “Internet Security Glossary, Version 2,” RFC 4949, IETF, 2007.

[5] Google, “Google Developers: Google Accounts Authentication and Authorization,” [Online]. Available: https://developers.google.com/accounts/. [Accessed 8 September 2014].

[6] E. Barker and J. Kelsey, “Draft NIST Special Publication 800-90A, Rev. 1 - Recommendation for Random Number Generation Using Deterministic Random Bit Generators,” National Institute of Standards and Technology, 21 April 2014. [Online]. Available: http://csrc.nist.gov/publications/drafts/800-90/sp800_90a_r1_draft.pdf. [Accessed 8 October 2014].

[7] R. Anderson, Security Engineering - A guide to building dependable distributed systems, 2nd ed., Indianapolis: Wiley Publishing, Inc, 2008.

[8] IANA, “Hypertext Transfer Protocol (HTTP) Authentication Scheme Registry,” IANA, 17 February 2014. [Online]. Available: http://www.iana.org/assignments/http-authschemes/http-authschemes.xhtml. [Accessed 26 October 2014].

[9] R. Fielding and J. Reschke, “Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content,” RFC 7231, IETF, 2014.

[10] J.-S. Coron, D. Yevgeniy, C. Malinaud and P. Puniya, “Merkle-Damgård Revisited : how to Construct a Hash Function,” 2005. [Online]. Available: https://www.cs.nyu.edu/~puniya/papers/merkle.pdf. [Accessed 28 October 2014].

[11] R. Rivest, “The MD5 Message-Digest Algoritm,” RFC 1321, IETF, 1992.

[12] Carnegie Mellon University, “CERT - Vulnerability Note VU#836068 - MD5

vulnerable to collision attacks," 31 December 2008. [Online]. Available: http://www.kb.cert.org/vuls/id/836068. [Accessed 9 October 2014].

[13] H. Krawczyk, M. Bellare and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," RFC 2104, IETF, 1997.

[14] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 5280, IETF, 2008.

[15] Netscape Communications Corporation, "The SSL Protocol," Netscape Communications Corporation, March 1996. [Online]. Available: https://web.archive.org/web/19970614020952/http://home.netscape.com/newsref/std/SSL.html. [Accessed 25 October 2014].

[16] R. Amadeo, "Google's iron grip on Android: Controlling open source by any means necessary," Ars Tecnica, 21 October 2013. [Online]. Available: http://arstechnica.com/gadgets/2013/10/googles-iron-grip-on-android-controlling-open-source-by-any-means-necessary/. [Accessed 1 November 2014].

[17] Open Handset Alliance, "Press - Industry Leaders Announce Open Platform for Mobile Devices," Open Handset Alliance, 5 November 2007. [Online]. Available: http://www.openhandsetalliance.com/press_110507.html. [Accessed 1 November 2014].

[18] Open Handset Alliance, "OHA - Members," Open Handset Alliance, [Online]. Available: http://www.openhandsetalliance.com/oha_members.html. [Accessed 1 November 2014].

[19] Internatiolan Data Corporation, "Smartphone OS Market Share, Q2 2014," Internatiolan Data Corporation, 2014. [Online]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp. [Accessed 1 November 2014].

[20] The OpenSSL Project, "OpenSSL - Cryptography and SSL/TLS Toolkit," 2014. [Online]. Available: https://www.openssl.org/. [Accessed 29 October 2014].

[21] D. Roethlisberger, "SSLsplit - transparent and scalable SSL/TLS interception," 2014. [Online]. Available: https://www.roe.ch/SSLsplit. [Accessed 29 October 2014].

[22] A. Cortesi, "mitmproxy: a man-in-the-middle proxy," 2013. [Online]. Available: http://mitmproxy.org/index.html. [Accessed 29 October 2014].

[23] Wireshark Foundation, "About Wireshark," [Online]. Available: https://www.wireshark.org/about.html. [Accessed 29 October 2014].

[24] Offensive Security Ltd., "Kali Linux," Offensive Security Ltd., 2014. [Online]. Available: http://www.kali.org/. [Accessed 29 October 2014].

[25] Amazon, "Kindle Fire Device and Feature Specifications," Amazon, 2014. [Online]. Available: https://developer.amazon.com/sdk/fire/specifications.html.

[Accessed 30 October 2014].

[26] mitmproxy project, “How mitmproxy works,” mitmproxy project, 2014. [Online]. Available: http://mitmproxy.org/doc/howmitmproxy.html. [Accessed 30 October 2014].

[27] D. Eastlake, “Transport Layer Security (TLS) Extensions: Extension Definitions,” RFC 6066, IETF, 2011.

[28] N. Elenkov, “Android online account management,” 5 11 2012. [Online]. Available: http://nelenkov.blogspot.jp/2012/11/android-online-account-management.html. [Accessed 3 September 2014].

[29] Google, “Android Developers: Reference - AccountManager,” [Online]. Available: http://developer.android.com/reference/android/accounts/AccountManager.html. [Accessed 8 September 2014].

[30] Google, “Android - Frequently Asked Questions: Compatibility,” [Online]. Available: http://source.android.com/source/faqs.html#compatibility. [Accessed 8 September 2014].

[31] P. Bright, “Neither Microsoft, Nokia, nor anyone else should fork Android. It's unforkable.,” Arstechnica, 8 February 2014. [Online]. Available: http://arstechnica.com/information-technology/2014/02/neither-microsoft-nokia-nor-anyone-else-should-fork-android-its-unforkable/. [Accessed 8 September 2014].

[32] Google, “Google Developers: Google Play Services,” [Online]. Available: http://developer.android.com/google/play-services/index.html. [Accessed 8 September 2014].

[33] R. Amadeo, “Balky carriers and slow OEMs step aside: Google is defragging Android,” Arstechnica, 3 September 2013. [Online]. Available: http://arstechnica.com/gadgets/2013/09/balky-carriers-and-slow-oems-step-aside-google-is-defragging-android/. [Accessed 8 September 2014].

[34] Google, “ClientLogin for Installed Applications,” [Online]. Available: https://developers.google.com/accounts/docs/AuthForInstalledApps?hl=en. [Accessed 8 September 2014].

[35] Google, “Google Developers: YouTube API v2.0 - ClientLogin for Installed Applications,” [Online]. Available: https://developers.google.com/youtube/2.0/developers_guide_protocol_clientlogin. [Accessed 8 September 2014].

[36] Google, “Google 2-Step Verification,” Google, [Online]. Available: https://www.google.com/landing/2step/. [Accessed 31 October 2014].

[37] Google, “Using Security Key for 2-Step Verification,” Google, 2014. [Online]. Available: https://support.google.com/accounts/answer/6103523?hl=en.

[Accessed 31 October 2014].

[38] Google, “Types of cookies used by Google,” [Online]. Available: http://www.google.com/intl/en/policies/technologies/types/. [Accessed 8 September 2014].

[39] K. Sriram, “Sbktech: Inside the Android Play Service's magic OAuth flow,” January 2014. [Online]. Available: http://sbktech.blogspot.fi/2014/01/inside-android-play-services-magic.html. [Accessed 24 September 2014].

[40] Google, “Android Developers: Reference - Settings.Secure,” Google, [Online]. Available: http://developer.android.com/reference/android/provider/Settings.Secure.html#ANDROID_ID. [Accessed 24 September 2014].

[41] Google, “Google Plus: Android Developers - Google Play services,” 26 September 2012. [Online]. Available: https://plus.google.com/+AndroidDevelopers/posts/J1A5hc1ZnS1. [Accessed 8 September 2014].

[42] Google, “Google Developers: Google Accounts Authentication and Authrorization - AuthSub for Web Applications,” [Online]. Available: https://developers.google.com/accounts/docs/AuthSub. [Accessed 8 September 2014].

[43] N. Elenkov, “Single sign-on to Google sites using AccountManager,” 9 November 2012. [Online]. Available: http://nelenkov.blogspot.fi/2012/11/sso-using-account-manager.html. [Accessed 8 September 2014].

[44] Y. Soo, "6th Kandroid conference - Google GMS (Google Mobile Services)," 15 October 2010. [Online]. Available: http://www.kandroid.org/board/data/board/conference/file_in_body/1/6th.kandroid.conference.gms.analysis.pdf. [Accessed 21 October 2014].

[45] Y. Soo, “7th Kandroid conference - Google GMS (Google Mobile Services),” 11 March 2011. [Online]. Available: http://www.kandroid.org/board/data/board/conference/file_in_body/1/7.session.7th.kandroid.gms.analysis.final.pdf. [Accessed 21 October 2014].

[46] J. Jonsson and B. Kaliski, “Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1,” RFC 3447, IETF, 2003.

[47] Amazon, “Amazon Web Services - AWS Documentation: Signature Version 2 Signing Process,” [Online]. Available: http://docs.aws.amazon.com/general/latest/gr/signature-version-2.html. [Accessed 8 September 2014].

[48] Amazon, “Amazon Web Services - Aws Documentation: Types of Security Credentials,” [Online]. Available: http://docs.aws.amazon.com/general/latest/gr/aws-sec-cred-types.html. [Accessed 8 September 2014].

[49] Y. Song, C. Yang and G. Gu, “Who is peeping at your passwords at Starbucks? — To catch an evil twin access point,” in *Dependable Systems and Networks (DSN)*, Chigago, 2010.

[50] E. Roger, “International Comparisons: The handset replacement cycle,” 2011. [Online]. Available: http://mobilefuture.org/wp-content/uploads/2013/02/mobile-future.publications.handset-replacement-cycle.pdf. [Accessed 11 November 2014].

[51] E. Roger, “Entner: Handset replacement cycles haven't changed in two years, but why?,” Recon Analytics, 18 March 2013. [Online]. Available: http://www.fiercewireless.com/story/entner-handset-replacement-cycles-havent-changed-two-years-why/2013-03-18. [Accessed 11 September 2014].

[52] Google, “Google accounts security - Account permissions,” [Online]. Available: https://security.google.com/settings/security/permissions?pli=1. [Accessed 6 October 2014].

[53] Google, “Android Source - Security Enhancements in Android 4.2,” [Online]. Available: https://source.android.com/devices/tech/security/enhancements42.html. [Accessed 8 September 2014].

[54] OWASP, “Certificate and Public Key Pinning,” OWASP, 14 August 2014. [Online]. Available: https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning. [Accessed 13 October 2014].

[55] N. Elenkov, “Certificate pinning in Android 4.2,” 12 December 2012. [Online]. Available: http://nelenkov.blogspot.fi/2012/12/certificate-pinning-in-android-42.html. [Accessed 13 October 2014].

[56] Google, “Android Source - Security Enchancements in Android 4.4,” [Online]. Available: https://source.android.com/devices/tech/security/enhancements44.html. [Accessed 8 September 2014].

[57] Google, “Android developers blog - Android 4.4 KitKat and Updated Developer Tools,” 31 October 2013. [Online]. Available: http://android-developers.blogspot.ca/2013/10/android-44-kitkat-and-updated-developer.html. [Accessed 13 October 2014].

[58] Google, “Android Developer - Dashboards: Platform Versions,” [Online]. Available: http://developer.android.com/about/dashboards/index.html#Platform. [Accessed 8 September 2014].

[59] Google, “Youtube - Google I/O 2014 - Keynote,” Google, 25 June 2014. [Online]. Available: http://www.youtube.com/watch?v=wtLJPvx7-ys#t=368. [Accessed 12 October 2014].

APPENDIX A

Appendix A presents 3 different custom requests made to Google with their response. The first one requests all the victim’s contacts. Second is a token request for Google calendar application and in third request the same token is used to retrieve all the victim’s calendar events.

A.1 Request and response for all of the victim’s contacts

```
~$ openssl s_client -connect android.clients.google.com:443
CONNECTED(00000003)
depth=2 C = US, O = GeoTrust Inc., CN = GeoTrust Global CA
verify error:num=20:unable to get local issuer certificate
verify return:0
---
Certificate chain
 0 s:/C=US/ST=California/L=Mountain View/O=Google Inc/CN=*.google.com
   i:/C=US/O=Google Inc/CN=Google Internet Authority G2
 1 s:/C=US/O=Google Inc/CN=Google Internet Authority G2
   i:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
 2 s:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
   i:/C=US/O=Equifax/OU=Equifax Secure Certificate Authority
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIHUzCCBjugAwIBAgIIANgyk4MznC ...
-----END CERTIFICATE-----
subject=/C=US/ST=California/L=Mountain View/O=Google Inc/CN=*.google.com
issuer=/C=US/O=Google Inc/CN=Google Internet Authority G2
---
No client certificate CA names sent
---
SSL handshake has read 4472 bytes and written 434 bytes
---
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES128-GCM-SHA256
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES128-GCM-SHA256
    Session-ID:
8EED3E0C3A24E7265F78B66544286E63AD86EFD898676A80D650B29421A65447
    Session-ID-ctx:
    Master-Key:
9817D8DA54F54A90DBF92D0B4BACBF06590086FEF8487F0A4D89100646B422BDF060502575A066
60193810131F211B25
    Key-Arg   : None
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 100800 (seconds)
    TLS session ticket:
    0000 - e3 00 d6 10 60 44 fb 6a-1f cd 02 31 fe e8 5a cd   ....`D.j...1..Z.
    0010 - 44 e2 e6 ff dd 0c 86 ba-c1 87 ef 13 bc 39 f0 27   D............9.'
    0020 - aa 61 c9 3f a9 b4 4e cd-d2 3e b2 b6 1d b6 40 d4   .a.?..N..>....@.
    0030 - a5 59 29 d9 a6 93 fb 7b-c6 0d 91 9d 6f 69 6b 58   .Y)....{....oikX
    0040 - 8b 9e a4 8e 3a 55 d6 ff-11 92 55 6d ba aa 77 81   ....:U....Um..w.
    0050 - a8 da 17 58 4d 0b 5f e1-fa 8b d4 98 9e 58 8e bc   ...XM._......X..
    0060 - e9 d5 0c 25 03 55 f7 a8-0b 9c 6c 47 cc aa 89 c3   ...%.U....lG....
    0070 - 4b 93 f0 00 3b f6 97 c6-2c c6 fb e9 12 96 3d b5   K...;...,.....=.
    0080 - b2 93 14 5a 20 fc e9 85-af e1 24 35 e7 6a 22 1a   ...Z .....$5.j".
    0090 - 25 ef 56 00 f8 ef 7b f5-c8 dc c7 7c 42 a1 c3 75   %.V...{....|B..u
    00a0 - 8d 12 d9 d5                                       ....

    Start Time: 1408709028
    Timeout   : 300 (sec)
    Verify return code: 20 (unable to get local issuer certificate)
---
GET    /proxy/contacts/contacts/*****@gmail.com/base2_property-android_linksto-
gprofiles_highresphotos?sz=720&showdeleted=false&orderby=lastmodified&sortorde
r=ascending&max-results=500 HTTP/1.1
Accept-Encoding: identity
Authorization:                                                     GoogleLogin
auth=DQAAAPsAAABmVviowlWXVULUgHnitLKbrhiDFkgfd5zzGp4WPwvrrLcIqrSQNLPQHtP1jVeBc
x_GbZDrXC96UvdcEyWYSJNBhOHA07Ph3Pa68yaNVWbYhJbx4BtmqYS6x9T8TWpDh1l5sGNJUtSMm9R
3-XxiB9VmJJ2yUdyX2H0_ZKGdEVgBI5iF1ySbwmsPzGaBMG42iukn8ggw0h6xkWf1HBjTJwGNvqNJQ
DVZiaREToB1DpXYj-ReHHhAb_YUsHIVaL9d1fsQC8Au_OL5KG9fp2QU2pombYk2vZmeXOjfZv6Vq4c
nJuVrwI8sTKWIr7KBgY7cLuBgf7ScXyZimcWsqS4fwWC
GData-Version: 5.0
Host: android.clients.google.com
Connection: close
User-Agent: Android-GData-Contacts/1.3 (m3 JSS15J);

HTTP/1.1 200 OK
Expires: Fri, 22 Aug 2014 12:03:53 GMT
ETag: "Qn4zcDVSLy17ImA9XRZbF00DRwc."
Content-Type: application/atom+xml; charset=UTF-8
Date: Fri, 22 Aug 2014 12:03:53 GMT
Cache-Control: private, max-age=0
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
Server: GSE
Connection: close
```

```
<?xml version="1.0" encoding="UTF-8"?>
<feed                        gd:etag="&quot;Qn4zcDVSLy17ImA9XRZbF00DRwc.&quot;"
xmlns="http://www.w3.org/2005/Atom"
xmlns:batch="http://schemas.google.com/gdata/batch"
xmlns:gContact="http://schemas.google.com/contact/2008"
xmlns:gd="http://schemas.google.com/g/2005"   xmlns:openSearch="http://a9.com/-
/spec/opensearch/1.1/">
 <id>*****@gmail.com</id>
 <updated>2014-08-22T12:03:53.088Z</updated>
 <category scheme="http://schemas.google.com/g/2005#kind"
term="http://schemas.google.com/contact/2008#contact"/>
 <title>Tuomo Tutkija's Contacts</title>
 <link rel="alternate" type="text/html" href="https://www.google.com/"/>
 <link rel="http://schemas.google.com/g/2005#feed" type="application/atom+xml"
href="https://www.google.com/m8/feeds/contacts/*****%40gmail.com/base2_propert
y-android_linksto-gprofiles_highresphotos"/>
 <link rel="http://schemas.google.com/g/2005#post" type="application/atom+xml"
href="https://www.google.com/m8/feeds/contacts/*****%40gmail.com/base2_propert
y-android_linksto-gprofiles_highresphotos"/>
 <link rel="http://schemas.google.com/g/2005#batch"
type="application/atom+xml"
href="https://www.google.com/m8/feeds/contacts/*****%40gmail.com/base2_propert
y-android_linksto-gprofiles_highresphotos/batch"/>
 <link rel="self" type="application/atom+xml"
href="https://www.google.com/m8/feeds/contacts/*****%40gmail.com/base2_propert
y-android_linksto-gprofiles_highresphotos?max-
results=500&amp;orderby=lastmodified&amp;showdeleted=false&amp;sortorder=ascen
ding"/>
 <author> <name>Tuomo Tutkija</name>
  <email>*****@gmail.com</email> </author>
 <generator version="1.0"
uri="http://www.google.com/m8/feeds">Contacts</generator>
 <openSearch:totalResults>3</openSearch:totalResults>
 <openSearch:startIndex>1</openSearch:startIndex>
 <openSearch:itemsPerPage>500</openSearch:itemsPerPage>
 <entry gd:etag="&quot;Q3szeTVSLi17ImA9Wh9RE00KTgE.&quot;">

<id>http://www.google.com/m8/feeds/contacts/*****%40gmail.com/base/243721788c6
0ec13</id>
  <updated>2014-02-13T14:15:52.581Z</updated>
  <app:edited xmlns:app="http://www.w3.org/2007/app">2014-02-
13T14:15:52.581Z</app:edited>
  <category scheme="http://schemas.google.com/g/2005#kind"
term="http://schemas.google.com/contact/2008#contact"/>
  <title/>
  <link rel="http://schemas.google.com/contacts/2008/rel#photo" type="image/*"
href="https://www.google.com/m8/feeds/photos/media/*****%40gmail.com/243721788
c60ec13"/>
  <link rel="self" type="application/atom+xml"
href="https://www.google.com/m8/feeds/contacts/*****%40gmail.com/base2_propert
y-android_linksto-gprofiles_highresphotos/243721788c60ec13"/>
  <link rel="edit" type="application/atom+xml"
href="https://www.google.com/m8/feeds/contacts/*****%40gmail.com/base2_propert
y-android_linksto-gprofiles_highresphotos/243721788c60ec13"/>
  <gd:email rel="http://schemas.google.com/g/2005#other"
address="foobar@onewaymail.com" primary="true"/>
 </entry>
 <entry gd:etag="&quot;QH87fTVSLS17ImA9XRZbF00ITwY.&quot;">

<id>http://www.google.com/m8/feeds/contacts/****%40gmail.com/base/488b5c940ec1
90e8</id>
  <updated>2014-08-22T10:20:21.105Z</updated>
  <app:edited xmlns:app="http://www.w3.org/2007/app">2014-08-
22T10:20:21.105Z</app:edited>
  <category scheme="http://schemas.google.com/g/2005#kind"
term="http://schemas.google.com/contact/2008#contact"/>
```

```
  <title>Wife</title>
  <link rel="http://schemas.google.com/contacts/2008/rel#photo" type="image/*"
href="https://www.google.com/m8/feeds/photos/media/*****%40gmail.com/488b5c940
ec190e8"/>
  <link rel="self" type="application/atom+xml"
href="https://www.google.com/m8/feeds/contacts/*****%40gmail.com/base2_propert
y-android_linksto-gprofiles_highresphotos/488b5c940ec190e8"/>
  <link rel="edit" type="application/atom+xml"
href="https://www.google.com/m8/feeds/contacts/*****%40gmail.com/base2_propert
y-android_linksto-gprofiles_highresphotos/488b5c940ec190e8"/>
  <gd:name> <gd:fullName>Wife</gd:fullName> <gd:givenName>Wife</gd:givenName>
  </gd:name>
  <gd:email rel="http://schemas.google.com/g/2005#home"
address="wife@example.com"/>
  <gd:phoneNumber rel="http://schemas.google.com/g/2005#mobile" uri="tel:+358-
5-6807456886">56807456886</gd:phoneNumber>
  <gd:structuredPostalAddress rel="http://schemas.google.com/g/2005#home">
   <gd:formattedAddress>SampleStreet 64</gd:formattedAddress>
   <gd:street>SampleStreet 64</gd:street>
  </gd:structuredPostalAddress>
  <gContact:groupMembershipInfo deleted="false"
href="http://www.google.com/m8/feeds/groups/*****%40gmail.com/base/6"/>
  <gContact:groupMembershipInfo deleted="false"
href="http://www.google.com/m8/feeds/groups/*****%40gmail.com/base/e"/>
 </entry>
 <entry gd:etag="&quot;Rnc8eTVSLi17ImA9XRZbF00ITw0.&quot;">

<id>http://www.google.com/m8/feeds/contacts/*****%40gmail.com/base/230fe6838eb
2917c</id>
  <updated>2014-08-22T10:21:37.971Z</updated>
  <app:edited xmlns:app="http://www.w3.org/2007/app">2014-08-
22T10:21:37.971Z</app:edited>
  <category scheme="http://schemas.google.com/g/2005#kind"
term="http://schemas.google.com/contact/2008#contact"/>
  <title>Eve</title>
  <link rel="http://schemas.google.com/contacts/2008/rel#photo" type="image/*"
href="https://www.google.com/m8/feeds/photos/media/*****%40gmail.com/230fe6838
eb2917c"/>
  <link rel="self" type="application/atom+xml"
href="https://www.google.com/m8/feeds/contacts/*****%40gmail.com/base2_propert
y-android_linksto-gprofiles_highresphotos/230fe6838eb2917c"/>
  <link rel="edit" type="application/atom+xml"
href="https://www.google.com/m8/feeds/contacts/*****%40gmail.com/base2_propert
y-android_linksto-gprofiles_highresphotos/230fe6838eb2917c"/>
  <gd:name> <gd:fullName>Eve</gd:fullName><gd:givenName>Eve</gd:givenName>
  </gd:name>
  <gd:email rel="http://schemas.google.com/g/2005#home"
address="xxx@example.com" primary="true"/>
  <gContact:groupMembershipInfo deleted="false"
href="http://www.google.com/m8/feeds/groups/*****%40gmail.com/base/6"/>
  <gContact:groupMembershipInfo deleted="false"
href="http://www.google.com/m8/feeds/groups/*****%40gmail.com/base/114a16f18ce
b4967"/>
 </entry>
</feed>
read:errno=0
~$
```

A.2 Token request and response for the calendar application

```
~$ openssl s_client -connect android.clients.google.com:443
CONNECTED(00000003)
depth=2 C = US, O = GeoTrust Inc., CN = GeoTrust Global CA
verify error:num=20:unable to get local issuer certificate
verify return:0
---
Certificate chain
 0 s:/C=US/ST=California/L=Mountain View/O=Google Inc/CN=*.google.com
   i:/C=US/O=Google Inc/CN=Google Internet Authority G2
 1 s:/C=US/O=Google Inc/CN=Google Internet Authority G2
   i:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
 2 s:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
   i:/C=US/O=Equifax/OU=Equifax Secure Certificate Authority
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIHUzCCBjugAwIBAgIIANgyk4M ...
-----END CERTIFICATE-----
subject=/C=US/ST=California/L=Mountain View/O=Google Inc/CN=*.google.com
issuer=/C=US/O=Google Inc/CN=Google Internet Authority G2
---
No client certificate CA names sent
---
SSL handshake has read 4472 bytes and written 434 bytes
---
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES128-GCM-SHA256
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES128-GCM-SHA256
    Session-ID:
F95A4AC8F3F0B281D2C04071F805127F3390195444F7FCFC2576F5828C731372
    Session-ID-ctx:
    Master-Key:
04B951EDC1B10B81C9AF108A6985729DB5596C3FBA378D96A59495348E2068821B8E1CE4673AEA
66340307F53188CC2A
    Key-Arg   : None
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 100800 (seconds)
    TLS session ticket:
    0000 - f0 4c 8f f3 09 31 d8 9c-1d 39 af b2 b2 91 7f 5d   .L...1...9.....]
    0010 - 6e f6 87 04 cd f6 1e 03-63 90 ae 6a b9 1e bd 01   n.......c..j....
    0020 - 13 cb 62 76 6f 97 b6 35-6e fa 68 1b ed 0f 46 19   ..bvo..5n.h...F.
    0030 - ca 08 ed 15 85 ae 84 5b-6b cf cc 9a ed 4b 9f 3b   .......[k....K.;
    0040 - be f5 3f f2 71 5d 5d ea-fc d2 74 6a ea a8 4a 82   ..?.q]]...tj..J.
    0050 - df 27 10 02 9c c5 be 50-ab f7 53 96 bf fd 46 30   .'.....P..S...F0
    0060 - be 77 2b 5d 8f 8e cb 80-28 ec 62 33 cd e7 33 79   .w+]....(.b3..3y
    0070 - db b3 73 4e f1 ea 6c 77-d6 ad c0 37 e3 82 43 e0   ..sN..lw...7..C.
    0080 - ee b7 a9 6e aa c9 09 05-53 5d 12 20 8f 5a c4 25   ...n....S]. .Z.%
    0090 - 07 44 20 50 27 23 29 c5-e1 37 6c ae 78 0a fc 5a   .D P'#)..7l.x..Z
    00a0 - 5f e4 79 a5                                       _.y.
    Start Time: 1408710551
    Timeout   : 300 (sec)
    Verify return code: 20 (unable to get local issuer certificate)
---
POST /auth HTTP/1.1
device: 3c5a5c3bcd5b2d7f
app: com.google.android.calendar
User-Agent: GoogleAuth/1.4 (m3 JSS15J) (m3 JSS15J)
content-length: 548
content-type: application/x-www-form-urlencoded
Host: android.clients.google.com
Connection: Keep-Alive
Accept-Encoding: identity

device_country=fi&operatorCountry=fi&lang=en_GB&sdk_version=18&google_play_ser
vices_version=5089036&accountType=HOSTED_OR_GOOGLE&system_partition=1&Email=**
**%40gmail.com&has_permission=1&service=oauth2%3Ahttps%3A%2F%2Fwww.googleapis.
com%2Fauth%2Fcalendar&source=android&androidId=3c5a5c3bcd5b2d7f&app=com.google
.android.calendar&client_sig=38918a453d07199354f8b19af05ec6562ced5788&callerPk
g=com.google.android.calendar&callerSig=38918a453d07199354f8b19af05ec6562ced57
88&EncryptedPasswd=oauth2rt_1%2F406mny5jZzr7hLLS8RMwYUX0M0okCkeBLRvA1c1ag3M


HTTP/1.1 200 OK
Content-Type: text/plain; charset=UTF-8
Date: Fri, 22 Aug 2014 12:29:14 GMT
Expires: Fri, 22 Aug 2014 12:29:14 GMT
Cache-Control: private, max-age=0
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
Server: GSE
Transfer-Encoding: chunked

d7
issueAdvice=auto
Auth=ya29.aQAjs_X49pHYY1EAAAB39h3uv9Euls1VFEf1NXLSm8nFnY8GFGXuPQTwjnaHgE_SNtQL
5mV2hyLgsDYH7kZ-uxfs1n-KgcxkO7p9xIN4G2b-
jvJAkpt7YkLzDGPbhxhE5XoxVQoYxnVixXkIVQU
Expiry=1408793692
storeConsentRemotely=0
0

^C
~$
```

A.3 Request and response of all the victim’s calendar events

```
openssl s_client -connect android.clients.google.com:443
CONNECTED(00000003)
depth=2 C = US, O = GeoTrust Inc., CN = GeoTrust Global CA
verify error:num=20:unable to get local issuer certificate
verify return:0
---
Certificate chain
 0 s:/C=US/ST=California/L=Mountain View/O=Google Inc/CN=*.google.com
   i:/C=US/O=Google Inc/CN=Google Internet Authority G2
 1 s:/C=US/O=Google Inc/CN=Google Internet Authority G2
   i:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
 2 s:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
   i:/C=US/O=Equifax/OU=Equifax Secure Certificate Authority
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIHUzCCBjugAwIBAgIIANg ...
-----END CERTIFICATE-----
subject=/C=US/ST=California/L=Mountain View/O=Google Inc/CN=*.google.com
issuer=/C=US/O=Google Inc/CN=Google Internet Authority G2
---
No client certificate CA names sent
---
SSL handshake has read 4472 bytes and written 434 bytes
---
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES128-GCM-SHA256
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES128-GCM-SHA256
    Session-ID:
9630BBF1B941DA42FB6C108E5B3E48AE32A765FFC116BE94C654AC19E286F278
    Session-ID-ctx:
    Master-Key:
DED47B7FF70F0E514BEE20DEB0F0194D2C61F7922203E8CCCD654DEAEF91D1ACC878868BCD4BC7
CE90E88001C88F1F7B
    Key-Arg   : None
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 100800 (seconds)
    TLS session ticket:
    0000 - f0 4c 8f f3 09 31 d8 9c-1d 39 af b2 b2 91 7f 5d   .L...1...9.....]
    0010 - d2 93 c6 68 4e 31 34 5b-10 90 65 66 a4 ae 46 0c   ...hN14[..ef..F.
    0020 - a7 8a 77 a8 e7 7d c6 ca-fb cc 08 25 08 ac 08 b6   ..w..}.....%....
    0030 - 4f d6 11 9c 13 bc de c6-92 53 5b 5e f3 71 c5 ca   O........S[^.q..
    0040 - 5a 85 bc 48 69 68 4c 9b-de 46 f1 2c 21 3a f8 1e   Z..HihL..F.,!:..
    0050 - 00 3d 6c 3f ae 36 ce 51-1d 00 08 ae 32 1b 71 1f   .=l?.6.Q....2.q.
    0060 - 91 f2 3b 0f ea c6 1f 47-f7 25 ac 1f 72 7e 28 41   ..;....G.%..r~(A
    0070 - 56 e0 6d 5e 54 90 59 8e-cf 7c 80 3f 9a 43 da ff   V.m^T.Y..|.?.C..
    0080 - e6 40 0d 1b e6 35 db 4b-af 28 a8 ea 59 fd 43 62   .@...5.K.(..Y.Cb
    0090 - 0a 54 f5 04 b8 86 ab b3-d8 db cb 80 23 b3 0f 90   .T..........#...
    00a0 - 03 32 75 4d                                       .2uM

    Start Time: 1408711220
    Timeout   : 300 (sec)
    Verify return code: 20 (unable to get local issuer certificate)
---
GET
/calendar/v3/calendars/****@gmail.com/events?maxAttendees=50&maxResults=200&ti
meMax=2015-09-02T00:00:00.000Z HTTP/1.1
Accept-Encoding: identity
```

```
Authorization:                                                     OAuth
ya29.aQAjs_X49pHYY1EAAAB39h3uv9Euls1VFEf1NXLSm8nFnY8GFGXuPQTwjnaHgE_SNtQL5mV2h
yLgsDYH7kZ-uxfs1n-KgcxkO7p9xIN4G2b-jvJAkpt7YkLzDGPbhxhE5XoxVQoYxnVixXkIVQU
User-Agent:             samsung/m3xx/m3:4.3/JSS15J/I9305XXUEMKC:user/release-
keys:com.google.android.calendar:201404014  Google-HTTP-Java-Client/1.14.1-beta
(gzip)
Host: www.googleapis.com
Connection: close

HTTP/1.1 200 OK
Expires: Fri, 22 Aug 2014 12:40:24 GMT
Date: Fri, 22 Aug 2014 12:40:24 GMT
Cache-Control: private, max-age=0, must-revalidate, no-transform
Content-Type: application/json; charset=UTF-8
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
Content-Length: 1144
Server: GSE
Alternate-Protocol: 443:quic
Connection: close
```

```
{
 "kind": "calendar#events",
 "etag": "\"1408693865712000\"",
 "summary": "*****@gmail.com",
 "updated": "2014-08-22T07:51:05.712Z",
 "timeZone": "Europe/Helsinki",
 "accessRole": "owner",
 "nextSyncToken": "CICT2suxpsACEICT2suxpsACGAU=",
 "items": [
  {
   "kind": "calendar#event",
   "etag": "\"2817387731424000\"",
   "id": "fjhb3rflgagu9s6rdadfmustd8",
   "status": "confirmed",
   "htmlLink":
"https://www.google.com/calendar/event?eid=ZmpoYjNyZmxnYWd1OXM2cmRhZGZtdXN0ZDg
gbmlzZWMudHV0QG0",
   "created": "2014-08-22T07:51:05.000Z",
   "updated": "2014-08-22T07:51:05.712Z",
   "summary": "2 week vacation with wife in Hong Kong",
   "location": "Hong Kong",
   "creator": {
    "email": "*****@gmail.com",
    "displayName": "Tuomo Tutkija",
    "self": true
   },
   "organizer": {
    "email": "*****@gmail.com",
    "displayName": "Tuomo Tutkija",
    "self": true
   },
   "start": {
    "date": "2014-09-01"
   },
   "end": {
    "date": "2014-09-02"
   },
   "transparency": "transparent",
   "iCalUID": "fjhb3rflgagu9s6rdadfmustd8@google.com",
   "sequence": 0,
   "reminders": {
    "useDefault": false
   }
  }
 ]
}
read:errno=0
~$
```