Heikki Taavettila

OVER-THE-AIR COPY PROTECTION

Using commonly used techniques
OVER-THE-AIR COPY PROTECTION

Using commonly used techniques

Heikki Taavettila
Master’s thesis
Spring year 2015
Degree Programme in Information Technology
Oulu University of Applied Sciences
ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology

Author: Heikki Taavettila
Title of thesis: Over-the-Air Copy Protection
Supervisor: Teemu Korpela
Term and year of completion: Spring 2015
Number of pages: 81

This Master’s thesis examines how copy prevention and market segmentation via licensing could utilize the fact that virtually all mobile applications have connection to the Internet at least occasionally. It surveys existing technical tools and methods for an author of a software product to be able to provision features on units at another edge of the world. The thesis is about a project to replace an old copy prevention and system which was based on the installing a software license using an installation package.

The thesis will examine basic building blocks for secure communication. It will review modern cryptography techniques and how a public key cryptography enables two parties without pre-existing acquaintance to be able to agree a common secret key using an unsecure channel. Also, it studies an RSA asymmetric cryptography and a symmetric Rijndael cryptographic algorithm behind the AES standard. The thesis will discuss how cryptography is used together with Internet communication protocols establishing secure sockets over unsecure channels. The thesis will drill down to practical considerations of how a key token, a license file, can be used in product segmentation as well as unauthorized copy prevention. It discusses practical protection methods against attempts to work around copy prevention from network, server and mobile application points of view. This thesis will also examine the possibilities of implementing a server from different levels of cloud computing to having servers in own premises.

This thesis will offer a summary of available tools for designing any client/server solution with a demand for a high availability and secure communication. Naturally, there is room for a further development such as Elliptic Curve Cryptography and how it could be utilized in all relevant platforms. In addition, dockers as a technology is becoming increasingly mature and it might allow interesting opportunities. Also, business models like in-app-purchases are not covered at all.

Keywords: copy protection, cloud computing, cryptography, rsa, aes
PREFACE

This thesis was supposed to be a project report of implementing an Over-The-Air copy prevention, license distribution and management system for a smartphone application I designed and implemented in Oulu, Finland during spring 2013. However, the original plan was overcome by events. After the project was completed, I moved to Virginia, USA with my family into a completely new role. Before I had personal resources to allocate to this thesis, we moved again, this time to Texas.

Now we have settled down here and I was able to dedicate enough time for this thesis. The server application has been replaced by its successor and the product portfolio of Amazon Web Services, Google and has been changed a lot. Instead of writing about obsolete approaches taken in history, I will focus on what should be taken into consideration if I would do a similar project today.

I’m in gratitude for my family and friends supporting me in finishing this work. Today, it’s been a full week since I was home awake at the same time with my daughter. Finishing this thesis was actually a huge effort for me but it may have been even greater effort for my wife as in addition to putting up with a tired, cranky and absent version of me, she has kept things rolling.

I want to thank Richard Milam for words of wisdom: “You just need to be ruthlessly efficient. You got time for TV when it’s done.” And I want to thank Sauli Möttinen and Timo Kumpumäki for encouragements: “Get it done. You’ll be sorry if you don’t”.

Also I want to thank Roy Rivera for his highly appreciated viewpoints.

And last but not least I want to thank Teemu Korpela for all the support, advices and deep knowledge of security.

Texas, USA, 19.04.2015
Heikki Taavettila
TABLE OF CONTENTS

ABSTRACT .................................................................................................................................. 3
PREFACE ........................................................................................................................................ 4
TABLE OF CONTENTS .................................................................................................................. 5
TERMS AND ABBREVIATIONS ....................................................................................................... 7
1 INTRODUCTION .......................................................................................................................... 10
2 INTELLECTUAL PROPERTY INFRINGEMENT ............................................................................. 12
   2.1 Entertainment industry against little girls .................................................................................. 12
   2.2 Big boys against pirates ............................................................................................................ 13
3 SECURE COMMUNICATION ........................................................................................................ 16
   3.1 Modern Cryptography .............................................................................................................. 17
      3.1.1 Public key exchange ........................................................................................................ 19
      3.1.2 Prime numbers and primitive roots ................................................................................ 20
      3.1.3 RSA ............................................................................................................................... 21
      3.1.4 AES ............................................................................................................................... 23
   3.2 Security over unsecure channel .............................................................................................. 26
      3.2.1 Secure Shell .................................................................................................................... 28
      3.2.2 HTTP Over TLS ............................................................................................................ 30
   3.3 SECURITY MAINTENANCE ..................................................................................................... 33
4 PRACTICAL CONSIDERATIONS ............................................................................................... 36
   4.1 Copy protection techniques ..................................................................................................... 37
      4.1.1 Key token ........................................................................................................................ 37
      4.1.2 Phone home .................................................................................................................... 39
      4.1.3 Segmentation .................................................................................................................. 42
4.1.4 What can be trusted ................................................................. 43
4.2 Case study: Reverse Engineering Android APK ................................. 47
4.3 The Server .................................................................................. 52
  4.3.1 Platform as a Service ................................................................. 54
  4.3.2 Infrastructure as a Service .......................................................... 59
  4.3.3 On-premises ......................................................................... 67
  4.3.4 Containers ............................................................................ 70

5 CONCLUSIONS .............................................................................. 73
REFERENCES .................................................................................... 75
### TERMS AND ABBREVIATIONS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>Advanced Encryption Standard. Encryption standard defined by NIST.</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface. API defines an abstract programming interface.</td>
</tr>
<tr>
<td>APK</td>
<td>Android Application Package. Installation package for Android applications.</td>
</tr>
<tr>
<td>ASCII</td>
<td>American Standard Code for Information Interchange. ASCII is a one 8 bit character.</td>
</tr>
<tr>
<td>AWS</td>
<td>Amazon Web Services.</td>
</tr>
<tr>
<td>Base64</td>
<td>Base64 is binary-to-text encoding scheme.</td>
</tr>
<tr>
<td>Bash</td>
<td>Shell for unix/linux.</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit. Commonly known as processor.</td>
</tr>
<tr>
<td>DES</td>
<td>Data Encryption Standard. Deprecated encryption standard defined by NIST.</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System. System that translates hostnames into IP addresses.</td>
</tr>
<tr>
<td>ECC</td>
<td>Elliptic Curve Cryptography.</td>
</tr>
<tr>
<td>ESN</td>
<td>ESN. Electric serial number.</td>
</tr>
<tr>
<td>FAQ</td>
<td>Frequently Asked Questions.</td>
</tr>
<tr>
<td>GAE</td>
<td>Google App Engine. SaaS cloud provider by Google.</td>
</tr>
<tr>
<td>GNU</td>
<td>GNU’s Not Unix. Unix like operating system.</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System. Satellite based positioning system.</td>
</tr>
<tr>
<td>GSMA</td>
<td>GSM Association.</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language. Markup language designed for multimedia content.</td>
</tr>
<tr>
<td>HTTP</td>
<td>HyperText Transfer Protocol. Data transfer protocol.</td>
</tr>
<tr>
<td>HTTPS</td>
<td>HTTP over TLS. Secure transfer protocol.</td>
</tr>
<tr>
<td>IAM</td>
<td>AWS Identity and Access Management.</td>
</tr>
<tr>
<td>Acronym</td>
<td>Definition</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>IMEI</td>
<td>International Mobile Station Equipment Identity. Similar to ESN.</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol. OSI layer 3 communication protocol.</td>
</tr>
<tr>
<td>JAR</td>
<td>Java Archive. Package file for Java application.</td>
</tr>
<tr>
<td>JNI</td>
<td>Java Native Interface. Interface for accessing native methods from Java.</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network. A local computer network.</td>
</tr>
<tr>
<td>MAC</td>
<td>Media Access Control. MAC address is a layer 2 address for interface.</td>
</tr>
<tr>
<td>MSDN</td>
<td>Microsoft Developer Network.</td>
</tr>
<tr>
<td>NIST</td>
<td>National Institute of Standards and Technology. US standardization organization.</td>
</tr>
<tr>
<td>NTP</td>
<td>Network Time Protocol.</td>
</tr>
<tr>
<td>OpenSSL</td>
<td>Open source implementation for SSL and TSL protocols.</td>
</tr>
<tr>
<td>OTA</td>
<td>Over-the-Air.</td>
</tr>
<tr>
<td>P2P</td>
<td>Peer-to-Peer.</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer.</td>
</tr>
<tr>
<td>PGP</td>
<td>Pretty Good Privacy. Cryptosystem.</td>
</tr>
<tr>
<td>ROT13</td>
<td>Rotate by 13 places. Similar weak encryption than Caesar cipher.</td>
</tr>
<tr>
<td>RSA</td>
<td>Public key cryptosystem.</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit.</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language. Language for database queries.</td>
</tr>
<tr>
<td>SSH</td>
<td>Secure Shell. Encrypted network protocol text based shell session.</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol. OSI layer 4 communication protocol.</td>
</tr>
<tr>
<td>TIM</td>
<td>Trustworthy Internet Movement.</td>
</tr>
<tr>
<td>TLS</td>
<td>Transport Layer Security. Provides secure connection that for example HTTPS.</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>UPS</td>
<td>Uninterrupted Power Supply. Provides battery backup for electric devices.</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator. Usually Internet address including protocol.</td>
</tr>
<tr>
<td>WIFI</td>
<td>Wireless LAN technology.</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine.</td>
</tr>
<tr>
<td>VMM</td>
<td>Virtual Machine Manager.</td>
</tr>
<tr>
<td>VPC</td>
<td>Virtual Private Cloud. Cloud version of Virtual Private Network.</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

There’s no business like software business. Write it once, sell it unlimited times. Back in a day selling a copy of software product involved a marginal material and labor cost. Software products where stored on physical media like a diskette, CD or DVD, which was delivered to a buyer. Today, digital distribution, which virtually removes all material costs, is an increasingly common method for selling software products. In fact, I haven’t bought a computer with an optical media drive since 2011.

An ability to sell a product without any manufacturing costs doesn’t come free. The ability to copy data is not limited to copyright holders and there are a lot of people around the word utilizing that ability. A co-founder of Microsoft, Bill Gates, wrote a famous open letter to hobbyists where he accused Altair hobby computer users for instead of buying Altair Basic, copying without payment and therefore stealing it (Gates, 1976). Bill Gates argued that software should not be considered free and something that could be shared whereas only hardware would be paid. Richard Stallman, a founder of a free software movement and GNU project, disagreed and he thought that software should not be only free of charge; also, source code should be freely available to everybody (Stallman, 2015).

Since those days the interest groups for proprietary software have been active to stop all software that users can use free of charge and associate Open Source Software to piracy (Johnson, 2010). For business users Microsoft argued that Open Source Software violates its patents and for its users are in risk for lawsuits (Parloff, 2007). Steve Ballmer, the former CEO of Microsoft, described Open Source as communism and that “Linux is a cancer that attaches itself in an intellectual property sense to everything it touches” (Green, 2001) (Graham, 2000).

Today software industry has done a bit of a soul searching and found that it can actually benefit from Open Source Software. Microsoft actually contributes to Open Source projects and offers cloud services for free of charge (Metz, 2012). However, illegal use
of Microsoft products persists. During the twenty years that I have been using Microsoft’s products, they have come up with technologies to prevent illegal copies but effectively caused annoyance for those users who have had a purchased copy of their product and who didn’t have a copy protection bypassed.

Having said this I don’t aim to prevent an illicit use of our product. I merely attempt to ensure that our legit customers have a smooth user experience and perhaps a bit of a challenge for those whose legitimacy is limited.

I will begin looking into what I’m up against to and how others have addressed the question of intellectual property infringement. In the next chapter I will discuss tools available with respect to secure communication. And finally, I will address the question how to grant usage on legit users in far away lands.
2 INTELLECTUAL PROPERTY INFRINGEMENT

Some Vikings had a superior sword called Ulfberht. Its metallurgy was far better than other swords during those times and these swords had a signature in the blade; a trademark. However, it was observed that some of these swords had a fault in signature. Also, the swords that had faulty signatures had metallurgy that was like any other during that era. Ulfberht was a victim of intellectual property violation. Somebody had forged a forged Ulfberht. (Stalsberg, 2015)

In addition to a possible loss of revenue to a genuine Ulfberht-smith low quality copies may have tainted an image of quality for genuine ones. Ulfberhts were signed and even though forgeries had invalid signature Vikings lacked method to validate it. Forging a sword requires a skill that not all possess. Even a child can copy a chunk of bits from media to media and that’s what software, music and videos are; a chunk of bits.

2.1 Entertainment industry against little girls

Back in a day when music was on analogic media, making copies always resulted in an inferior sound quality. CD albums didn’t really change the setup as unauthorized copies tended not to work always as the reliability of first generations of CD burners was poor. Purchasing an official copy of CD assured the superior quality of a product.

However entertainment industry became increasingly worried about people making copies at home and they began embedding a copy protection to music albums. As result of this, consumers were not able to use their official copy in computers and some car stereos had problems playing copy protected CD-like products. With regard to being appropriate Sony Entertainment crossed the line with a worldwide scandal. Their copy protection was in fact a malware that violated consumer’s privacy in addition to exposing their computer to other malware. (Nykänen, 2003) (Schneier, 2005)

At this point, industry turned the tables by making pirate music of higher quality than official copies had. When Napster introduced an easy and fast online distribution for
music instead of offering a legit service for purchasing music online, the music industry began legal actions randomly pursuing individuals, boys and girls downloading music (Teosto, 2012). After music industry began offering a legit music online, it began to look like p2p users actually would buy more than non-users (Karaganis & Renkema, 2015).

**FIGURE 1.** Music collections collection p2p users vs non-users (Karaganis & Renkema, 2015).

My interpretation is that the lesson to take home here is the following: “In your effort to prevent an unauthorized usage of your product, don’t jump on toes of a paying customer while insulting him/her verbally. Instead, pursue a smooth and convenient user experience for the main source of your income”. My interpretation could be wrong but I’ll go with it anyway.

### 2.2 Big boys against pirates

So what is the extent of software piracy? The Software Alliance (BSA) does an annual survey which seems to be the number refered to in most sources I’ve seen. According to BSA, the value of unlicenced (pirate) software on planet earth is $62.7 billion (BSA,
According to Business Action to Stop Counterfeiting and Piracy (BASCAP) group, digital piracy cost the EU more than 20 billion euros between 2008 and 2011 and creative industries would be expected to see revenue losses up to 240 billion euros between 2010 and 2015 resulting up to 1.2 million job losses (Baker, 2014) (TERA Consultants, 2014). The Economist magazine has accused BSA for inflating its figures to suit its political aims (The Economist, 2005) (The Economist, 2012). The Economist isn’t completely alone with that thought as preventing software piracy is not easy while trying to respect the freedom of speech and right to privacy and therefore justification needs to be considerable (McCullagh, 2002) (Wikipedia, 2015).

As legislation has given copyright holders better tools to inform copyright violators to law enforcement agencies, copyright violators have started to use encrypted tunnels for preventing monitoring of their Internet usage. BBC Worldwide considers it to be reasonable that ISP’s would be obligated to identify and take action for a suspicious behavior such as high data volumes and use of IP obfuscation tools (BBC Worldwide, 2014). At this point I want to note that copyright violators use the same technology to hide their Internet traffic from copyright holders than business and government use for secure communication.

China has reported to have arrested 60,000 people for a copyright infringement in 2013 (Muncaster, 2014). Meanwhile, the Office of the United States Trade Representative concluded in its annual The Special 301 Report that China’s Government has reported to complete legalization at a central and provincial level. However, US software companies have seen only modest increase in sales to China’s Government. In addition, it seems that Chinese companies are stealing IP’s under government protection: “Particularly troubling are public reports by independent security firms that actors affiliated with the Chinese military and Chinese Government have systematically infiltrated the computer systems of a significant number of U.S. companies and stolen hundreds of terabytes of data, including IP, from these companies.” (Marantis, 2013)

The Russia’s Government has been accused of using copyright infringement as a scapegoat suppressing critics of the current regime (Levy, 2010). The Special 301
Report notes that even though online piracy is growing in Russia, the number of criminal raids has decreased (Marantis, 2013). I can’t avoid wondering if Russia has run out of opposition or opposition has run out of computers. Also, I just wonder who were those 60,000 arrested for piracy in China.

I believe that the take home lesson here is the following: “Preventing an unauthorized usage is likely to be a too big bite to chew. Instead try to provide a bit of a challenge for pirates and focus on a smooth and convenient user experience for the main source of your income”.
3 SECURE COMMUNICATION

An ability to communicate securely is crucial when attempting to prevent an unauthorized usage. I feel gratitude for smart individuals who have developed standard techniques that enable a secure communication without requiring me as a developer to re-invent a wheel, which wouldn’t necessarily be completely round.

In the previous chapter we took a peak to history 1,200 years back. This time we’ll stick to more recent events. Mary, Queen of Scots, was convicted of high treason and beheaded on February 8, 1587. For her misfortune, Diffie-Hellman Key Exchange and AES were not available those days. Mary had the impression that she would have a secure communication with her allies by exchanging encrypted messages in beer kegs. Thomas Phelippes, a cryptanalyst, was cryptanalyzing her messages and even added his own content for getting Mary’s allies to reveal more information than Mary had originally requested. (Kahn, 1973)

Mary’s communication was under man-in-the-middle attack. Her messages were not private anymore and they were corrupted. A failure to ensure confidentiality, data integrity and authentication caused her a head. The world might look a bit different if Mary’s conspiracy had not been revealed. Moreover, what if cryptanalysts had not been able to reveal the contents of German messages encrypted with Enigma.

When a plaintext is processed through cryptosystem, it is encrypted into a unreadable ciphertext. On opposite direction a cipher text is processed through a cryptosystem and it is decrypted back to a plaintext. In many examples Alice and Bob are sending messages to each other. However, evil Eve is eavesdropping and tries to find out the content of Alice’s and Bob’s messages. Eve is a cryptanalyst trying to break ciphered messages, also called cryptograms. (Kahn, 1973)
Caesar used to write encrypted messages by replacing a letter with a letter standing three places further down the alphabet. A couple of years ago in Finland a murderer ended up giving police discriminating evidence by leaving a ROT13 encrypted letter to his brother who was also accomplish. ROT13 is almost identical to Caesar and from a cryptography point of view it’s ridiculous. (Iltalehti, 2012)

3.1 Modern Cryptography

Instead of secret methods modern cryptography relies on secret keys with commonly known methods.
Modern cryptography falls into two major categories. *Symmetric cryptography* uses same secret key to encrypt a plaintext to a ciphertext which then decrypts a ciphertext back to a plaintext.

*Asymmetric cryptography*, also known as *public key cryptography*, uses key pairs; a secret *private key* and a *public key*. A public key can be used to encrypt messages that can be decrypt only with a private key. Also, a private key can be used to add *digital signature* messages which can be verified by public key to verify data integrity and authority.

*Message Digest* summarizes the contents of a message with a few bytes long *hash*. A hash function is a one-way function which is easy in one way and hard or infeasible in the other way. It is infeasible to find a message that created hash and it is infeasible to find two messages that would an produce identical hash. (NIST, 2012)

Message digest is often used to store passwords or verify data integrity. When passwords are stored as message digests or hashes, credentials can be verified without storing actual passwords. Hash is also often used to verify data integrity for files shared in the Internet. Message Digest may be used to verify that data has not been changed after hash has been calculated but it does nothing to verify who calculated the hash checksum. MD5, SHA1, SHA-256, SHA-512 are commonly used hash functions.

### Eclipse downloads - mirror selection

All downloads are provided under the terms and conditions of the Eclipse Foundation Software User Agreement unless otherwise specified.

**Download** eclipse-java-luna-5R2-win32-x86_64.zip from:

[United States] Virginia Tech (http)

```
Checksums: [MD5] [SHA1] [SHA-512]
```

...or pick a mirror site below.

*FIGURE 2. List of Hash checksums for Eclipse download.*
There are interesting cryptographic solutions like Quantum Key Distribution where man-in-the-middle attack would be revealed by the physical laws of photons. Also, Pretty Good Privacy (PGP) and Elliptic Curve cryptography seem very interesting. However, I will stick to the ones that are best supported by most operating systems and platforms as default. Therefore, I will take a bit deeper dive to a public key cryptography, RSA and an AES Rijndael algorithm.

3.1.1 Public key exchange

Before Diffie-Hellman public key exchange, secure communication required that a commonly used secret key was agreed before secure communication could be initiated. A public key exchange gave an answer to a question: “How two people with no previous acquaintance agree on a secret key”. Public key exchange enables agreeing on a secret key over an unsecure channel. (Diffie & Hellman, 1976)

Whitfield Diffie and Martin Hellman came up with the idea of using mathematical one-way functions that would be to easy calculate one way but it would be unfeasible to reverse the calculation even though the result and most ingredients used in the calculation would be commonly known.

Bob and Alice will both share their public key (Y) openly, in addition $a$ and $q$ are shared openly.

\[ Y = \alpha^X \mod q, \text{ for } 1 \leq X \leq q - 1 \]

\( Y = \text{public key} \)

\( X = \text{private key} \)

\( \alpha = \text{base (primitive root of } q) \)

\( q = \text{modulo (prime number)} \)

The received Y from other party is combined with own secret key in a same way that Y was calculated and both parties end up with the same secret key.

19
\[ K_{ab} = \alpha^x_a x_b \mod q = Y_a^x_b \mod q = Y_b^x_a \mod q \]

\[ K_{ab} = \text{Secret key} \]

\[ X_a = \text{Alice's private key} \]

\[ X_b = \text{Bob's private key} \]

TABLE 1 is an example of how Alice and Bob agree to use the number 16 as a secret key without saying it directly. At the beginning \( \alpha \) and \( q \) are agreed publicly (\( \alpha = 11 \) \( q = 29 \)). Alice has a private key \( X_a = 7 \) and Bob has a private key \( X_b = 12 \).

**TABLE 1. Diffie-Hellman public key exchange in values.**

<table>
<thead>
<tr>
<th>Alice Variable</th>
<th>Value</th>
<th>Bob Variable</th>
<th>Value</th>
<th>Bob Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X_a )</td>
<td>3</td>
<td>( X_b )</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>( Y_a = 11^x_a \mod 29 )</td>
<td>26</td>
<td></td>
<td>23</td>
<td>( Y_b = 11^x_b \mod 29 )</td>
</tr>
<tr>
<td>( Y_{ab} = 23^{12} \mod 29 )</td>
<td>16</td>
<td></td>
<td>16</td>
<td>( Y_{ab} = 26^{12} \mod 29 )</td>
</tr>
</tbody>
</table>

3.1.2 **Prime numbers and primitive roots**

In the previous example all values between 1 and 28 (\( q - 1 \)) would have been equally likely. The crucial requirement is that \( q \) is a prime number and \( \alpha \) is a primary root. I was going to illustrate this with Excel but \( 11^x \) results become bigger than Excel can handle and therefore I did the calculation with Java. Notice how every allowed \( X \) value (\( 1 \leq X \leq q - 1 \)) results a unique \( Y \) value but there isn’t a recognizable pattern that could be used. Except that there is; when \( X = 1 \) and \( X = q - 1 \) values are always \( \alpha \) and 1.
Evil Eve would not need a super computer to try enough combinations to figure out Alice’s and Bob’s private and therefore secret key for mod 29. However, if q is a big number closer to $10^{300}$, it takes much longer. Just to give an idea how long it would take to go through all values between 1 and 10,300 assuming that one CPU could check 8 keys in one nanosecond. Adding more CPU’s gets the job done faster but it is still a long wait even if a correct key would be found before going through 1% of possible keys.

<table>
<thead>
<tr>
<th>Possible keys</th>
<th>$1E+300$</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpu’s seconds</td>
<td>1,3E+290</td>
</tr>
<tr>
<td></td>
<td>1,3E+289</td>
</tr>
<tr>
<td></td>
<td>1,3E+288</td>
</tr>
<tr>
<td></td>
<td>1,3E+287</td>
</tr>
<tr>
<td></td>
<td>1,3E+286</td>
</tr>
<tr>
<td></td>
<td>1,3E+285</td>
</tr>
<tr>
<td></td>
<td>1,3E+284</td>
</tr>
<tr>
<td></td>
<td>1,3E+283</td>
</tr>
<tr>
<td></td>
<td>1,3E+282</td>
</tr>
<tr>
<td></td>
<td>1,3E+281</td>
</tr>
<tr>
<td>days</td>
<td>1,4E+285</td>
</tr>
<tr>
<td></td>
<td>1,4E+284</td>
</tr>
<tr>
<td></td>
<td>1,4E+283</td>
</tr>
<tr>
<td></td>
<td>1,4E+282</td>
</tr>
<tr>
<td></td>
<td>1,4E+281</td>
</tr>
<tr>
<td></td>
<td>1,4E+280</td>
</tr>
<tr>
<td></td>
<td>1,4E+279</td>
</tr>
<tr>
<td>years</td>
<td>5,3E+287</td>
</tr>
<tr>
<td></td>
<td>5,3E+286</td>
</tr>
<tr>
<td></td>
<td>5,3E+285</td>
</tr>
<tr>
<td></td>
<td>5,3E+284</td>
</tr>
<tr>
<td></td>
<td>5,3E+283</td>
</tr>
<tr>
<td></td>
<td>5,3E+282</td>
</tr>
<tr>
<td></td>
<td>5,3E+281</td>
</tr>
</tbody>
</table>

3.1.3 RSA

In addition to a public key exchange, Diffie and Hellman came up with a whole concept of public key cryptography with digital signatures and a trap-door function but they did not introduce a practical implementation of that concept (Diffie & Hellman, 1976). Ron

RSA’s version of one-way function is multiplying two large prime numbers. Factoring large numbers is much harder than multiplying them even for computers and when multiplied numbers are big enough factoring them becomes unfeasible. RSA uses Euler totient function for $\phi(p)$ where $p$ is a prime number. (Rivest, Shamir & Adleman, 1978)

RSA public key has two parts; the RSA modulus ($n$) and the RSA public exponent ($e$). $n$ is a multiplication of two large primes. RSA private key has also two parts; $n$ and the RSA private exponent ($d$). $d$ is a large random integer that is relatively prime with $\phi(n)$. Relatively prime means that $d$ and $\phi(n)$ do not have a greater common divisor than 1. $e$ can be calculated from an equation below. (Rivest, Shamir & Adleman, 1978)

$$e \cdot d \equiv 1 \pmod{(p-1) \cdot (q-1)}$$ \hspace{1cm} \text{FORMULA 3}

A plaintext message ($m$) is encrypted into a ciphertext ($c$) using $n$ and $e$ parts from a public key. $c$ is then decrypted back to $m$ by using $d$ and $n$ parts from a private key. (RSA Laboratories, 2012)

Encryption $$c = m^e \mod n$$ \hspace{1cm} \text{Decryption} $$m = c^d \mod n$$ \hspace{1cm} \text{FORMULA 4}

RSA digital signature ($s$) works in the same way that decryption with a deviation that private $d$ is used for message encryption and $e$ for decryption. With a signature signed message isn’t usually encrypted instead a message digest, also known as hash is encrypted. The output of hash function is then compared. (RSA Laboratories, 2012) (NIST, 2012)

Encryption $$s = m^d \mod n.$$ \hspace{1cm} \text{Decryption} $$m = s^e \mod n$$ \hspace{1cm} \text{FORMULA 5}
Recommended length for n mentioned in the original paper published in 70’s is 200 decimal digits which has a corresponding bit length of 662 bits (Rivest, Shamir, & Adleman, 1978). Today practical key lengths are between 1024 – 4096 bits. Also the current RSA Cryptography Standard allows more than 2 prime factors for making the calculation more feasible for less efficient hardware. The maximum length of one RSA ciphertext is n – 1. Random padding defined by a standard needs to be added if message is shorter than n – 1. (RSA Laboratories, 2012).

I did some experimenting using Java API’s and noticed that there’s a quite big variation between what time it takes to generate a key pair. Creating a 4096 bit key pair took between 560 ms and 13,694 ms an average being 3,689 ms for 100 key pairs. It is my understanding that big differences are due to the fact that finding large prime numbers may take some time and a mere prime number check would be time consuming if it would not be optimized. In addition, d value needs to be factored for verifying that it is relatively prime with φ(n). Even though generating an RSA key pair may be CPU intensive, using a created key pair is not.

<table>
<thead>
<tr>
<th>Key length</th>
<th>Minimum ms</th>
<th>Average (ms)</th>
<th>Maximum (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>13</td>
<td>43</td>
<td>166</td>
</tr>
<tr>
<td>2048</td>
<td>41</td>
<td>296</td>
<td>1490</td>
</tr>
<tr>
<td>4096</td>
<td>560</td>
<td>3689</td>
<td>13694</td>
</tr>
</tbody>
</table>

3.1.4 AES

In November 2001 US National Institute of Standards and Technology (NIST) Announced Rijndael cryptography algorithm as Advanced Encryption Standard (AES) superseding old Data Encryption Standard (DES). Rijndael is a symmetric block cipher designed by Vincent Rijmen and Joan Daemen. Rijndael encrypts 128 bit plaintext
blocks into 128 bit ciphertext using 128, 192 or 256 bit keys. Algorithm itself supports other block sizes but they are not adopted by AES. (NIST, 2001)

Rijndael was born in a totally different world than public key cryptography. When Diffie and Hellman came up with a public key exchange, they invented something that had not been thought before. In addition, they came up with a concept of asymmetric cryptography. Rivest, Adi and Adlerman invented an implementation for an asymmetric cryptography concept. Rijmen and Daemen used the existing cryptographic primitives for building a robust cryptosystem and then participated in a competition for the next AES. Even though they didn’t invent anything new per se, they used the existing primitives better than for example IBM and Ron Rivest.

AES takes 4 plaintext words and a 4-, 6-, 8-words key as input and processes it from 10 to 14 rounds depending on a key length. A word in this context is 32 bits or 8 bytes. A key is a random number; any random number. (NIST, 2001)

<table>
<thead>
<tr>
<th>Key Length (Nk words)</th>
<th>Block Size (Nb words)</th>
<th>Number of Rounds (Nr)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-128</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>AES-192</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>AES-256</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>

*FIGURE 4. Key-Block-Round combinations (NIST, 2001).*

While a plaintext data is undergoing encryption process it is called state. Bytes are organized logically in 4 x 4 tables.

*FIGURE 5. Input, state and output tables (arrays) (NIST, 2001).*
Encryption starts with *AddRoundKey* which is an Exclusive OR (XOR) operation between state and key. The first step in a loop is *SubBytes* transformation where bytes are changing places according to a substitution table (S-box). *ShiftRows* transformation shifts bytes within rows and *MixColumns* transformation scrambles contents of each column by multiplications. A state array is looped through these functions from 10 to 14 times and encryption is done.

![Pseudo code for Rijndael cipher](binaryimage)

*FIGURE 6. Pseudo code for Rijndael cipher (NIST, 2001)*

I did some testing between RSA and AES to compare an output of two almost identical messages. I encrypted 'My Secret Message' twice to see what is the difference between AES which does not use pseudorandom padding, and RSA which uses padding. In addition, I encrypted a third message with a 2-bit difference in one byte (M -> N). Two identical plaintext messages are identical as a ciphertext also with AES. Also, only 8 MSB bytes have changed when a message with two 2-bit differences was encrypted. RSA has that pseudorandom padding included so all ciphertexts seem unique.

RSA ciphertext is 1024 bits which is also the length of the n value in a public key (and a private key). It needs to be recognized that RSA and AES key lengths cannot be
compared. A secret part of RSA private key is the prime number factor of n which could be calculated from a public key if eternity would be the deadline. AES secret key is a 128-bit long pseudorandom number, which could be anything between 0 and 340,282,366,920,938,463,463,374,607,431,768,211,456.

FIGURE 7. AES and RSA encryption output test.

3.2 Security over unsecure channel

The Internet; everything is connected to the Internet. So are the devices running our applications and we will use it to communicate with applications in the scope of this project. Therefore, a few topics of the Internet need to be addressed as a foundation for discussion in the next chapter.

Virtually, all traffic over the Internet travels in IP packages (Internet Protocol) between two network interfaces associated with the IP address. There are two versions of IP (IPv4 and IPv6) commonly used and both make a lot of sense to computers but not so
much for a human being. Servers are often given a more human friendly hostname which a computer uses to find a corresponding IP address by requesting it from Domain Name Server (DNS). For example my computer (IP 10.7.100.158) will send DNS (10.7.100.101) a query asking what is the IP address for a server called ec2-52-28-53-105.eu-central-1.compute.amazonaws.com. DNS will acquire the requested IP address and sends it back to my computer. Now my computer knows what IP to use for that host.

![FIGURE 8. DNS request response content.](image)

All data between my computer and the server in Frankfurt travels in IP packages which has a source and a destination address in IP package headers. My computer passes this package to a gateway defined in a network interface configuration and relies that the gateway will know how to reach the destination address. In fact there are many hops before the destination server is reached and we have no control over the hops once an IP package has left our premises on the second hop.

![FIGURE 9. IP packet path from Dallas to Frankfurt.](image)

Some packages will not reach their destination and some packages end up traveling a different and faster route passing another package with a slower route. Transmission Control Protocol (TCP) ensures that all packages are received in a correct order and it
handles retransmissions if any packages are missing or corrupted. TCP has a method for identifying transfer errors and corrupted data but it does not offer any protection against attacks. TCP/IP is a perfect example of an unsecure channel.

### 3.2.1 Secure Shell

I’ll discuss briefly about Secure Shell (SSH) because it is a good example of using the methods mentioned in this chapter and SSH implementations usually don’t try to hide its behavior. In a picture below it is nicely visible how a client and a server agree on a secret key over an unsecure channel. When the Server and the Client have sent ‘New Keys’ –message, the communication is encrypted and an eavesdropper would only see that TCP is carrying encrypted SSH packets.
FIGURE 10. SSH handshake from packet capture
3.2.2 HTTP Over TLS

For those who like unix/linux shell SSH is an awesome tool. However, there are many people who prefer a graphical user interface, namely almost everybody and all of those who shower. Hypertext Transfer Protocol (HTTP) was developed together with HyperText Markup Language for transferring a Hypermedia content instead of files. HTTP sends requests to an HTTP server and it receives responses with a possible requested content and always with a status code. HTTP is not encrypted and HTML files are literally plaintext with human readable XML like syntax. The default authentication with html is sending BASE64 encoded credentials in HTTP headers. Base64 is encoding for storing binary data where ASCII is only expected and provides now protection for privacy. (Berners-Lee, 1992) (Berners-Lee, Fielding, Irvine, Gettys, Mogul, Frystyk, Masinter & Leach, 1999)

Secure Sockets Layer (SSL) and its successor Transport Layer Security (TLS) were designed to establish a secure connection between two communicating applications and for encapsulating higher layer protocols. HTTP over TLS (HTTPS) offers an HTTP functionality with an encrypted channel and an authenticated server. (Dierks & Rescorla, 2008)

Let’s look into TLS handshake between my computer and Google App Engine server (GAE). After a TCP handshake is completed, a client sends a TLS Hello message to a server with various parameters including a list of cipher suites the client supports. The server replies with a Server Hello message including a cipher suite select server selected from the list client supports (underscored with blue).

Server Hello is followed by a Certificate message which has a list of SSL Certificates. The first certificate on the list is the certificate for the server. The certificate includes a hostname and a public key for verifying the server’s digital signature. The certificate usually has more information about the organization and/or person certificate issued to. Each following certificate must directly certify the one preceding it. Before client can trust a received certificate, it must be able to validate at least one of the certificates. For
this reason, web browsers and browser API’s have a built-in list of trusted certificates. The list is called a certificate chain and last certificate is Root CA Certificate (Certificate Authority). (Dierks & Rescorla, 2008)

The client will then initiate a key exchange. In this instance the key exchange is done using Elliptic Curve Diffie-Hellman. Elliptic Curve Cryptography (ECC) has a lot shorter key lengths compared to RSA with the same level of security. (Blake-Wilson, Bolyard, Gupta, Hawk & Moeller, 2006)
FIGURE 11. TLS Handshake with HTTPS transfer.
3.3 SECURITY MAINTENANCE

A chain is only as strong as its weakest link. There has been number of instances that the secure communication chain has been weaker than the force trying to break it.

![Graph showing total number of vulnerabilities between 2006 and 2014](Symantec, 2015).

From server point of view it is crucial that software is kept up to date and insecure cipher suites will not be supported. Defining a correct set of supported cipher suites is a compromise between security and availability. Browsers are not always updated soon after update becomes available and therefore it would be inconvenient for a user if the access to a web server would be prevented because of an outdated set of cipher suites in the browser. Other side of the coin is that attackers will make sure that their client supports only the one cipher suite, the vulnerability of which is being exploited (Möller, 2014).

Even though maintaining a server is mostly quite tranquil after the server has been configured to do its job, there are times when there is no time to lose. Heartbleed is a bug in an OpenSSL implementation that caused a web server to include the content of memory to HTTP request. 4 hours after it had been published, Symantec was recording attacks trying to exploit it (Symantec, 2015).

ShellShock is an interesting vulnerability in a sense that it affects Bourne Again Shell (Bash) which is used not only in web servers but also in many other devices like routers.
(Symantec, 2015). When was the last time YOU updated the firmware for a router or a web camera?


As general, it seems that system administrators are not really hasty to address vulnerabilities. According to Symantec, the 3rd most common vulnerability is the support for SSLv2 which is about 20 years old and as number one is the SSLv3 vulnerability which has been discovered for almost 6 months ago (Symantec, 2015). According to SSL Pulse less than every fourth site is secure (TIM, 2015).

FIGURE 14. Top 10 vulnerabilities found unpatched on scanned web servers (Symantec, 2015) and summary for SSL Pulse summary (TIM, 2015).
In addition to most servers in the Internet having neglected security concerns, there are plenty of vulnerabilities on the client side, too. This isn’t really in the scope of this thesis but it needs to be mentioned for being able to interpret statistic. Also, according to Google, a support for TLS_FALLBACK_SCSV prevents Poodle attacks and in their opinion SSLv3 is still safe enough so that the GAE supports it.

![Plugin vulnerabilities by month and web browser vulnerabilities by year](image)

**FIGURE 15.** Plugin vulnerabilities by month and web browser vulnerabilities by year (Symantec, 2015).

As a conclusion of this chapter, I would like to note that there seems to be a trend to use HTTPS everywhere. In Google I/O 2014 -developers event there was a talk called HTTPS Everywhere where Google representatives encouraged developers to use HTTPS everywhere (Far & Grigorik, 2014). Google also seems to practice what they preach and youtube uses HTTPS as default (Atkins, 2015). Also, the open source community is developing an automated service for getting SSL certificates easy, fast and free. (Kerner, 2015)

There are people helping developers and system administrators to do better choices. For example, Mozilla and Google have security blogs which seem to be a good source for information. As for now it seems that a site should be quite secure by following the two guidelines:

1. Keep server applications and security updates up to date.
2. Don’t support over 10-year-old cipher suites.
4 PRACTICAL CONSIDERATIONS

Now we will focus on practical considerations for making the unauthorized use of our application less convenient. When our application (on left) is being launched, it needs to know if it’s a legitimate usage. We (on right) need to be able to provide that information to our application. In this chapter we’ll ponder how to replace question mark.

![Diagram](image)

FIGURE 16. How application will know if it is authorized or not.

A big part of our products run on smartphones. This introduces a few limitations that are commonly used in desktop computers. We know from our previous licensing schemes that copying files to a smartphone may be a hassle. Also, typing or copy/pasting a cryptic sequence of numbers and letters is error prone cumbersome. An application needs to be able to acquire knowledge of its authorization without annoying the user. The obvious answer is a server from which the application can ask if it’s properly licensed. In this chapter we will discuss methods for application to separate an unauthorized usage from an authorized and how to transfer that information to an application.
In this chapter I will use an imaginary notepad application called ‘apocalyptic notes’. I don’t plan to make one; instead I’ll use it in order to discuss features that wouldn’t make any sense in my real project.

4.1 Copy protection techniques

An unauthorized copying of software has been around from the beginning of the personal computer era. Therefore, there have been attempts to prevent the usage of unauthorized copies. Back in a day applications were always sold in a package which included some physical material like a paper manual. Many applications and games relied on the fact that an authorized user would have the retail package and in addition to media, storing the executable authorized user would have all the material that came with the package. Copying physical items is not as convenient than copying bytes.

One of my favorite games during 90’s was Stunts. At every launch it would ask the word in a manual in a given position. There were also attempts to prevent copying of manual by adding random characters on the background with a different color than the text. These were easy to read but black and white photo copier would produce an unreadable mess. Lenslok was an optical device bundled with game package which was used to make a scrambled text on a display readable. The only method from early days that still remains is a dongle. A dongle is a hardware device that needs to be connected to computer when the application is used. (Pingdom, 2009)

There are two methods the application can use to determine whether the usage is authorized or not. It can ask from somebody it trusts and like a child unknowing what to do, it will phone home. An application has one or more hardcoded URLs for connecting to a server which will accept or decline the usage.

4.1.1 Key token

The other option is to check if there is a key token available. A key token may be either hardware or software. A hardware key or a dongle is a hardware device that needs to be
connected when the application is used. The software key is basically a chunk of data that only authorized users have.

A software key is often a file in a mass storage but it doesn’t have to be. It could be anywhere where it is accessible by the application. As an example I have created a license for my imaginary notes application. I have used xml in my example just because it’s easy to read by a human being and the application.

The license contains two important entities. The name of the authorized product and a digital signature for verifying that <product> entity has not been tampered. When my ‘apocalyptic notes’ will open this license, it would read the ‘product’ entity and use the signature to verify

```
<?xml version="1.0"?>
<license>
   <product name="apocalyptic notes"/>
   <signature>ba9aa5609b125c3a5b7a47281cab9a36</signature>
</license>
```

**FIGURE 17. Simple software key example**

There is an obvious weakness in my license. This license can be copied as easily as the application itself. There is nothing to identify who has been authorized to use my application. If this application would be shared in P2P networks, the license file would be included in that same package and everyone would be able to use it.

Before focusing on how to prevent an unauthorized copy of a software key, authorized users need to be defined. Defining authorized users is more of a question of business model than a technical one. Usually, defining authorized users will give obvious limitations to the license.

Authorizing one device is quite a straightforward licensing scheme. A software key may be bound to any hardware identifier on the device executing the application. Data capable devices have always an identifier such as ESN, IMEI or MAC address of a
WIFI interface. Adding a device identification software key to a tamper proof license file is a pretty solid copy prevention.

<license>
  <product name="apocalyptic notes">
    <device imei="990004004105474"/>
  </product>
  <signature>854b43af233c2ad7a825f47b87c5b20b</signature>
</license>

FIGURE 18. Simple software key with device identification

However, using hardware identifiers with PC’s is not that unambiguous. PC’s usually don’t have one serial number. PC’s are a collection of parts which are interchangeable.

If authorization is given to a user or a group of users who are authorized to use the application on more than one device, enforcing a copy prevention is more difficult. One option is to include an authorized user in the license file. This will not prevent an unauthorized usage but it will make it less appealing to share the license in P2P networks. Also, if a software key has been compromised, it can be rejected in the following software releases.

<license>
  <product name="apocalyptic notes">
    <User name="Acme Inc" contact="james.patkukelka@achme.com"/>
  </product>
  <signature>3d5710cb2f5d528e1a2e3f93dc3d69f</signature>
</license>

FIGURE 19. Simple software key with user identification

4.1.2 Phone home

The problem with just a key token is that the method for a key validation is locked on the build time. Even though the author would learn that a software key has been compromised and is now shared publicly, there is nothing to be done before the next release. Phoning home allows the application to react on the information acquired after
the build time. At this point instead of ‘application’, I will use the term ‘client’, which phones home to the ‘server’.

Phone home also enables new business models like subscription and volume licensing that can actually be enforced by Product Activation. Ultraedit, my tool of choice for text editing, is an example of applications that utilizes Product Activation. Buying an Ultraedit for a personal use will authorize the user to install that application to 3 devices. The user will do Product Activation by entering the license id and password acquired at purchase. I made a simplified illustration of this pattern based on Ultraedit FAQ and observations on a packet capture. I want to emphasize that this is a simplification excluding all encrypted transfers content of which remains a mystery to me. (IDM Computer Solutions, 2015)
DIAGRAM 3. Simplified illustration of Ultraedit Product Activation
When designing phone calling feature, it needs to be taken into account that on wireless devices a data connection is not always available. Also, when a cellular network is used for communication, every byte may cost the user hard cold money.

### 4.1.3 Segmentation

A key token does not have to be a master key that would open every door, in this context every feature. Neither does it have to last forever. A key token is a versatile tool to customize one product for a customer’s needs and ability to pay premium for more advanced features.

Also, anyone selling products globally needs to recognize that there is a difference in the ability and willingness to pay for software between customers in different geographical regions. If the product would be priced at the level where developed markets would bring a good cash flow, it would not sell at all in developing countries. Therefore, there could be cheaper version for developing countries and a key token could be used to ensure it wouldn’t be used in developed countries.

Now let’s go back to my Apocalyptic Notes app. I could sell a basic version of my application with a really affordable price and it would be good for most use cases for average user. However, I could implement more advanced features that would make using the application more convenient, for example speech recognition. In my example Acme Inc purchased my application for making meeting notes. Speech recognition reduces manual work with notes and allows them to focus more on their core business. Speech recognition reduces manual work therefore it saves money and has more value.

If I would sell licenses that instead of lasting forever would actually expire at some point in time, I could expect to sell extensions and perhaps even feature upgrades. A prospect of getting more sales in future will motivate me to develop my product in addition to allowing initial an sale prize to be a bit lower. In my example I’ve sold Acme a license that will be valid until the end of 2016 and after that the license is considered to be expired (See expiration element). In addition, Acme will get free
updates to any version that is released before the end of year 2015 (see updates element).

```xml
<?xml version="1.0"?>
<license>
  <product name="apocalyptic notes">
    <expiration year="2016" month="12" day="31" />
    <updates year="2015" month="12" day="31"/>
    <user name="Acme Inc" contact="james.poikulnikko@acme.com" />
    <option>SPEECH_RECOGNITION</option>
  </product>
  <signature>deeb8ecf2c2dcdbe4fc5f0fb9d7a18d6</signature>
</license>
```

**FIGURE 20.** License with optional feature.

A key token can also be used for marketing paid advanced features which customers need but are not aware of it just yet. For example, when I’m delivering the Acme license for their purchased Apocalyptic Notes, I will give them a free of charge feature called Cloud Sync that will expire in the beginning of July. It is my wish that once it will expire, users will recognize how valuable feature it was and they want to upgrade the license with Cloud Sync.

```xml
<?xml version="1.0"?>
<license>
  <product name="apocalyptic notes">
    <expiration year="2016" month="12" day="31" />
    <updates year="2015" month="12" day="31"/>
    <user name="Acme Inc" contact="james.poikulnikko@acme.com" />
    <option>SPEECH_RECOGNITION</option>
    <option year="2015" month="07" day="01">CLOUD_SYNC</option>
  </product>
  <signature>9fabfe36dd7b5d2e7bbee133adc427cb</signature>
</license>
```

**FIGURE 21.** License with expiring optional feature.

### 4.1.4 What can be trusted

In the beginning of this chapter I have described a number of variables that a license could be bound to. While the price for a licensed product increases, the level of trust for acquiring these variables decreases. We’ll dig into this but let’s first list variables we need for enforcing:
The level of trust differs a lot between different platforms and methods used to acquire them. Also, the author of an affordable consumer application can be quite confident that some of the most extreme measures to forge variables just would not be worth it. However, for expensive professional tools, your copy protection technique may be attacked by ‘hackers of fortune’.

For example, let’s take an imaginary company Acme Pacific East Coast that provides 3D modeling services. The company has 1,000 employees doing 3D modeling and all of them a need copy of 3DS Max $1470 per year. For this company it seems like a financially sound decision to hire an army of hackers to break the copy prevention (Doherty, Gegeny, Spasojevic & Baltazar, 2013).

Device ID in smartphones is usually Electronic Serial Number (ESN) or international Mobile Equipment Identity. That would be a really good way to limit the usage of application to one device, if it wasn’t changeable. Google will tell you how to do it in no time. However there is a risk that the device will get blacklisted by operators globally if a fraud is being suspected (GSMA, 2015). There’s no more obvious indicator for a fraud than more than one device with one IMEI in one network. Also, some operators have a whitelist of devices that they accept in their network.

Contact information in data connections means a hostname or a direct IP address. If a network is in hostile control, it can’t be trusted that the server answering from home address is actually home. There’s at least a network router between the devices running
the application and the Internet. The router intermediates all data client sends, including requests to Domain Name Server (DNS). All communication in the Internet is done using IP addresses and DNSs are used to acquire an IP address for a hostname. A router could direct DNS requests to a rogue DNS server (DNS hijacking), which would give an address to a ‘forged home’ instead of a ‘home’. Also, packages sent to a ‘home’ IP address could be redirected to a ‘forged home’ (IP hijacking). A countermeasure to this attack is to communicate securely with the server and use digital signatures to make sure we’re really discussing with a legitimate server.

Also, it needs to be taken into account that sometimes networks just don’t work and especially in wireless systems a data network is not always available. Whether the cause for not being able to reach home is due to a hostile attack or normal network problems, it can’t be always be detected by a client.

FIGURE 22. Hostile network.
A region can be detected in a number of ways. Mobile devices usually provide Location API that will use cellular or wifi networks or GPS to detect a location. The location could be faked with for example an external GPS source that would send NMEA location messages with fake coordinates. I’ve done that for verifying that the application can handle situations where a longitude changes from the maximum positive to the negative maximum by crossing the opposite of the prime median. One way is to check Mobile Country Code (MCC) which is a quite good indication of current country even though it can be faked with a network simulator.

What’s the time is not as trivial question as it would first seem. Whether it is a smartphone or a computer, a user may change the system time freely. Network capable devices often acquire time using Network Time Protocol (NTP) from private or public server. However this method is subject to DNS and IP hijacking. GPS can be used to acquire GPS time, which is subject to a forged GPS source suspicion. A home server is a dependable source for time and also copy protection dongles often provide a secure source for time. Also, devices usually provide an uptime or tick count which is a good time source while the application is running.

Credentials identify the user or organization instead of the device. However it’s difficult to tell if an authorized user has given his credential to other users. There isn’t really any feasible way to prevent this. However, there are ways to discourage an unauthorized sharing of credentials. Team Support (a customer support central web application) allows only one web client to be logged in for one user account at any given time. So if you use it with two computers or two browsers at logon, it automatically ends any pre-existing sessions for your username. Also, utilizing highly personal services for the user authentication makes the authentication for a legitimate user easy and due to a personal content, it discourages sharing those credentials. For example, Sports Tracker utilizes a Facebook logon to enable one click logon when the browser has an open session with Facebook.

Executable and your code are not safe from attacks either. How much effort and skill is required for tampering with executable is highly platform dependent. Java applications
are especially vulnerable to reverse engineering as Java applications are compiled into a platform agnostic bytecode. Applications, which are compiled into a platform and CPU architecture specific executable binary, do not contain high level structures. Distributed binaries are merely a chunk of CPU instructions. Executable made with C++ can be converted into assembly which is still quite difficult to interpret. Also, there are approaches like encrypting the executable and decrypt it on demand. However it eventually boils down to the fact that CPU must be provided instructions without encryption and therefore attacker will be able to obtain it.

_Cryptographic keys_ are the essence of verifying the license. Generally, an application installation package should have only public keys which can be used to verify the integrity of communication with a home server or the license. As discussed previously, the installation package should be considered as public. Even though acquiring data by reverse engineering can be made hard it’s really hard to prevent it completely. If the application would have a need to store a private key, most platforms provide a secure method for storing keys (Android, 2015) (MSDN, 2015). But can we trust a platform provided key store? What if an attacker has its own platform like modified CyanogenMod which provides a rogue key store?

In short nothing is dependable, everything could be forged. What we can do is to use different methods together in a way that we’re able to provide a challenge at least and try to come up new innovative ways to provide a challenge to hackers while trying to keep the authorized user happy.

4.2 Case study: Reverse Engineering Android APK

All Android applications are packed in the Android Application Package (APK) file which is a ZIP archive of an executable bytecode (classes.dex) and resources. There are plenty of tools to reverse engineer apk and Java Archive (JAR) files back to readable code. In the example below I used dex2jar for converting classes.dex to classes_dex2jar.jar and jd-gui to read a jar file.
Android development tools have built-in countermeasure technique for reverse engineering called ProGuard which obfuscates the bytecode when the installation package is being created. Obfuscation does a lexical transformation to classes and variables making it harder to read. However, it does not hide the logic or initial values of local variables which can reveal everything. Also, lines added for a debug logging may reveal everything.
Using resources instead of initializing local variables doesn’t really complicate the reading of original source but it makes a word of difference in reverse engineered code. Also revealing a debug logging can be stripped by configuring ProGuard properly. ProGuard is not only a tool making reverse engineers life a bit more challenging but I chose it as it’s a default with Android tools.
There’s no going around the fact that Java code is quite easy to reverse engineer but it doesn’t mean that tampering an Android application would be that easy. Each apk is digitally signed with the author’s private key. Android does not install an application that doesn’t have a matching signature and after tampering, the application needs to be re-signed before it can be installed. In addition, it is easy for the application to read its signature programatically.

And there’s more. Android Native Development Kit (Android NDK) can be used to compile C++ code into native libraries that can be called from Java code using Java
Native Interface (JNI). In addition to being more difficult to debug, it is also more difficult to reverse engineer. Also, it can be used together with Java code to detect tampering by checking a native lib’s hash checksum on Java code and apk’s signature on native code. Sure it can be done, but it provides a better challenge than mere obfuscation. Also, native code could implement the communication with the server back home and allow server to verify that the apk signature is valid.

Obfuscation doesn’t seem like much when studying a small project. However, when I opened one Android application as example, I noticed that 36 of it’ 6400 .class files were named a.class.
4.3 The Server

Now I will proceed to discussion about a server. A server is an incredibly broad concept but at the end it boils down to any device that serves responses to requests clients send it. Today there are a lot of technical solutions for how a service can be made available to clients. The final part of this chapter discusses different approaches to provide a server.
For example, in a FIGURE 29 there are two totally different approaches for providing an https server with a battery backup. Another one is a server rack in one of Google’s data centers and another one is a Raspberry Pi on my desk. The main difference is that Raspberry Pi involves an initial material cost and it’s less scalable than Google, which is free to use with a similar work load than Raspberry Pi can handle. Also, Raspberry Pi is more vulnerable to spilled coffee.

![FIGURE 29. Google data center vs Raspberry Pi](image)

Before discussing which one is better, I will discuss a bit what requirements there are for a license server. The one thing you don’t want to do is to cause paying customers’ employees to be sitting idle unable to work because you spilled your coffee to the license server. I consider this scenario to be more damaging than somebody being able to use your product without paying. If the application keeps on being usable even when the connection to the license server fails, you don’t end up alienating pre-existing customers and that will give you some room for failures. Also, when we’re discussing OTA license servers we need to recognize the possibility of breaks in data connectivity.

Also, it needs to be addressed that the application needs to be able to connect to the server globally. In addition to technical issues, there may be political factors like Google App Engine can’t be accessed from China without some additional effort. As a precaution, we should have more than one address for a home server defined and prepared just in case the primary address would not be available anymore.

With Cloud services hardware skill and will is not required. Figure 30 is describing an illustration of different levels of cloud computing including required skill sets. Only
skill and will SaaS requires is using the application. PaaS requires someone to develop the application. On top of mentioned IaaS requires someone to design and manage the platform. (Redcentric, 2015)

![Different levels of cloud](redcentric_cloud.png)

*FIGURE 30. Different levels of cloud (Redcentric, 2015).*

### 4.3.1 Platform as a Service

*Platform as a Service (PaaS)* as defined by The NIST: "The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.” (Mell & Grance, 2011)

PaaS is an awesome starting point for those who want to focus on the application instead of platform. A developer does neither have a real control nor responsibility over the platform. There are some differences how much configuration different providers provide. For example, Amazon Elastic Beanstalk provides a small selection of Platforms and a possibility to disable auto scaling and load balancing (AWS, 2015). There are number of PaaS providers but I will focus on the GAE as it is the most familiar to me.
Google App Engine supports currently four programming languages: Python, Java, PHP and Go. However not all features and API’s are available, for instance a local file system is not available. When a developer deploys the application to the GAE, it will actually be copied to a number of fault tolerant servers running in Google’s data centers. When a client sends a request, Google will connect the request to one available server. The same applies for data storing. The GAE supports SQL and SQL like schemaless data storage location and Google will move that data where it’s needed. (Gibbs, 2008) (Chun, 2015)

I will demonstrate usage of the GAE with a really simple and inefficient code for finding next prime number after the initial number given as a parameter with HTTP GET request. On a client code there is a static method which takes two parameters, URL and a number of repeats launched simultaneously in their own thread.
FIGURE 32. Client and Server implementation.

I began testing with starting number at 100,000,000 and one request at a time. At the same time I was observing the GAE Dashboard to see how many requests I’m getting and how many instances are running my application. At this time one instance was enough to serve me.

At the second step I selected a smaller starting number for the prime number search and started sending requests sending 100 requests simultaneously. In addition, I configured a timer which would resend all requests that had been completed. Usually, my client got the response it was looking for in a few seconds. After setting a greater initial number to the prime number search, the GAE added a few more instances and responses kept on arriving in timely fashion.
With the Google App Engine it’s incredibly fast to implement an incredibly scalable server without any need to consider how much CPU or processing power would be needed for my application. If the GAE recognizes that more resources would be in order, it will launch another instance and the load will be balanced to one more instance. Within 30 minutes after installing the Google App Engine SDK, I had a service running. It was able to handle almost 30,000 CPU intensive requests with up to 30 instances running simultaneously and without any money spent from my part.
FIGURE 34. Google App Engine under stress with 20 CPU intensive request / second.

The GAE has a free quota which will be reset every 24 hours. The free quota includes 28 Instance Hours which don’t last long when utilizing 30 simultaneous instances. When using App Engine to run a service that needs to be available, it is important to setup a billing account for avoiding running out of free quota and service to become unavailable. Even if at normal usage the free quota is more than enough, it needs to be recognized that out of the ordinary situations. For example, administrators’ data base
integrity checks may involve a high number of database queries which might exceed the free quota.

At the development phase there are a few things developer needs to take into account. The GAE SDK enables running the server locally which makes it faster and easier to debug and verify that everything goes as planned. The GAE has its own methods for storing data and it is a good practice to hide the GAE implementation behind a generic interface as a precaution to event that another service would be chosen or the GAE would introduce a another method. Anything that differs from a standard way of doing server code with a particular language should be separated from the core code. The GAE requires that requests are completed in 60 seconds or it will cancel the request. Typically, this shouldn’t be a problem but in case of long lasting requests, a task should be given to backend instead.

**Error: Server Error**

*The server encountered an error and could not complete your request.*

Please try again in 30 seconds.

*FIGURE 35. Error message if App Engine requests takes longer than 60 seconds.*

**4.3.2 Infrastructure as a Service**

*Infrastructure as a Service* (IaaS) as defined by The NIST: "The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls)." (Mell & Grance, 2011)
IaaS gives a lot of control and responsibility and only the hardware is provided. IaaS usually provides one or more Virtual Machine instances where a customer is in a role of a system administrator and the only authority of those Virtual Machine instances. The single most haunting responsibility is to keep instances secure. In opposed to PaaS where there is an army of professionals with bleeding edge information of cyber security, in IaaS it is only YOU and the set of skills and personnel at your disposal. The other side of the coin is that there is most likely less skills and ambition attacking individual IaaS VM instances than Google App Engine. Therefore, IaaS may provide some Security through obscurity.

A configuration of IaaS instance begins with a selection of region where the server instance locates followed by a selection of desired hardware resources, operating systems and the physical location of instance. Amazon EC2 and Google Computing Engine both provide predefined sets of hardware profiles optimized for different purposes.

![FIGURE 36. AWS region selection.](image)

I will fire up an example server for looking into the process how to create an IaaS server. Again there are a number of service providers for IaaS but I will focus only one. Google provides IaaS called Google Computing Engine. However, in this example I will focus on Amazon EC2 instance.
In the first step I will select Amazon Machine Image (AMI) to start with. From 22 choices from Windows Servers to different Linux distributions I will select Ubuntu Server. At the second step it’s time to select hardware resources. Amazon provides a selection of 29 instance types. The cheapest comes with 1 virtual CPU, 1 GiB memory, and a low or moderate network performance. There are options available up to 36 virtual CPUs, 244 GiB memory and a 24 x 2048 GB storage.

![AWS virtual hardware selection](image)

**FIGURE 37. AWS virtual hardware selection.**

At the third step it’s time to select a number of instances and network settings. Instances may be purchased as spot instances where you can define a price which you are willing to pay for an instance hour. Depending on Amazon’s hardware utilization and spot price, the instance may or may not be running. A spot instance is an attractive option for CPU intensive applications that do not have to be available always.

Each AWS account has one or more virtual networks called Amazon Virtual Private Cloud (VPC). Also, it can be configured whether a service will be assigned IP
automatically from Amazon’s public address pool. If there’s a need for a fast networking between two servers, they should be located in the same placement group.

If the application running on the server will need any access privileges, it the application needs to have an access to AWS credentials or those privileges can be granted to all applications running in this server by selecting the Identity and Access Management (IAM) role.

FIGURE 38. AWS instance detail configuration.

The fourth step is for configuring data storages. My instances didn’t have any local storage. Local storages are not persistent and they will be cleared every time the service stops. Instead I’m using Amazon Elastic Block Store (Amazon EBS) as a data storage. In the fifth step the server can be configured with key-value pair tags for example ‘purpose’=‘webserver’.
On the sixth step it’s time to configure a firewall for incoming connections. In this example I enabled incoming connections to default HTTP and HTTPS ports without any limitations with respect to a source IP address. In addition, I enabled incoming TCP connections to a default SSH port from my own IP address. Also, I enabled incoming connections to a port 8080 from my company subnet. Limiting TCP connections to my company subnet only would allow me to create an administrative interface that would not be accessible from outside our company network. This is not really a bulletproof protection but it makes hackers life a bit more difficult.

**FIGURE 39. AWS storage selection.**
Finally it’s time to review the instance and launch. For a secure communication with a new server I had a choice of using existing key pair when I would send a public key to a newly created server while I would hold on to my private key. Another option is to create a new key pair where Amazon will create private and public keys. A public key is stored in my server and I will download the new private key. After this point I’m only person in possession of the private key.
Due to unexpected brain fart I ended up losing the private key file. I foolishly assumed that I would be able to assign another one from the management console. I was wrong. It’s easy to create or import keys to the AWS management console but a public key is assigned to instance at launch and changing a key pair after that can’t be done. The only way to recover from this is to create an image from the instance and launch a new instance from that image and assign a new key pair. Launching new instances is really easy. Configuring a new instance with existing AMI and security groups takes less than a minute and the initialization of the new instance takes a few minutes.

Instead of connecting the server instance directly, it’s a good idea to use a load balancer as a connection point. The load balancer can be used to handle HTTPS SSL Certificates and to select which Ciphers are allowed. Making good decisions by enabling and disabling ciphers requires some knowledge and that’s why choosing predefined security policy is a sound choice.
If there were any reason to pull any instance out of service, it would be enough to remove that instance from the load balancers instance list manually or by causing load balancers health check to fail. As a default, a load balancer considers an instance to be healthy if it gets a successful response for GET index.html request. As long as there remains other healthy instances with enough capacity, pulling instances from service is completely transparent to end users.

Load balancing isn’t enough when there’s more work to be done than there are workers working. By configuring an auto scaling, new instances will be automatically launched when existing instances are not responding fast enough. Dynamically launched
instances can be configured with the same or different hardware configuration, which allows firing up better performance resources when the default performance is not sufficient.

With load balancing and auto scaling, it needs to be recognized that there is no guarantee that two consecutive requests would be connected to the same instance, in fact odds are against it. Andreas Chatzakis, a solution architect in Amazon AWS, recommends separating a web server and a data storage. Using one separate database from the web server instance allows spawning new web servers when more capacity is needed. (Chatzakis, 2014)

Finally, we have reached a point where we have a dynamically scalable service. It’s not quite as dynamic as PaaS example and its physical location remains where we set it up to be. There’s one more thing to do for ensuring the data availability in an unlikely event that there would be issues in the cross continental network. Running web server instances in more than one continent with Cross-Region Read Replicas for database provides a good protection against regional disasters in addition to providing better database read response times. Add geolocation routing together with instance health checks and we have a similar solution than Google App Engine with respect to redundancy and decentralizing risks. (Barr, 2013) (AWS, 2015)

Now that we have acquired an idea of what kind of configuration is needed for hardware resources, we are ready to install a web server. In my example, I’m using Ubuntu and installing a web server application is one liner in shell ‘sudo apt-get install tomcat7’ and the platform is ready for an application development.

4.3.3 On-premises

Having own hardware in own premises is the traditional way to run web servers. The servers on premises rely on system admins’ ability to do good choices instead of bad ones. There are all the same responsibilities than with IaaS but instead of adding features from a console, it requires connecting cables and firing up a real hardware.
There are a lot of things that should be addressed when doing everything in house. Cisco’s Data center design guide presents different layers of data center as building blocks.

For avoiding a whole data center to go black in an event of hardware failure every component should be duplicated. While using either Amazon’s or Google’s cloud services, all data storage is redundantly stored to more than one site, which is pretty much the only protection in case the data center would be destroyed for example in a fire.

FIGURE 44. Data center pyramid of service layers (Cisco, 2014).
Also, a physical environment needs to be addressed. Even though servers could lie on the table, it would be better to dedicate a room for data center devices. Servers should be located in server racks with arranged cooling. Also, data center may grow up to consume a lot of power and it needs to be considered how much power must be supplied to the server room.

Servers and network components all need power to work and the whole chain needs to be powered or nothing works. Where to get power to run servers in an event of power outage? Also, it is good to keep in mind that Uninterrupted Power Supplies (UPS) are usually conversions that actually consume power. Google has an embedded battery backup in each server for avoiding this (Shankland, 2009).
Now that we have defined nice descriptions between PaaS, IaaS and on-premises, it’s time to mix them up a bit. Amazon and Google provide a public cloud which is available to everybody. An organization could manage its own private cloud where centralized computers would be shared between different users inside the organization. A private cloud may or may not be on-premises. In addition, on-premises solution could have a backup for high demand or hardware failure situations. It would do a cloud bursting to accommodate requests that it is unable to complete in timely fashion. A private cloud that uses partly own resources and partly public cloud is called a hybrid cloud.

4.3.4 Containers

Another interesting virtualization technique is containers. In AWS EC2 each server in Amazon’s data center is running a hypervisor, Virtual Machine Monitor (VMM), which allocates hardware resources to one or more Virtual Machines (VM). In IaaS example AWS VMM allocated us a virtual machine instance which loaded full blooded operating systems of my choice (Ubuntu). Instead of containing an operating system, a container contains the application and libraries it requires. Therefore container applications are much smaller than VM images.
Containers and VM do not have to be mutually exclusive, instead they can be used together. Using a container engine without a hypervisor binds a hardware to one operating system. With a hypervisor each VM has its own operating system and is completely isolated from any other VM running on the same hardware. (Bernstein, 2014)
The most commonly used container is Docker. Joyent provides a public cloud for running Dockers in addition to providing the same platform to be used in private clouds (Fine, 2014). Google Container Engine is powered by an open source project called Kubernetes for Dockers. AWS has EC2 Container Service for Dockers. It seems a sweet deal to develop a server application as Docker and to spawn instances where ever it seems to make sense at the time. The answer might even be different in different regions. For example, if we had existing on-premises data centers in Americas and in Europe, we could use AWS ECS Container Service to run same Dockers in Asia that we run in our on-premises servers. Request to corresponding regions would be managed by geo routing.
5 CONCLUSIONS

Implementing a license enforcing scheme requires resources. There’s only one thing worse for business than unauthorized users; absent users. Using resources to prevent an illicit usage takes resources from acquiring more legit users and it comes with a risk of alienating paying users. Therefore, in my opinion any attempt to prevent the illicit usage needs to be designed and implemented carefully. The further you go in preventing the illicit usage, the better equipped you need to be when surprises occur. While keeping that in mind let’s conclude five approaches with a modified cloud pyramid.

![Figure 49. Five levels of stress over license server.](image)

The easiest option is not worrying about it. Focus on making your product so good that people are willing to pay for it. Also, if an application is distributed using mobile application stores they usually have built-in solutions for reducing an unauthorized copying.

Using Software as a Service in this context would be a third party copy prevention solution. This doesn’t really release you from the responsibility to keep paying customers happy. If paying customers’ experience problems with the copy prevention solution you have chosen, it’s still your fault.
At every step while descending on the pyramid, you increase the number of choices you need to figure out yourself in good and bad. A correct level is where you feel confident that your knowhow and resources are best at use and where you feel confident that you are able to do as good or better choices than commercial service providers; so called professionals.
REFERENCES


Android, Android Keystore System. Date of retrieval 25.04.2015
https://developer.android.com/training/articles/keystore.html

https://www.youtube.com/watch?v=2uzK3VwzraM

http://www.theregister.co.uk/2014/10/08/digital_piracy_is_killing_creative_industries_says_industry_group/

Barr, J. 2013. Cross-Region Read Replicas for Amazon RDS for MySQL. Date of retrieval 25.04.2015

BBC Worldwide. 2014. Submission in response to the Australian government’s online copyright infringement discussion paper. Date of retrieval 25.04.2015

http://www.w3.org/History/19921103-hypertext/hypertext/WWW/Protocols/HTTP.html


http://d0.awsstatic.com/whitepapers/wordpress-best-practices-on-aws.pdf

Chun, W. DevBytes - File Storage in the Cloud, Date of retrieval 25.04.2015
https://www.youtube.com/watch?v=vylap827rHs

Cisco. 2014. Data Center Technology Design Guide. Date of retrieval 25.04.2015

Dierks, T., Rescorla, E. 2008. IETF, HTTP over TLS. Date of retrieval 25.04.2015


https://www.youtube.com/watch?v=cBhZ6S0PFCY

Fine, B. 2014. Making Joyent the Best Place to Run Docker. Date of retrieval 25.04.2015
https://www.joyent.com/blog/making-joyent-the-best-place-to-run-docker


https://www.youtube.com/watch?v=oG6Ac7d-Nx8

Graham, L. 2000. The Register, MS' Ballmer: Linux is communism. Date of retrieval 25.04.2015
http://www.theregister.co.uk/2000/07/31/ms_ballmer_linux_is_communism/

Green, T.C. 2001. The Register, Ballmer: “Linux is a cancer”. Date of retrieval 25.04.2015
http://www.theregister.co.uk/2001/06/02/ballmer_linux_is_a_cancer/
GSMA. Fraud & Security, Date of retrieval 25.04.2015
http://www.gsma.com/technicalprojects/fraud-security

IDM Computer Solutions. Ultraedit Licensing and activation FAQ. Date of retrieval 25.04.2015
http://www.ultraedit.com/support/activation-faq.html

http://www.iltalehti.fi/uutiset/2012060515670142_uu.shtml


Karaganis, J., Renkema, L. Copy culture in us and Germany. Date of retrieval 25.04.2015

Kerner, S.M. 2015. eWeek, Let’s Encrypt. Date of retrieval 25.04.2015

http://www.nytimes.com/2010/09/12/world/europe/12raids.html?_r=0


Metz, C. 2012. Wired, Meet Bill Gates, the man who changed open source software. Date of retrieval 25.04.2015
http://www.wired.com/2012/01/meet-bill-gates/

Microsoft Developer Network. Walkthrough: Creating a Cryptographic Application. Date of retrieval 25.04.2015

Muncaster, P. 2014. The Register, China cuffs 60000 pirates in 2013 crackdown. Date of retrieval 25.04.2015
http://www.theregister.co.uk/2014/01/24/china_piracy_crackdown_arrests/

Möller, B. 2014. Google Security Team, This POODLE bites: exploiting the SSL 3.0 fallback. Date of retrieval 25.04.2015
http://googleonlinesecurity.blogspot.co.uk/2014/10/this-poodle-bites-exploiting-ssl-30.html


http://tekniikanmaailma.fi/vanhat/vialliset-levyt

Parloff, R. 2007. Fortuen, Microsoft takes on the free world. Date of retrieval 25.04.2015

Pingdom. 2009. Wacky copy protection methods from the good old days. Date of retrieval 25.04.2015

Redcentric. What is infrastructure as a service (IaaS)? Date of retrieval 25.04.2015
http://www.redcentricplc.com/resources/articles/what-is-iaas/


http://www.gnu.org/philosophy/free-software-even-more-important.html

Stalsberg, A. The Vlfberht sword blades reevaluated. Date of retrieval 25.04.2015
http://jenny-rita.org/Annestamanus.pdf

Symantec. 2015. Internet security threat report, Volume 20

https://www.teosto.fi/teosto/uutiset/chisu-gatessa-sovinto

TERA Consultants. 2014. The Economic Contribution of the Creative Industries to the EU in terms of GDP and Jobs. Date of retrieval 25.04.2015

The Economist. 2005. BSA or just BS?, Software theft is bad; so is misstating the evidence. Date of retrieval 25.04.2015
http://www.economist.com/node/3993427

The Economist. 2012. Online software piracy, Head in the clouds. Date of retrieval 25.04.2015
http://www.economist.com/blogs/graphicdetail/2012/07/online-software-piracy

https://www.trustworthyinternet.org/ssl-pulse/

Wikipedia. 2015. Protests against SOPA and PIPA. Date of retrieval 25.04.2015
http://en.wikipedia.org/wiki/Protests_against_SOPA_and_PIPA