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 Mönttinen 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

TERMS AND ABBREVIATIONS

AES	Advanced Encryption Standard. Encryption standard defined by NIST.
API	Application Programming Interface. API defines an abstract programming interface.
APK	Android Application Package. Installation package for Android applications.
ASCII	American Standard Code for Information Interchange. ASCII is a one 8 bit character.
AWS	Amazon Web Services.
Base64	Base64 is binary-to-text encoding scheme.
Bash	Shell for unix/linux.
CPU	Central Processing Unit. Commonly known as processor.
DES	Data Encryption Standard. Deprecated encryption standard defined by NIST.
DNS	Domain Name System. System that translates hostnames into IP addresses.
ECC	Elliptic Curve Cryptography.
ESN	ESN. Electric serial number.
FAQ	Frequently Asked Questions.
GAE	Google App Engine. SaaS cloud provider by Google.
GNU	GNU’s Not Unix. Unix like operating system.
GPS	Global Positioning System. Satellite based positioning system.
GSMA	GSM Association.
HTML	HyperText Markup Language. Markup language designed for multimedia content.
HTTP	HyperText Transfer Protocol. Data transfer protocol.
HTTPS	HTTP over TLS. Secure transfer protocol.
IAM	AWS Identity and Access Management.

IMEI	International Mobile Station Equipment Identity. Similar to ESN.
IP	Internet Protocol. OSI layer 3 communication protocol.
JAR	Java Archive. Package file for Java application.
JNI	Java Native Interface. Interface for accessing native methods from Java.
LAN	Local Area Network. A local computer network.
MAC	Media Access Control. MAC address is a layer 2 address for interface.
MCC	Mobile Country Code. Country code in cellular networks.
MSDN	Microsoft Developer Network.
NIST	National Institute of Standards and Technology. US standardization organization.
NTP	Network Time Protocol.
OpenSSL	Open source implementation for SSL and TSL protocols.
OTA	Over-the-Air.
P2P	Peer-to-Peer.
PC	Personal Computer.
PGP	Pretty Good Privacy. Cryptosystem.
ROT13	Rotate by 13 places. Similar weak encryption than Caesar cipher.
RSA	Public key cryptosystem.
SDK	Software Development Kit.
SQL	Structured Query Language. Language for database queries.
SSH	Secure Shell. Encrypted network protocol text based shell session.
TCP	Transmission Control Protocol. OSI layer 4 communication protocol.
TIM	Trustworthy Internet Movement.
TLS	Transport Layer Security. Provides secure connection that for example HTTPS.

UPS	Uninterrupted Power Supply. Provides battery backup for electric devices.
URL	Uniform Resource Locator. Usually Internet address including protocol.
WIFI	Wireless LAN technology.
VM	Virtual Machine.
VMM	Virtual Machine Manager.
VPC	Virtual Private Cloud. Cloud version of Virtual Private Network.
XML	Extensible Markup Language. Really generic markup language.

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 that non-users (Karaganis & Renkema, 2015).

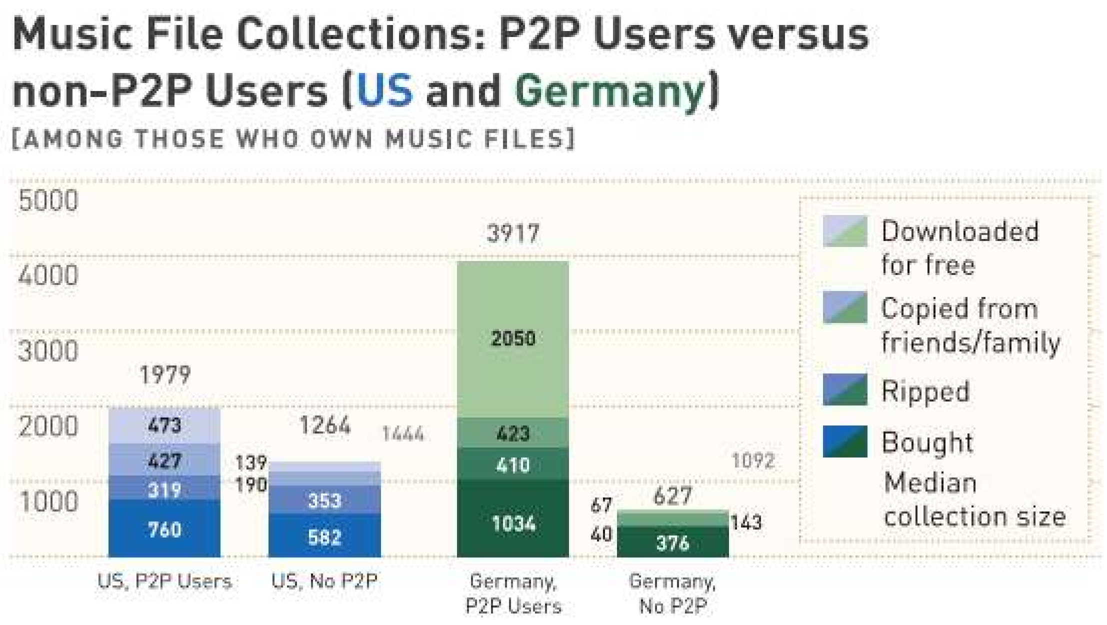


FIGURE 1. Music collections collection p2p users vs non-users (Karaganis & Renkema, 2015).

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, 2014). 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)

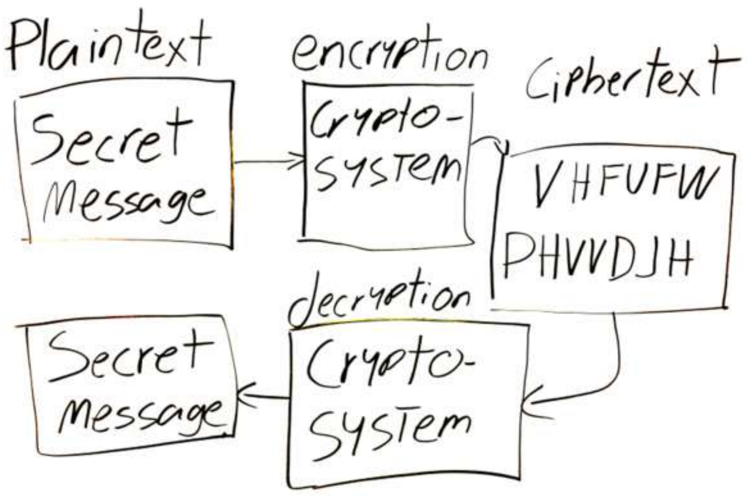
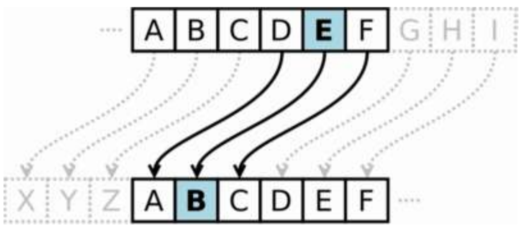


DIAGRAM 1. 'Secret message' encrypted and decrypted with Caesar cipher. (Kahn, 1973)

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)



Plain	a	b	c	d	e	f	g	h	I	j	k	l	m
Cipher	D	E	F	G	H	I	J	K	L	M	N	O	P
Plain	n	o	p	q	r	s	t	u	v	w	x	y	z
Cipher	Q	R	S	T	U	V	W	X	Y	Z	A	B	O

DIAGRAM 2. Caesar alphabets. (Kahn, 1973)

3.1 Modern Cryptography

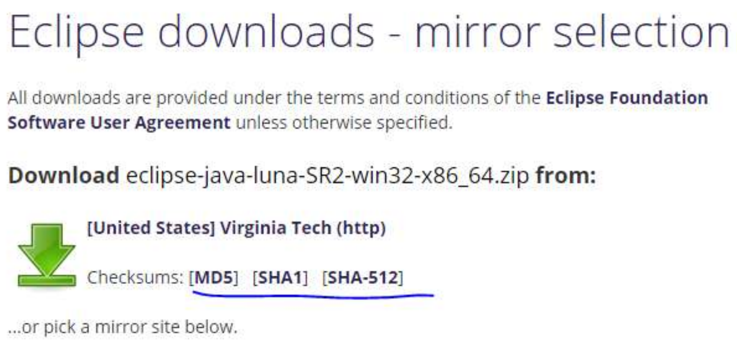
Instead of secret methods modern cryptography relies on secret keys with commonly known methods.

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 a 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 identic 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-SR2-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 α and q are shared openly.

$$Y = \alpha^X mod\, q, \ for\, 1 \leq X \leq q - 1 \qquad \textit{FORMULA 1}$$

$Y = public\ key$

$X = private\ key$

$\alpha = base\ (primitive\ root\ of\ q)$

$q = modulo\ (prime\ n\ umber)$

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.

$K_{ab} = \alpha^{X_a X_b} mod\ q = Y_a^{X_b} mod\ q = Y_b^{X_a} mod\ q$ *FORMULA 2*

$K_{ab} = Secret\ key$

$X_a = Alice's\ private\ key$

$X_b = Bob's\ private\ key$

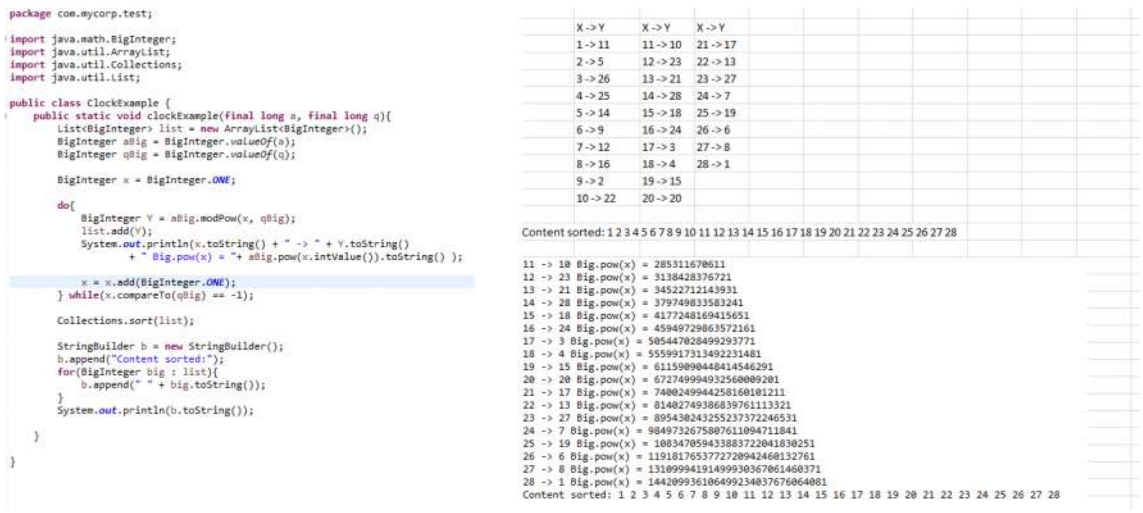
TABLE 1 is an example of how Alice and Bob agree to use the number 16 as a secret key without saying it directly. At the beginning α and q are agreed publicly (α = 11 q = 29). Alice has a private key Xa = 7 and Bob has a private key Xb = 12.

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

Alice *Variable*	*Value*		*Value*	*Bob* *Variable*
X_a	*3*		*12*	X_b
$Y_a = 11^3\ mod\ 29$	*26*		*23*	$Y_b = 11^{12}\ mod\ 29$
$Y_{ab}=23^3\ mod\ 29$	***16***		***16***	$Y_{ab}=26^{12}\ mod\ 29$

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 α 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 α and 1.



```java
package com.mycorp.test;

import java.math.BigInteger;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class ClockExample {
    public static void clockExample(final long a, final long q){
        List<BigInteger> list = new ArrayList<BigInteger>();
        BigInteger aBig = BigInteger.valueOf(a);
        BigInteger qBig = BigInteger.valueOf(q);

        BigInteger x = BigInteger.ONE;

        do{
            BigInteger Y = aBig.modPow(x, qBig);
            list.add(Y);
            System.out.println(x.toString() + " -> " + Y.toString()
                    + " Big.pow(x) = "+ aBig.pow(x.intValue()).toString() );

            x = x.add(BigInteger.ONE);
        } while(x.compareTo(qBig) == -1);

        Collections.sort(list);

        StringBuilder b = new StringBuilder();
        b.append("Content sorted:");
        for(BigInteger big : list){
            b.append(" " + big.toString());
        }
        System.out.println(b.toString());

    }

}
```

X -> Y	X -> Y	X -> Y
1 -> 11	11 -> 10	21 -> 17
2 -> 5	12 -> 23	22 -> 13
3 -> 26	13 -> 21	23 -> 27
4 -> 25	14 -> 28	24 -> 7
5 -> 14	15 -> 18	25 -> 19
6 -> 9	16 -> 24	26 -> 6
7 -> 12	17 -> 3	27 -> 8
8 -> 16	18 -> 4	28 -> 1
9 -> 2	19 -> 15	
10 -> 22	20 -> 20	

Content sorted: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28

```
11 -> 10 Big.pow(x) = 285311670611
12 -> 23 Big.pow(x) = 3138428376721
13 -> 21 Big.pow(x) = 34522712143931
14 -> 28 Big.pow(x) = 379749833583241
15 -> 18 Big.pow(x) = 4177248169415651
16 -> 24 Big.pow(x) = 45949729863572161
17 -> 3 Big.pow(x) = 505447028499293771
18 -> 4 Big.pow(x) = 5559917313492231481
19 -> 15 Big.pow(x) = 61159090448414546291
20 -> 20 Big.pow(x) = 672749994932560009201
21 -> 17 Big.pow(x) = 7400249944258160101211
22 -> 13 Big.pow(x) = 81402749386839761113321
23 -> 27 Big.pow(x) = 895430243255237372246531
24 -> 7 Big.pow(x) = 9849732675807611094711841
25 -> 19 Big.pow(x) = 108347059433883722041830251
26 -> 6 Big.pow(x) = 1191817653772720942460132761
27 -> 8 Big.pow(x) = 13109994191499930367061460371
28 -> 1 Big.pow(x) = 144209936106499234037676064081
Content sorted: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
```

FIGURE 3. Clock arithmetic example with primary roots.

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 2. Rough time estimates for trying all combinations for 300 digits long key.

Possible keys	1E+300						
cpu's	1	10	100	1000	10000	100000	1000000
seconds	1,3E+290	1,3E+289	1,3E+288	1,3E+287	1,3E+286	1,3E+285	1,3E+284
days	1,4E+285	1,4E+284	1,4E+283	1,4E+282	1,4E+281	1,4E+280	1,4E+279
years	5,3E+287	5,3E+286	5,3E+285	5,3E+284	5,3E+283	5,3E+282	5,3E+281

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

Rivest, Adi Shamir and Leonard Adlerman introduced an implementation called RSA (Rivest, Adi and Adlerman) (Rivest, Shamir& Adleman, 1978).

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 ϕ(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 ϕ(n). Relatively prime means that d and ϕ(n) do not have a greater common divisor than 1. e can be calculated from an equation below. (Rivest, Shamir& Adleman, 1978)

$$e \cdot d \equiv 1 \pmod{(p-1)\cdot(q-1)}$$ 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 $c = m^e \bmod n$ Decryption $m = c^d \bmod n$ 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 $s = m^d \bmod n.$ Decryption $m = s^e \bmod n$ *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 $\phi(n)$. Even though generating an RSA key pair may be CPU intensive, using a created key pair is not.

TABLE 3. Generation time statistics for 100 key pairs.

Key length	Minimum ms	Average (ms)	Maximum (ms)
1024	13	43	166
2048	41	296	1490
4096	560	3689	13694

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)

	Key Length *(Nk words)*	**Block Size** *(Nb words)*	**Number of Rounds** *(Nr)*
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

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.

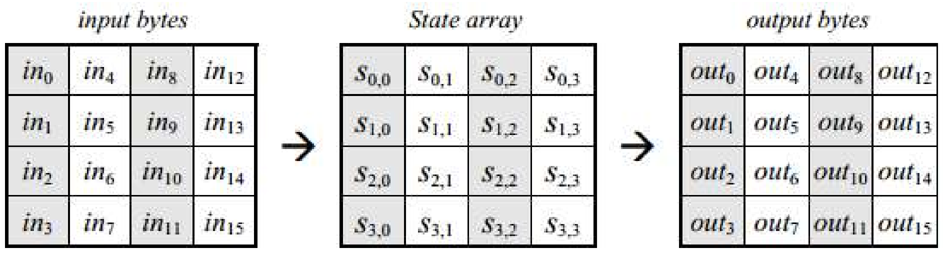
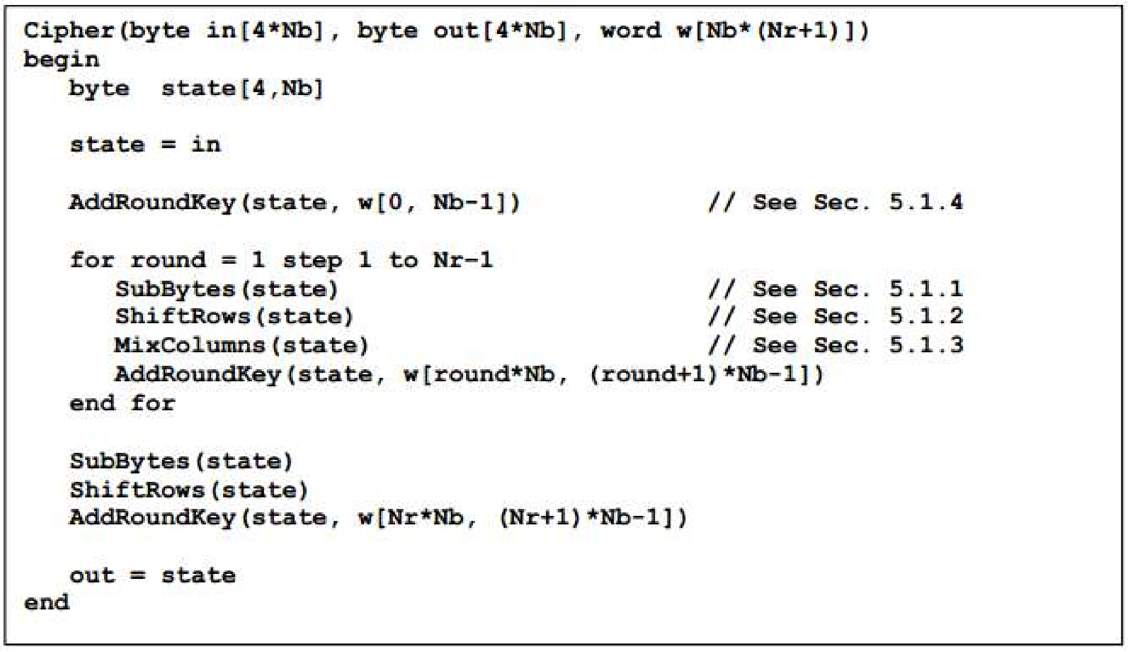

input bytes
State array
output bytes
in_0 in_4 in_8 in_{12}
in_1 in_5 in_9 in_{13}
in_2 in_6 in_{10} in_{14}
in_3 in_7 in_{11} in_{15}
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
out_0 out_4 out_8 out_{12}
out_1 out_5 out_9 out_{13}
out_2 out_6 out_{10} out_{14}
out_3 out_7 out_{11} out_{15}

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.

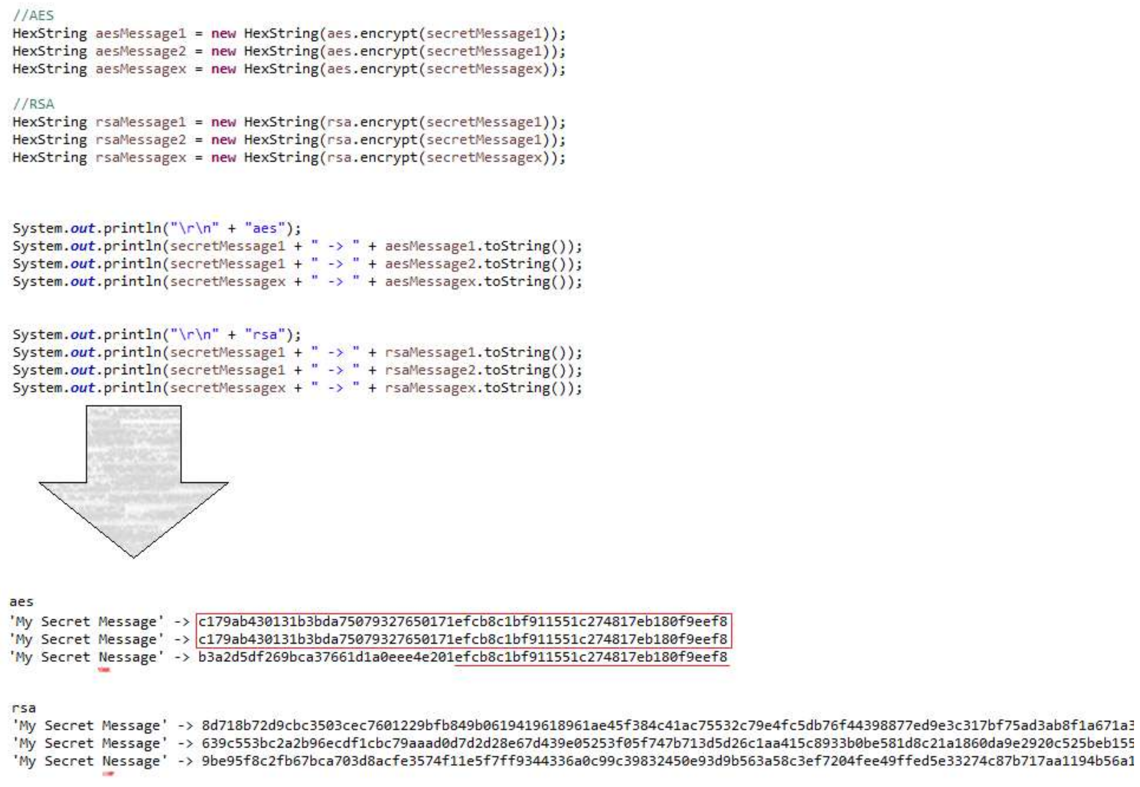


```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])            // See Sec. 5.1.4

   for round = 1 step 1 to Nr-1
      SubBytes(state)                        // See Sec. 5.1.1
      ShiftRows(state)                       // See Sec. 5.1.2
      MixColumns(state)                      // See Sec. 5.1.3
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for

   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   out = state
end
```

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.



```java
//AES
HexString aesMessage1 = new HexString(aes.encrypt(secretMessage1));
HexString aesMessage2 = new HexString(aes.encrypt(secretMessage1));
HexString aesMessagex = new HexString(aes.encrypt(secretMessagex));

//RSA
HexString rsaMessage1 = new HexString(rsa.encrypt(secretMessage1));
HexString rsaMessage2 = new HexString(rsa.encrypt(secretMessage1));
HexString rsaMessagex = new HexString(rsa.encrypt(secretMessagex));


System.out.println("\r\n" + "aes");
System.out.println(secretMessage1 + " -> " + aesMessage1.toString());
System.out.println(secretMessage1 + " -> " + aesMessage2.toString());
System.out.println(secretMessagex + " -> " + aesMessagex.toString());


System.out.println("\r\n" + "rsa");
System.out.println(secretMessage1 + " -> " + rsaMessage1.toString());
System.out.println(secretMessage1 + " -> " + rsaMessage2.toString());
System.out.println(secretMessagex + " -> " + rsaMessagex.toString());
```

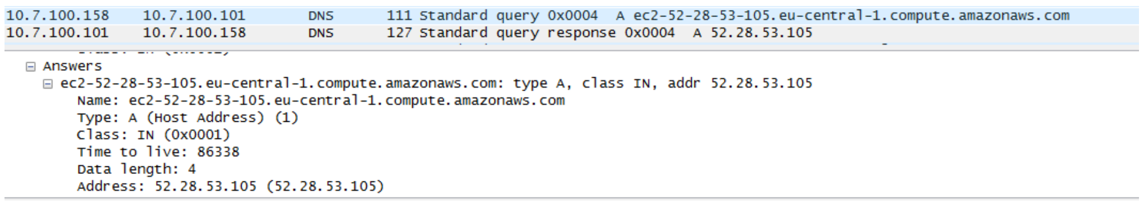
```
aes
'My Secret Message' -> c179ab430131b3bda75079327650171efcb8c1bf911551c274817eb180f9eef8
'My Secret Message' -> c179ab430131b3bda75079327650171efcb8c1bf911551c274817eb180f9eef8
'My Secret Nessage' -> b3a2d5df269bca37661d1a0eee4e201efcb8c1bf911551c274817eb180f9eef8

rsa
'My Secret Message' -> 8d718b72d9cbc3503cec7601229bfb849b0619419618961ae45f384c41ac75532c79e4fc5db76f44398877ed9e3c317bf75ad3ab8f1a671a3
'My Secret Message' -> 639c553bc2a2b96ecdf1cbc79aaad0d7d2d28e67d439e05253f05f747b713d5d26c1aa415c8933b0be581d8c21a1860da9e2920c525beb155
'My Secret Nessage' -> 9be95f8c2fb67bca703d8acfe3574f11e5f7ff9344336a0c99c39832450e93d9b563a58c3ef7204fee49ffed5e33274c87b717aa1194b56a1
```

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.

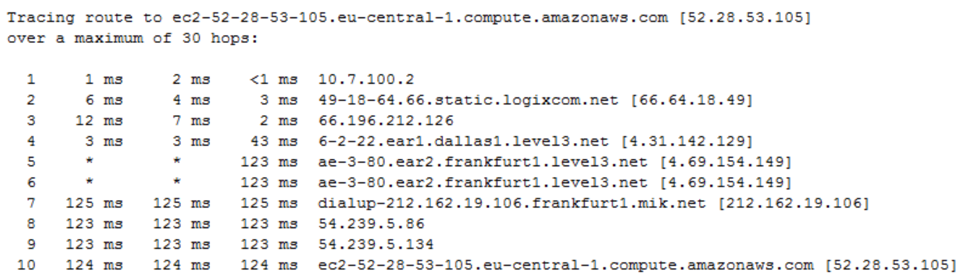
Virtually, all traffic over the Internet travels in IP packages (Internet Protocol) between two network interfaces associated with the IP address. There are two versions of IP (IPv4 and IPv6) commonly used and both make a lot of sense to computers but not so much for a human being. Servers are often 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.



```
10.7.100.158    10.7.100.101        DNS       111 Standard query 0x0004   A ec2-52-28-53-105.eu-central-1.compute.amazonaws.com
10.7.100.101    10.7.100.158        DNS       127 Standard query response 0x0004   A 52.28.53.105
  ⊟ Answers
    ⊟ ec2-52-28-53-105.eu-central-1.compute.amazonaws.com: type A, class IN, addr 52.28.53.105
        Name: ec2-52-28-53-105.eu-central-1.compute.amazonaws.com
        Type: A (Host Address) (1)
        Class: IN (0x0001)
        Time to live: 86338
        Data length: 4
        Address: 52.28.53.105 (52.28.53.105)
```

FIGURE 8. DNS request response content.

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 hobs once ab IP package has left our premises on the second hop.



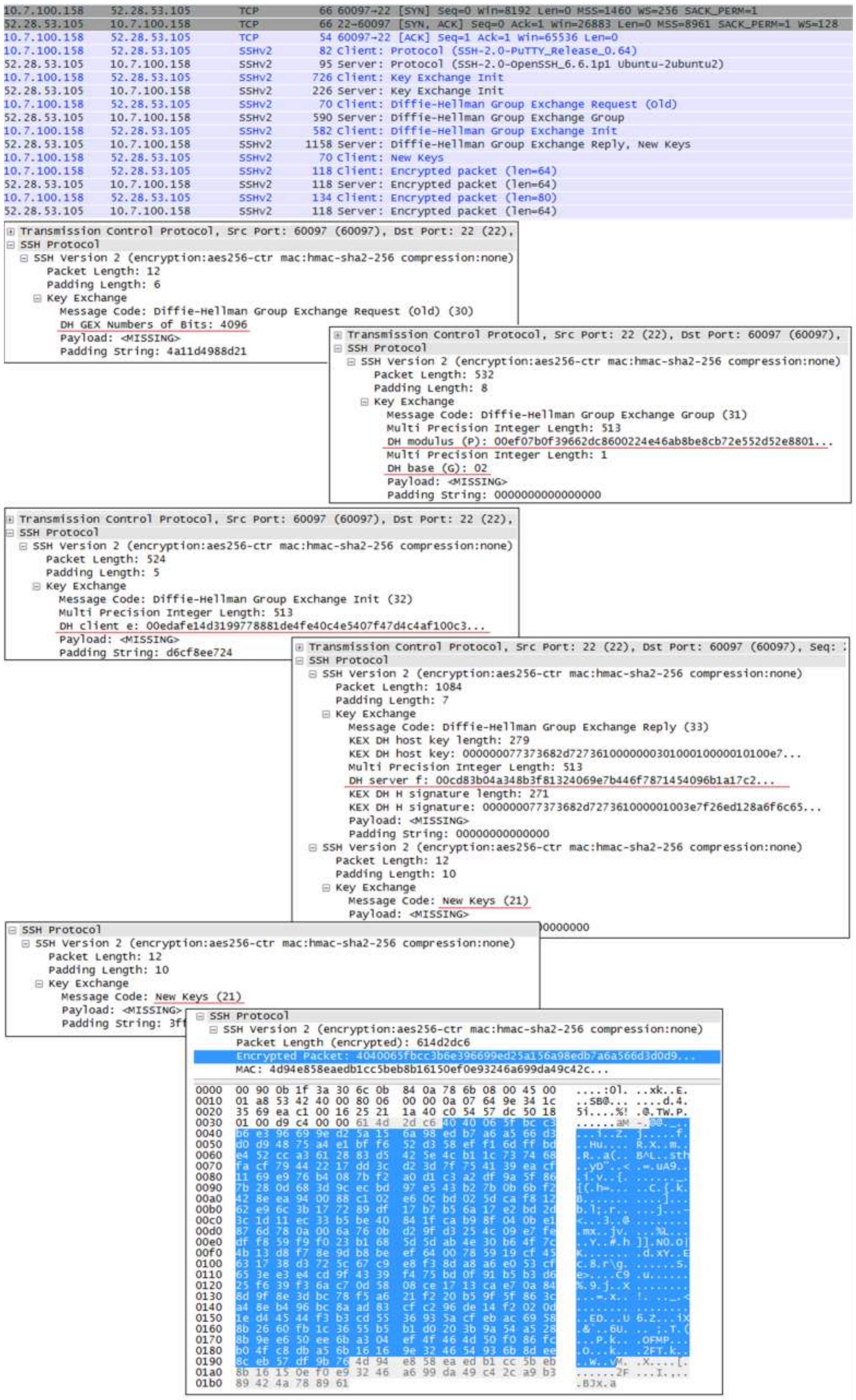
```
Tracing route to ec2-52-28-53-105.eu-central-1.compute.amazonaws.com [52.28.53.105]
over a maximum of 30 hops:

  1     1 ms     2 ms    <1 ms  10.7.100.2
  2     6 ms     4 ms     3 ms  49-18-64.66.static.logixcom.net [66.64.18.49]
  3    12 ms     7 ms     2 ms  66.196.212.126
  4     3 ms     3 ms    43 ms  6-2-22.ear1.dallas1.level3.net [4.31.142.129]
  5     *        *      123 ms  ae-3-80.ear2.frankfurt1.level3.net [4.69.154.149]
  6     *        *      123 ms  ae-3-80.ear2.frankfurt1.level3.net [4.69.154.149]
  7   125 ms   125 ms   125 ms  dialup-212.162.19.106.frankfurt1.mik.net [212.162.19.106]
  8   123 ms   123 ms   123 ms  54.239.5.86
  9   123 ms   123 ms   123 ms  54.239.5.134
 10   124 ms   124 ms   124 ms  ec2-52-28-53-105.eu-central-1.compute.amazonaws.com [52.28.53.105]
```

FIGURE 9. IP packet path from Dallas to Frankfurt.

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.



```
10.7.100.158    52.28.53.105    TCP      66 60097→22 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
52.28.53.105    10.7.100.158    TCP      66 22→60097 [SYN, ACK] Seq=0 Ack=1 Win=26883 Len=0 MSS=8961 SACK_PERM=1 WS=128
10.7.100.158    52.28.53.105    TCP      54 60097→22 [ACK] Seq=1 Ack=1 Win=65536 Len=0
10.7.100.158    52.28.53.105    SSHv2    82 Client: Protocol (SSH-2.0-PuTTY_Release_0.64)
52.28.53.105    10.7.100.158    SSHv2    95 Server: Protocol (SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2)
10.7.100.158    52.28.53.105    SSHv2   726 Client: Key Exchange Init
52.28.53.105    10.7.100.158    SSHv2   226 Server: Key Exchange Init
10.7.100.158    52.28.53.105    SSHv2    70 Client: Diffie-Hellman Group Exchange Request (Old)
52.28.53.105    10.7.100.158    SSHv2   590 Server: Diffie-Hellman Group Exchange Group
10.7.100.158    52.28.53.105    SSHv2   582 Client: Diffie-Hellman Group Exchange Init
52.28.53.105    10.7.100.158    SSHv2  1158 Server: Diffie-Hellman Group Exchange Reply, New Keys
10.7.100.158    52.28.53.105    SSHv2    70 Client: New Keys
10.7.100.158    52.28.53.105    SSHv2   118 Client: Encrypted packet (len=64)
52.28.53.105    10.7.100.158    SSHv2   118 Server: Encrypted packet (len=64)
10.7.100.158    52.28.53.105    SSHv2   134 Client: Encrypted packet (len=80)
52.28.53.105    10.7.100.158    SSHv2   118 Server: Encrypted packet (len=64)
```

```
⊞ Transmission Control Protocol, Src Port: 60097 (60097), Dst Port: 22 (22),
⊟ SSH Protocol
  ⊟ SSH Version 2 (encryption:aes256-ctr mac:hmac-sha2-256 compression:none)
      Packet Length: 12
      Padding Length: 6
    ⊟ Key Exchange
        Message Code: Diffie-Hellman Group Exchange Request (Old) (30)
        DH GEX Numbers of Bits: 4096
        Payload: <MISSING>
        Padding String: 4a11d4988d21
```

```
⊞ Transmission Control Protocol, Src Port: 22 (22), Dst Port: 60097 (60097),
⊟ SSH Protocol
  ⊟ SSH Version 2 (encryption:aes256-ctr mac:hmac-sha2-256 compression:none)
      Packet Length: 532
      Padding Length: 8
    ⊟ Key Exchange
        Message Code: Diffie-Hellman Group Exchange Group (31)
        Multi Precision Integer Length: 513
        DH modulus (P): 00ef07b0f39662dc8600224e46ab8be8cb72e552d52e8801...
        Multi Precision Integer Length: 1
        DH base (G): 02
        Payload: <MISSING>
        Padding String: 0000000000000000
```

```
⊞ Transmission Control Protocol, Src Port: 60097 (60097), Dst Port: 22 (22),
⊟ SSH Protocol
  ⊟ SSH Version 2 (encryption:aes256-ctr mac:hmac-sha2-256 compression:none)
      Packet Length: 524
      Padding Length: 5
    ⊟ Key Exchange
        Message Code: Diffie-Hellman Group Exchange Init (32)
        Multi Precision Integer Length: 513
        DH client e: 00edafe14d3199778881de4fe40c4e5407f47d4c4af100c3...
        Payload: <MISSING>
        Padding String: d6cf8ee724
```

```
⊞ Transmission Control Protocol, Src Port: 22 (22), Dst Port: 60097 (60097), Seq:
⊟ SSH Protocol
  ⊟ SSH Version 2 (encryption:aes256-ctr mac:hmac-sha2-256 compression:none)
      Packet Length: 1084
      Padding Length: 7
    ⊟ Key Exchange
        Message Code: Diffie-Hellman Group Exchange Reply (33)
        KEX DH host key length: 279
        KEX DH host key: 000000077373682d7273610000000301000100000010100e7...
        Multi Precision Integer Length: 513
        DH server f: 00cd83b04a348b3f81324069e7b446f7871454096b1a17c2...
        KEX DH H signature length: 271
        KEX DH H signature: 000000077373682d727361000001003e7f26ed128a6f6c65...
        Payload: <MISSING>
        Padding String: 00000000000000
  ⊟ SSH Version 2 (encryption:aes256-ctr mac:hmac-sha2-256 compression:none)
      Packet Length: 12
      Padding Length: 10
    ⊟ Key Exchange
        Message Code: New Keys (21)
        Payload: <MISSING>
                                     0000000
```

```
⊟ SSH Protocol
  ⊟ SSH Version 2 (encryption:aes256-ctr mac:hmac-sha2-256 compression:none)
      Packet Length: 12
      Padding Length: 10
    ⊟ Key Exchange
        Message Code: New Keys (21)
        Payload: <MISSING>
        Padding String: 3f
```

```
⊟ SSH Protocol
  ⊟ SSH Version 2 (encryption:aes256-ctr mac:hmac-sha2-256 compression:none)
      Packet Length (encrypted): 614d2dc6
      Encrypted Packet: 4040065fbcc3b6e396699ed25a156a98edb7a6a566d3d0d9...
      MAC: 4d94e858eaedb1cc5beb8b16150ef0e93246a699da49c42c...

0000  00 90 0b 1f 3a 30 6c 0b  84 0a 78 6b 08 00 45 00   ....:0l. ..xk..E.
0010  01 a8 53 42 40 00 80 06  00 00 0a 07 64 9e 34 1c   ..SB@... ....d.4.
0020  35 69 ea c1 00 16 25 21  1a 40 c0 54 57 dc 50 18   5i....%! .@.TW.P.
0030  01 00 d9 c4 00 00 61 4d  2d c6 40 40 06 5f bc c3   ......aM -.@@._..
0040  b6 e3 96 69 9e d2 5a 15  6a 98 ed b7 a6 a5 66 d3   ...i..Z. j.....f.
0050  d0 d9 48 75 a4 e1 bf f6  52 d3 58 ef f1 6d ff bd   ..Hu.... R.X..m..
0060  e4 52 cc a3 61 28 83 d5  42 5e 4c b1 1c 73 74 68   .R..a(.. B^L..sth
0070  fa cf 79 44 22 17 dd 3c  d2 3d 7f 75 41 39 ea cf   ..yD"..< .=.uA9..
0080  11 69 e9 76 b4 08 7b f2  a0 d1 c3 a2 df 9a 5f 86   .i.v..{. ......_.
0090  7b 28 0d 68 3d 9c ec bd  97 e5 43 b2 7b 0b 6b f2   {(.h=... ..C.{.k.
00a0  42 8e ea 94 00 88 c1 02  e6 0c bd 02 5d ca f8 12   B....... ....]...
00b0  62 e9 6c 3b 17 72 89 df  17 b7 b5 6a 17 e2 bd 2d   b.l;.r.. ...j...-
00c0  3c 1d 11 ec 33 b5 be 40  84 1f ca b9 8f 04 0b e1   <...3..@ ........
00d0  87 6d 78 0a 00 6a 76 0b  d2 9f d3 25 4c 09 e7 fe   .mx..jv. ...%L...
00e0  df f8 59 f9 f0 23 b1 68  5d 5d ab 4e 30 b6 4f 7c   ..Y..#.h ]].N0.O|
00f0  4b 13 d8 f7 8e 9d b8 be  ef 64 00 78 59 19 cf 45   K....... .d.xY..E
0100  63 17 38 d3 72 5c 67 c9  e8 f3 8d a8 a6 e0 53 cf   c.8.r\g. ......S.
0110  65 3e e3 e4 cd 9f 43 39  f4 75 bd 0f 91 b5 b3 d6   e>....C9 .u......
0120  25 f6 39 f3 6a c7 0d 58  08 ce 17 13 ca e7 0a 84   %.9.j..X ........
0130  8d 9f 8e 3d bc 78 f5 a6  21 f2 20 b5 9f 5f 86 3c   ...=.x.. !. .._.<
0140  a4 8e b4 96 bc 8a ad 83  cf c2 96 de 14 f2 02 0d   ........ ........
0150  1e d4 45 44 f3 b3 cd 55  36 93 5a cf eb ac 69 58   ..ED...U 6.Z...iX
0160  8b 26 60 fb 1c 36 55 b5  b1 d0 20 3b 9a 54 a5 28   .&`..6U. .. ;.T.(
0170  8b 9e e6 50 ee 6b a3 04  ef 4f 46 4d 50 f0 86 fc   ...P.k.. .OFMP...
0180  b0 4f c8 db a5 6b 16 16  9e 32 46 54 93 6b 8d ee   .O...k.. .2FT.k..
0190  8c eb 57 df 9b 76 4d 94  e8 58 ea ed b1 cc 5b eb   ..W..vM. .X....[.
01a0  8b 16 15 0e f0 e9 32 46  a6 99 da 49 c4 2c a9 b3   ......2F ...I.,..
01b0  89 42 4a 78 89 61                                  .BJx.a
```

FIGURE 10. SSH handshake from packet capture

3.2.2 HTTP Over TLS

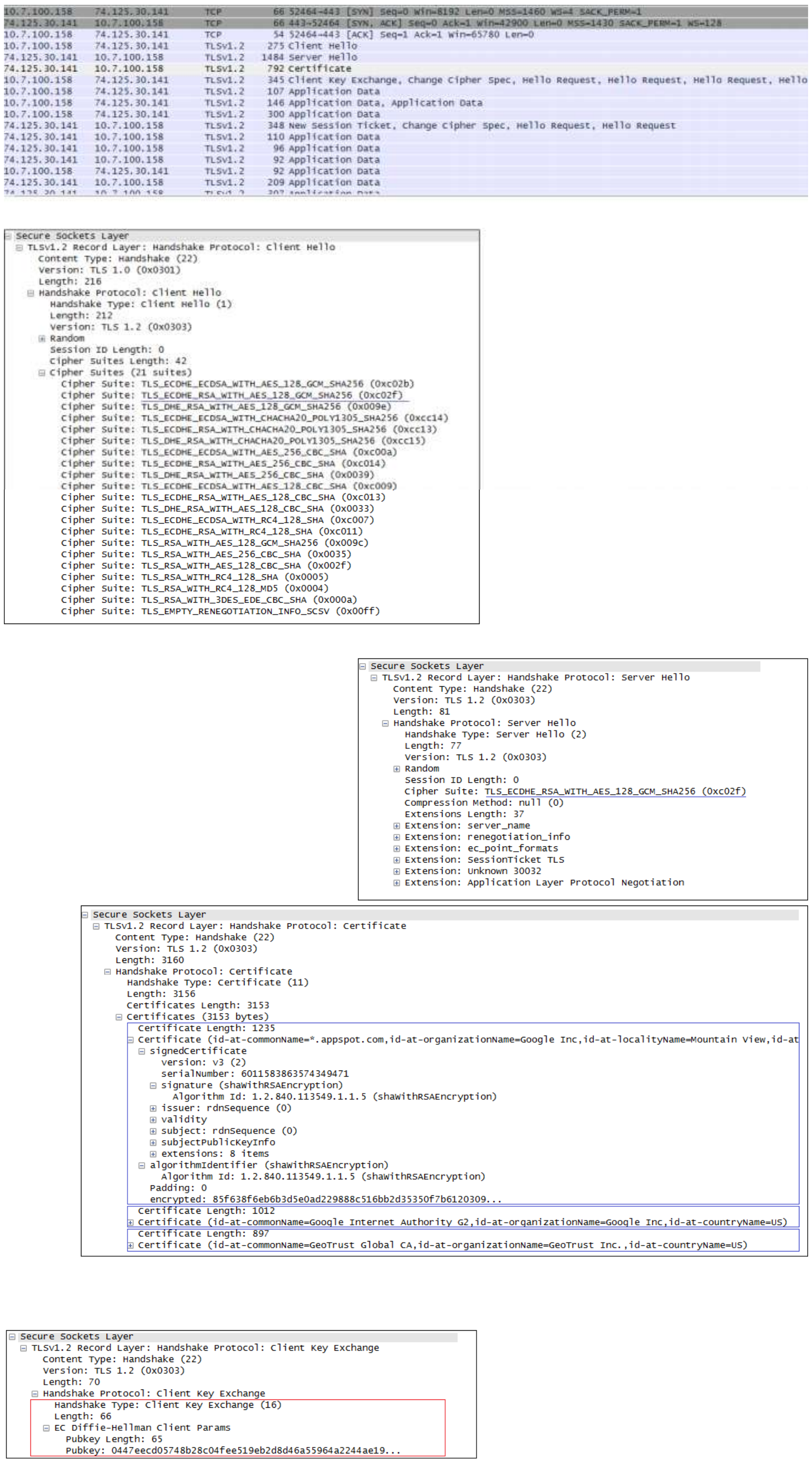
For those who like unix/linux shell SSH is an awesome tool. However, there are many people who prefer a graphical user interface, namely almost everybody and all of those who shower. Hypertext Transfer Protocol (HTTP) was developed together with HyperText Markup Language for transferring a Hypermedia 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)



```
10.7.100.158    74.125.30.141   TCP       66 52464→443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
74.125.30.141   10.7.100.158    TCP       66 443→52464 [SYN, ACK] Seq=0 Ack=1 Win=42900 Len=0 MSS=1430 SACK_PERM=1 WS=128
10.7.100.158    74.125.30.141   TCP       54 52464→443 [ACK] Seq=1 Ack=1 Win=65780 Len=0
10.7.100.158    74.125.30.141   TLSv1.2  275 Client Hello
74.125.30.141   10.7.100.158    TLSv1.2 1484 Server Hello
74.125.30.141   10.7.100.158    TLSv1.2  792 Certificate
10.7.100.158    74.125.30.141   TLSv1.2  345 Client Key Exchange, Change Cipher Spec, Hello Request, Hello Request, Hello Request, Hello
10.7.100.158    74.125.30.141   TLSv1.2  107 Application Data
10.7.100.158    74.125.30.141   TLSv1.2  146 Application Data, Application Data
10.7.100.158    74.125.30.141   TLSv1.2  300 Application Data
74.125.30.141   10.7.100.158    TLSv1.2  348 New Session Ticket, Change Cipher Spec, Hello Request, Hello Request
74.125.30.141   10.7.100.158    TLSv1.2  110 Application Data
74.125.30.141   10.7.100.158    TLSv1.2   96 Application Data
74.125.30.141   10.7.100.158    TLSv1.2   92 Application Data
10.7.100.158    74.125.30.141   TLSv1.2   92 Application Data
74.125.30.141   10.7.100.158    TLSv1.2  209 Application Data
74.125.30.141   10.7.100.158    TLSv1.2  207 Application Data
```

```
⊟ Secure Sockets Layer
  ⊟ TLSv1.2 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 216
    ⊟ Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 212
        Version: TLS 1.2 (0x0303)
      ⊞ Random
        Session ID Length: 0
        Cipher Suites Length: 42
      ⊟ Cipher Suites (21 suites)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
          Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
          Cipher Suite: TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 (0x009e)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcc14)
          Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcc13)
          Cipher Suite: TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcc15)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
          Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
          Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
          Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
          Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_RC4_128_SHA (0xc007)
          Cipher Suite: TLS_ECDHE_RSA_WITH_RC4_128_SHA (0xc011)
          Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
          Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
          Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
          Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
          Cipher Suite: TLS_RSA_WITH_RC4_128_MD5 (0x0004)
          Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
          Cipher Suite: TLS_EMPTY_RENEGOTIATION_INFO_SCSV (0x00ff)
```

```
⊟ Secure Sockets Layer
  ⊟ TLSv1.2 Record Layer: Handshake Protocol: Server Hello
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 81
    ⊟ Handshake Protocol: Server Hello
        Handshake Type: Server Hello (2)
        Length: 77
        Version: TLS 1.2 (0x0303)
      ⊞ Random
        Session ID Length: 0
        Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
        Compression Method: null (0)
        Extensions Length: 37
      ⊞ Extension: server_name
      ⊞ Extension: renegotiation_info
      ⊞ Extension: ec_point_formats
      ⊞ Extension: SessionTicket TLS
      ⊞ Extension: Unknown 30032
      ⊞ Extension: Application Layer Protocol Negotiation
```

```
⊟ Secure Sockets Layer
  ⊟ TLSv1.2 Record Layer: Handshake Protocol: Certificate
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 3160
    ⊟ Handshake Protocol: Certificate
        Handshake Type: Certificate (11)
        Length: 3156
        Certificates Length: 3153
      ⊟ Certificates (3153 bytes)
          Certificate Length: 1235
        ⊟ Certificate (id-at-commonName=*.appspot.com,id-at-organizationName=Google Inc,id-at-localityName=Mountain View,id-at
          ⊟ signedCertificate
              version: v3 (2)
              serialNumber: 6011583863574349471
            ⊟ signature (shaWithRSAEncryption)
                Algorithm Id: 1.2.840.113549.1.1.5 (shaWithRSAEncryption)
            ⊞ issuer: rdnSequence (0)
            ⊞ validity
            ⊞ subject: rdnSequence (0)
            ⊞ subjectPublicKeyInfo
            ⊞ extensions: 8 items
          ⊟ algorithmIdentifier (shaWithRSAEncryption)
              Algorithm Id: 1.2.840.113549.1.1.5 (shaWithRSAEncryption)
            Padding: 0
            encrypted: 85f638f6eb6b3d5e0ad229888c516bb2d35350f7b6120309...
          Certificate Length: 1012
        ⊞ Certificate (id-at-commonName=Google Internet Authority G2,id-at-organizationName=Google Inc,id-at-countryName=US)
          Certificate Length: 897
        ⊞ Certificate (id-at-commonName=GeoTrust Global CA,id-at-organizationName=GeoTrust Inc.,id-at-countryName=US)
```

```
⊟ Secure Sockets Layer
  ⊟ TLSv1.2 Record Layer: Handshake Protocol: Client Key Exchange
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 70
    ⊟ Handshake Protocol: Client Key Exchange
        Handshake Type: Client Key Exchange (16)
        Length: 66
      ⊟ EC Diffie-Hellman Client Params
          Pubkey Length: 65
          Pubkey: 0447eecd05748b28c04fee519eb2d8d46a55964a2244ae19...
```

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.

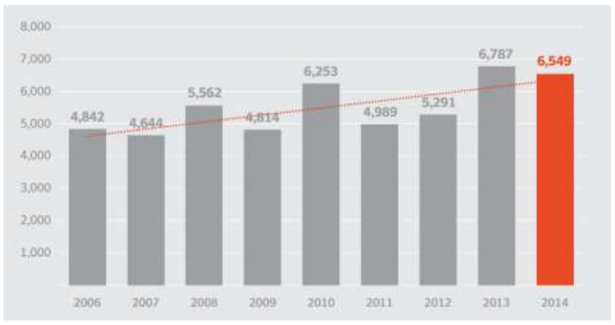


FIGURE 12. Total number of vulnerabilities between 2006 and 2014 (Symantec, 2015).

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?

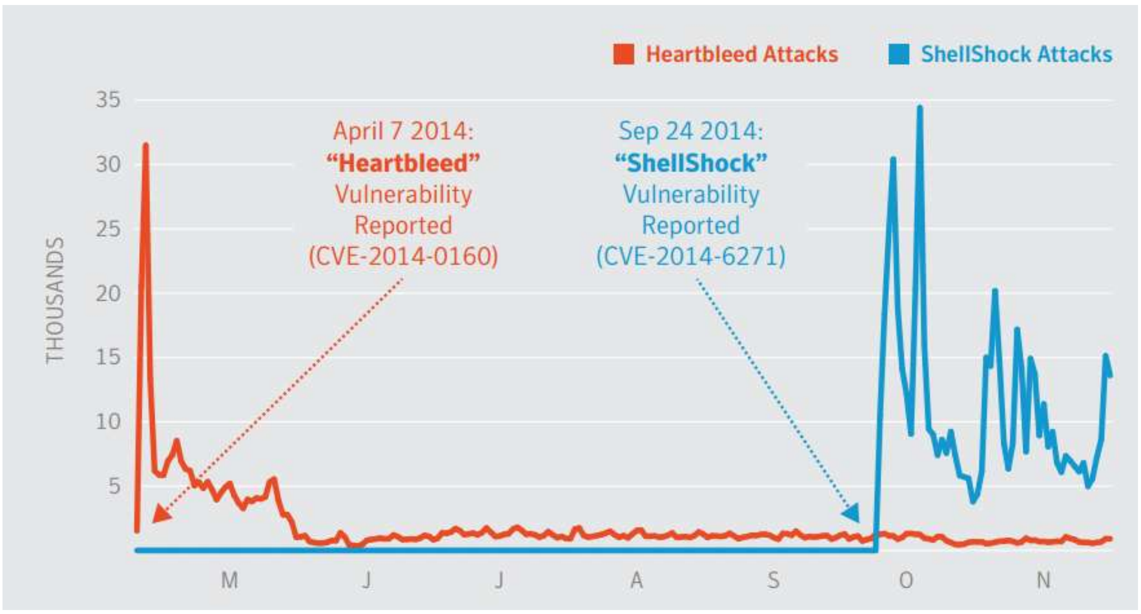
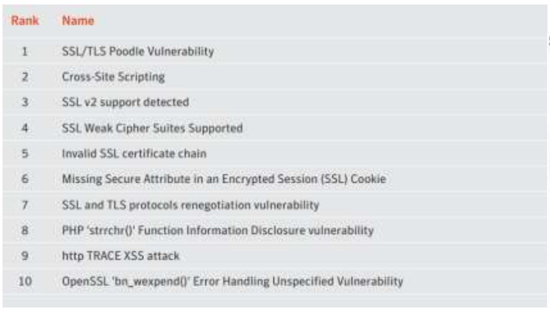


FIGURE 13. Heartbleed and ShellShock attacks between April and November 2014 (Symantec, 2015).

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).



Rank	Name
1	SSL/TLS Poodle Vulnerability
2	Cross-Site Scripting
3	SSL v2 support detected
4	SSL Weak Cipher Suites Supported
5	Invalid SSL certificate chain
6	Missing Secure Attribute in an Encrypted Session (SSL) Cookie
7	SSL and TLS protocols renegotiation vulnerability
8	PHP 'strrchr()' Function Information Disclosure vulnerability
9	http TRACE XSS attack
10	OpenSSL 'bn_wexpend()' Error Handling Unspecified Vulnerability

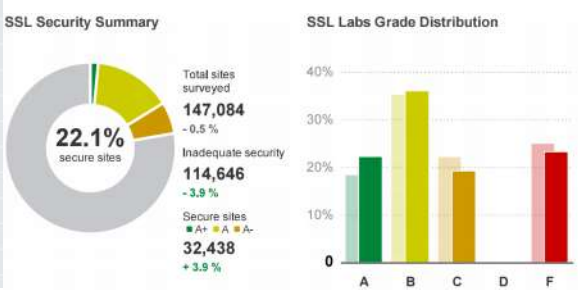


FIGURE 14. Top 10 vulnerabilities found unpatched on scanned web servers (Symantec, 2015) and summary for SSL Pulse summary (TIM, 2015).

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.

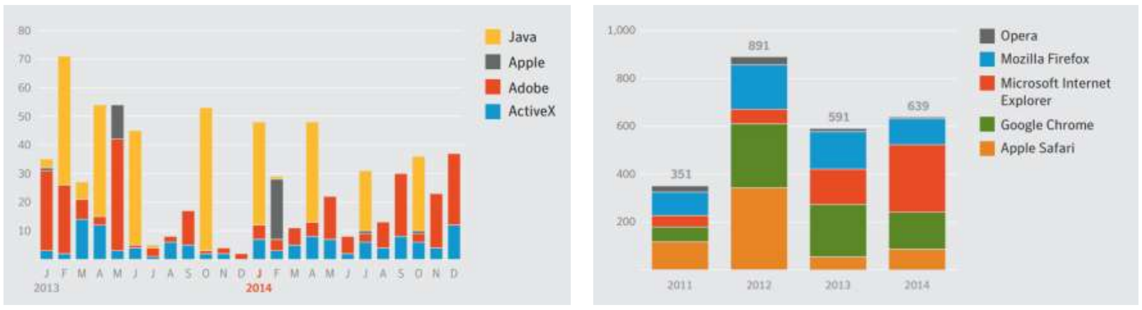


FIGURE 15. Plugin vulnerabilities by month and web browser vulnerabilities by year (Symantec, 2015).

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 a replace question mark.

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)

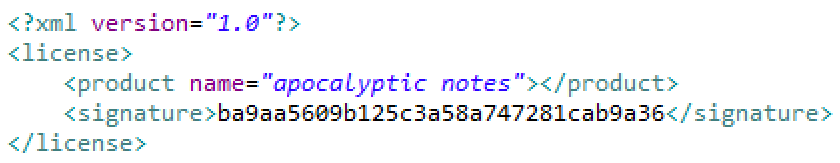
There are two methods the application can use to determine whether the usage is authorized or not. It can ask from somebody it trusts and like a child unknowing what to do, it will phone home. An application has one or more hardcoded URLs for connecting to a server which will accept or decline the usage.

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
<?xml version="1.0"?>
<license>
    <product name="apocalyptic notes"></product>
    <signature>ba9aa5609b125c3a58a747281cab9a36</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 straight forward 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.

```xml
<?xml version="1.0"?>
<license>
    <product name="apocalyptic notes">
        <device imei="990004904105474" />
    </product>
    <signature>854b43af233c2ad7a825f47b87c5820b</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.

```xml
<?xml version="1.0"?>
<license>
    <product name="apocalyptic notes">
        <User name="Acme Inc" contact="james.potkukelkka@achme.com" />
    </product>
    <signature>3d5718cb2f5d528e1a2a3f993dc3d691</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)

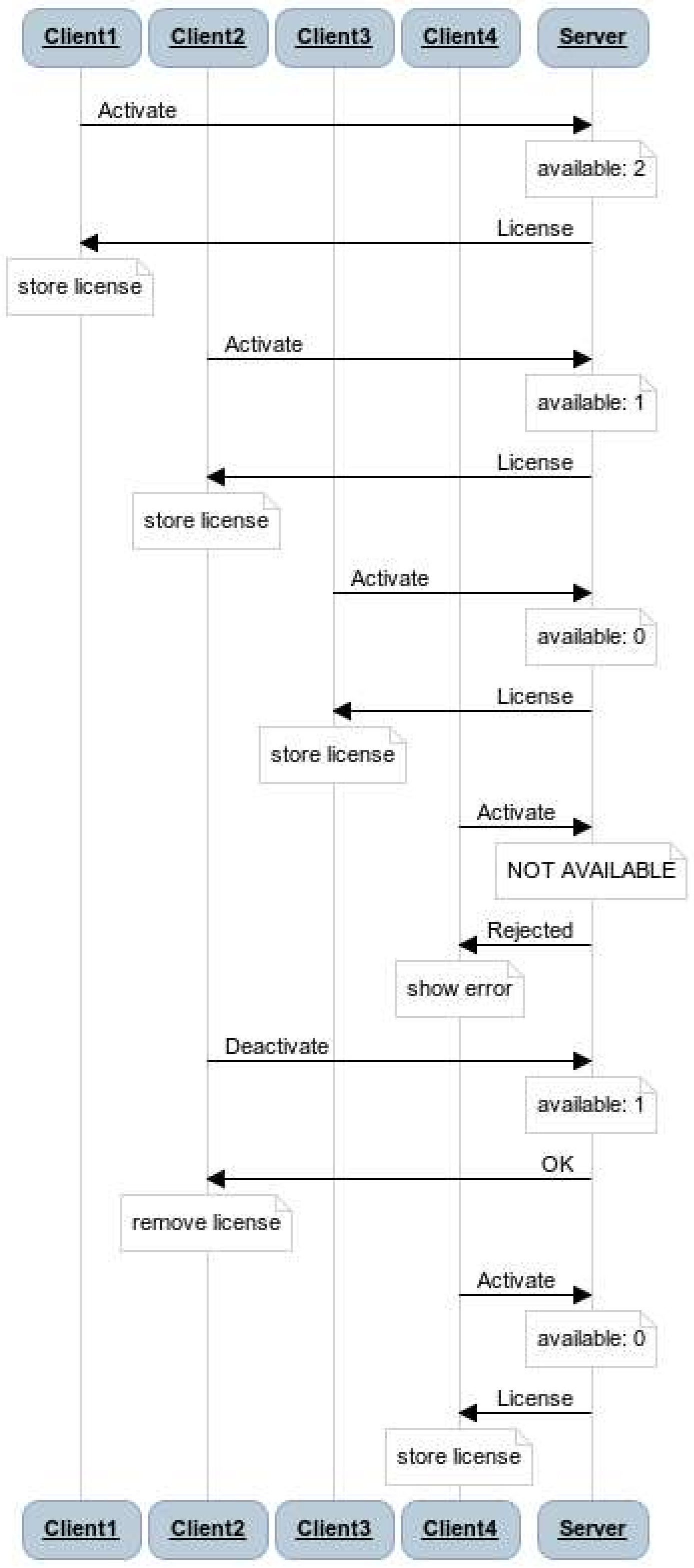


DIAGRAM 3. Simplified illustration of Ultraedit Product Activation

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.potkukelkka@achme.com" />
        <option>SPEECH_RECOGNITION</option>
    </product>
    <signature>deeb8ecf2c2dcdba4fc5f0fb9d7a18d6</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.potkukelkka@achme.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:

- Device ID
- Contact information
- Time
- Region (Location)
- Credentials
- Executable
- Cryptographic keys

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.

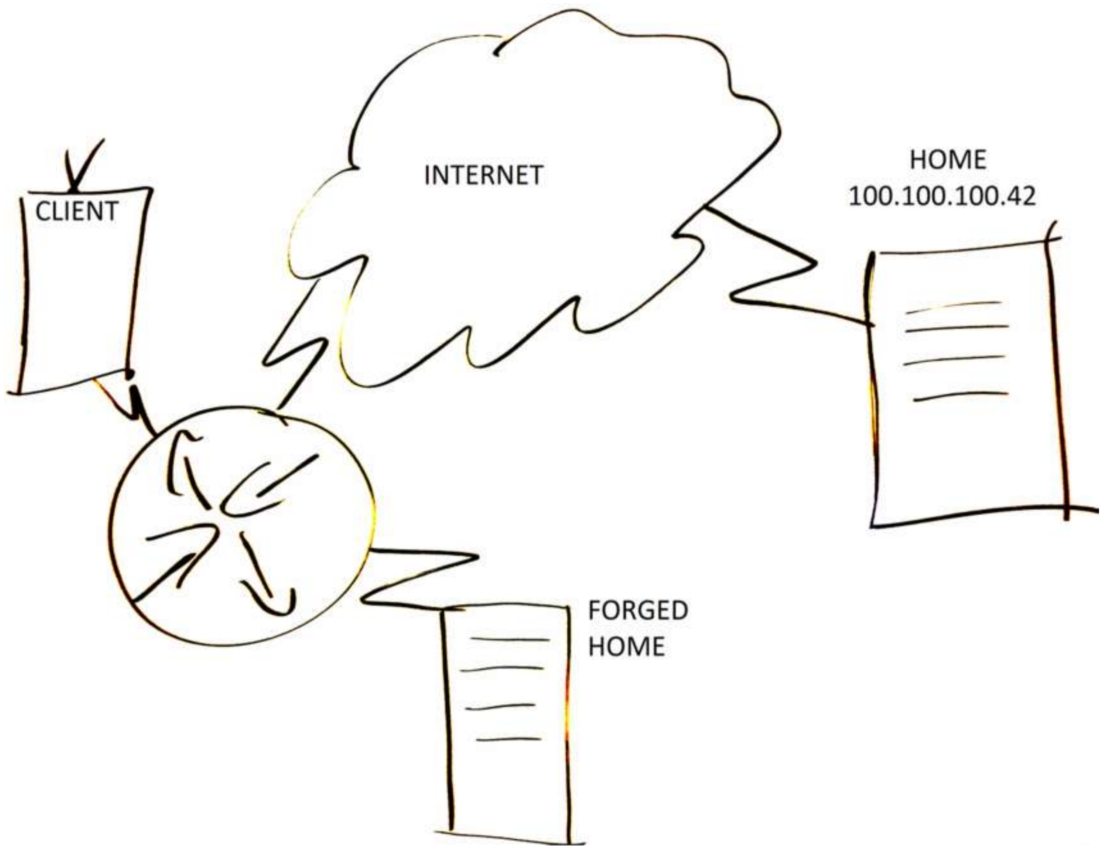


FIGURE 22. Hostile network.

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.

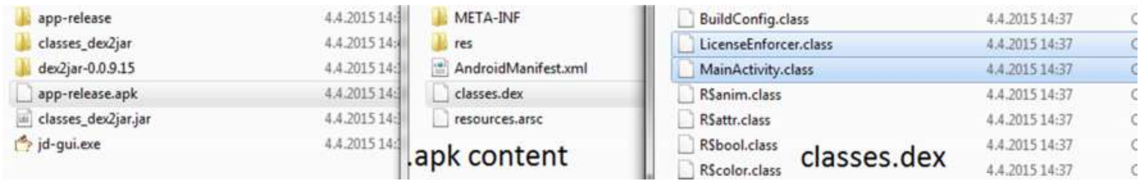
Executable and your code are not safe from attacks either. How much effort and skill is required for tampering with executable is highly platform dependent. Java applications are especially vulnerable to reverse engineering as Java applications are compiled into a platform agnostic bytecode. Applications, 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.

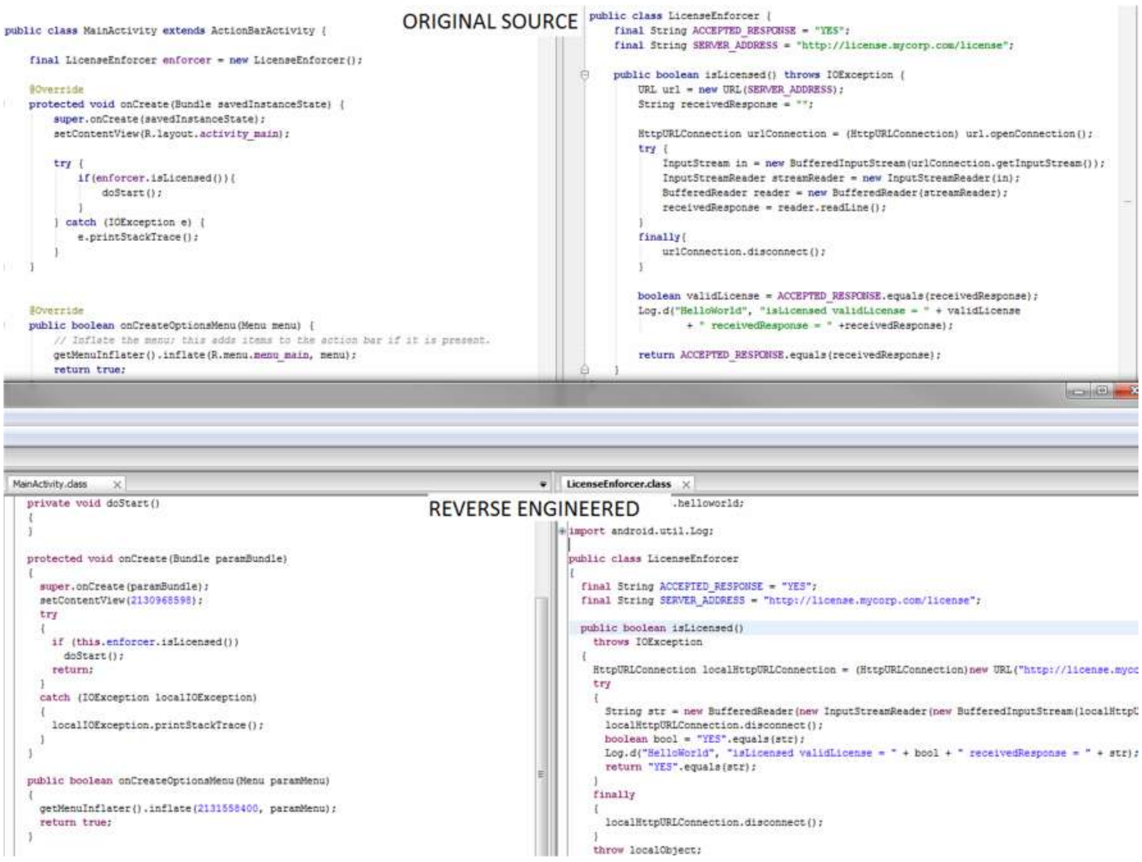
4.2 Case study: Reverse Engineering Android APK

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.



```
app-release            4.4.2015 14:      META-INF               BuildConfig.class      4.4.2015 14:37
classes_dex2jar        4.4.2015 14:      res                    LicenseEnforcer.class  4.4.2015 14:37
dex2jar-0.0.9.15       4.4.2015 14:      AndroidManifest.xml    MainActivity.class     4.4.2015 14:37
app-release.apk        4.4.2015 14:      classes.dex            R$anim.class           4.4.2015 14:37
classes_dex2jar.jar    4.4.2015 14:      resources.arsc         R$attr.class           4.4.2015 14:37
jd-gui.exe             4.4.2015 14:                             R$bool.class           4.4.2015 14:37
                                                                R$color.class          4.4.2015 14:37
```

FIGURE 23. *Path to class files in apk.*



```java
public class MainActivity extends ActionBarActivity {

    final LicenseEnforcer enforcer = new LicenseEnforcer();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        try {
            if(enforcer.isLicensed()){
                doStart();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }


    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
```

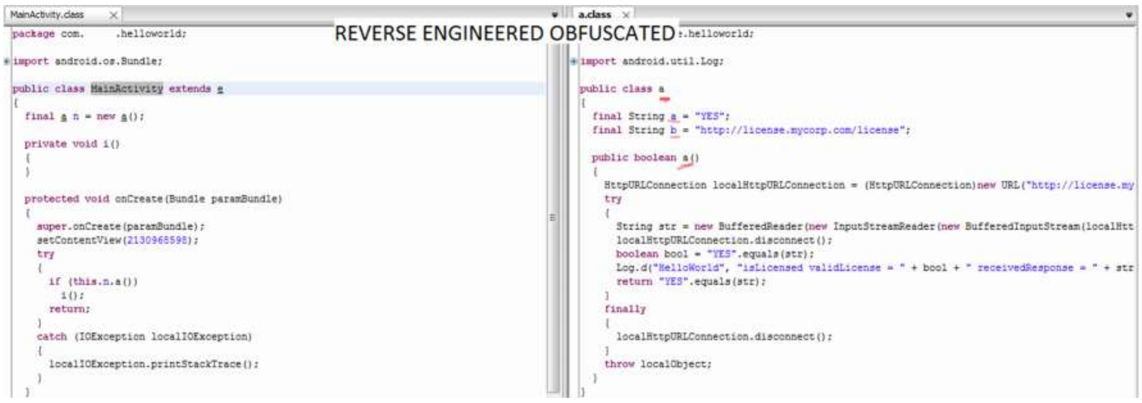
```java
public class LicenseEnforcer {
    final String ACCEPTED_RESPONSE = "YES";
    final String SERVER_ADDRESS = "http://license.mycorp.com/license";

    public boolean isLicensed() throws IOException {
        URL url = new URL(SERVER_ADDRESS);
        String receivedResponse = "";

        HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
        try {
            InputStream in = new BufferedInputStream(urlConnection.getInputStream());
            InputStreamReader streamReader = new InputStreamReader(in);
            BufferedReader reader = new BufferedReader(streamReader);
            receivedResponse = reader.readLine();
        }
        finally{
            urlConnection.disconnect();
        }

        boolean validLicense = ACCEPTED_RESPONSE.equals(receivedResponse);
        Log.d("HelloWorld", "isLicensed validLicense = " + validLicense
                + " receivedResponse = " +receivedResponse);

        return ACCEPTED_RESPONSE.equals(receivedResponse);
    }
```

```java
private void doStart()
{
}

protected void onCreate(Bundle paramBundle)
{
  super.onCreate(paramBundle);
  setContentView(2130968598);
  try
  {
    if (this.enforcer.isLicensed())
      doStart();
    return;
  }
  catch (IOException localIOException)
  {
    localIOException.printStackTrace();
  }
}

public boolean onCreateOptionsMenu(Menu paramMenu)
{
  getMenuInflater().inflate(2131558400, paramMenu);
  return true;
}
```

```java
.helloworld;

import android.util.Log;

public class LicenseEnforcer
{
  final String ACCEPTED_RESPONSE = "YES";
  final String SERVER_ADDRESS = "http://license.mycorp.com/license";

  public boolean isLicensed()
    throws IOException
  {
    HttpURLConnection localHttpURLConnection = (HttpURLConnection)new URL("http://license.mycc
    try
    {
      String str = new BufferedReader(new InputStreamReader(new BufferedInputStream(localHttpU
      localHttpURLConnection.disconnect();
      boolean bool = "YES".equals(str);
      Log.d("HelloWorld", "isLicensed validLicense = " + bool + " receivedResponse = " + str);
      return "YES".equals(str);
    }
    finally
    {
      localHttpURLConnection.disconnect();
    }
    throw localObject;
```

FIGURE 24. *Original and reverse engineered code compared.*

Android development tools a 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.

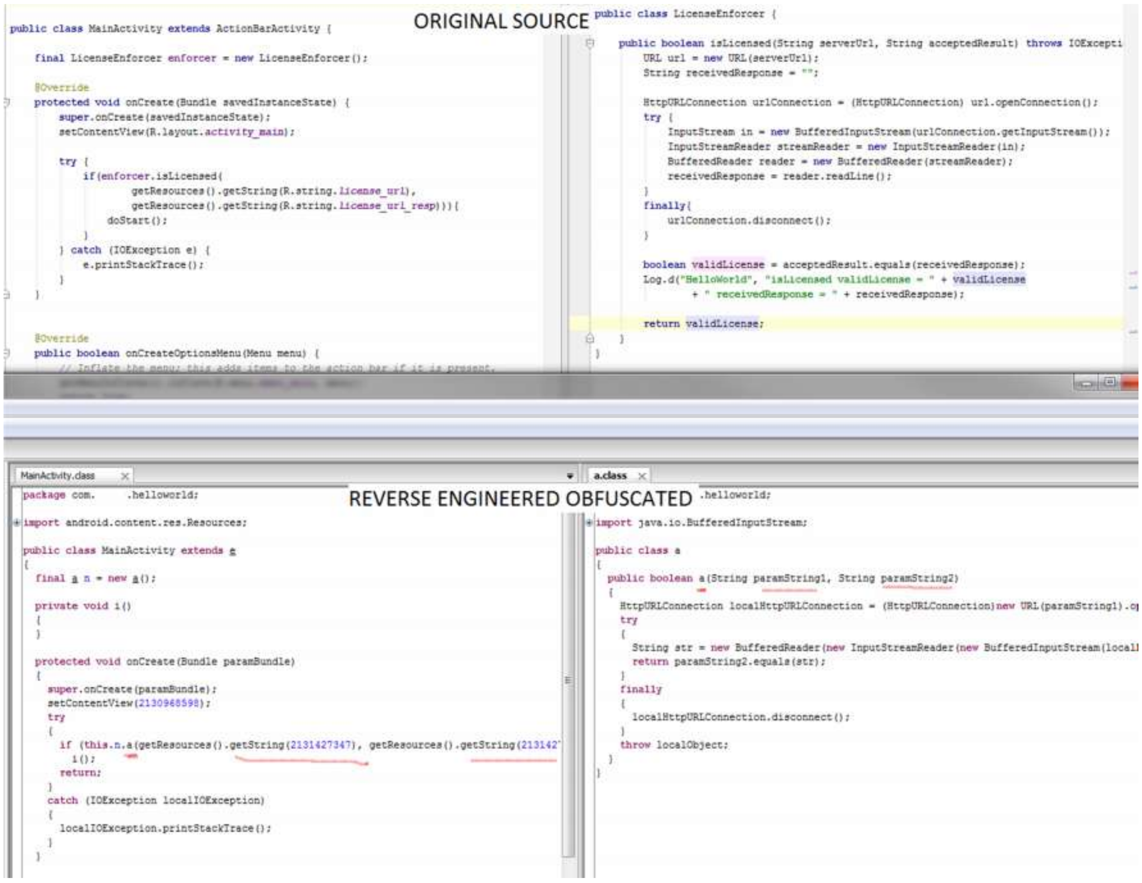


```
package com.     .helloworld;

import android.os.Bundle;

public class MainActivity extends e
{
  final a n = new a();

  private void i()
  {
  }

  protected void onCreate(Bundle paramBundle)
  {
    super.onCreate(paramBundle);
    setContentView(2130968598);
    try
    {
      if (this.n.a())
        i();
      return;
    }
    catch (IOException localIOException)
    {
      localIOException.printStackTrace();
    }
  }
```

```
:.helloworld;

import android.util.Log;

public class a
{
  final String a = "YES";
  final String b = "http://license.mycorp.com/license";

  public boolean a()
  {
    HttpURLConnection localHttpURLConnection = (HttpURLConnection)new URL("http://license.my
    try
    {
      String str = new BufferedReader(new InputStreamReader(new BufferedInputStream(localHtt
      localHttpURLConnection.disconnect();
      boolean bool = "YES".equals(str);
      Log.d("HelloWorld", "isLicensed validLicense = " + bool + " receivedResponse = " + str
      return "YES".equals(str);
    }
    finally
    {
      localHttpURLConnection.disconnect();
    }
    throw localObject;
  }
}
```

FIGURE 25. Obfuscated code reverse engineered.

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.



```java
public class MainActivity extends ActionBarActivity {

    final LicenseEnforcer enforcer = new LicenseEnforcer();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        try {
            if(enforcer.isLicensed(
                    getResources().getString(R.string.license_url),
                    getResources().getString(R.string.license_url_resp))){
                doStart();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }


    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
```

```java
public class LicenseEnforcer {

    public boolean isLicensed(String serverUrl, String acceptedResult) throws IOExcepti
        URL url = new URL(serverUrl);
        String receivedResponse = "";

        HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
        try {
            InputStream in = new BufferedInputStream(urlConnection.getInputStream());
            InputStreamReader streamReader = new InputStreamReader(in);
            BufferedReader reader = new BufferedReader(streamReader);
            receivedResponse = reader.readLine();
        }
        finally{
            urlConnection.disconnect();
        }

        boolean validLicense = acceptedResult.equals(receivedResponse);
        Log.d("HelloWorld", "isLicensed validLicense = " + validLicense
                + " receivedResponse = " + receivedResponse);

        return validLicense;
    }
}
```

```java
package com.     .helloworld;

import android.content.res.Resources;

public class MainActivity extends e
{
  final a n = new a();

  private void i()
  {
  }

  protected void onCreate(Bundle paramBundle)
  {
    super.onCreate(paramBundle);
    setContentView(2130968598);
    try
    {
      if (this.n.a(getResources().getString(2131427347), getResources().getString(213142
        i();
      return;
    }
    catch (IOException localIOException)
    {
      localIOException.printStackTrace();
    }
  }
```

```java
.helloworld;

import java.io.BufferedInputStream;

public class a
{
  public boolean a(String paramString1, String paramString2)
  {
    HttpURLConnection localHttpURLConnection = (HttpURLConnection)new URL(paramString1).o
    try
    {
      String str = new BufferedReader(new InputStreamReader(new BufferedInputStream(local
      return paramString2.equals(str);
    }
    finally
    {
      localHttpURLConnection.disconnect();
    }
    throw localObject;
  }
}
```

FIGURES 26. Original and reverse engineered code compared with text resources.

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 programmatically.

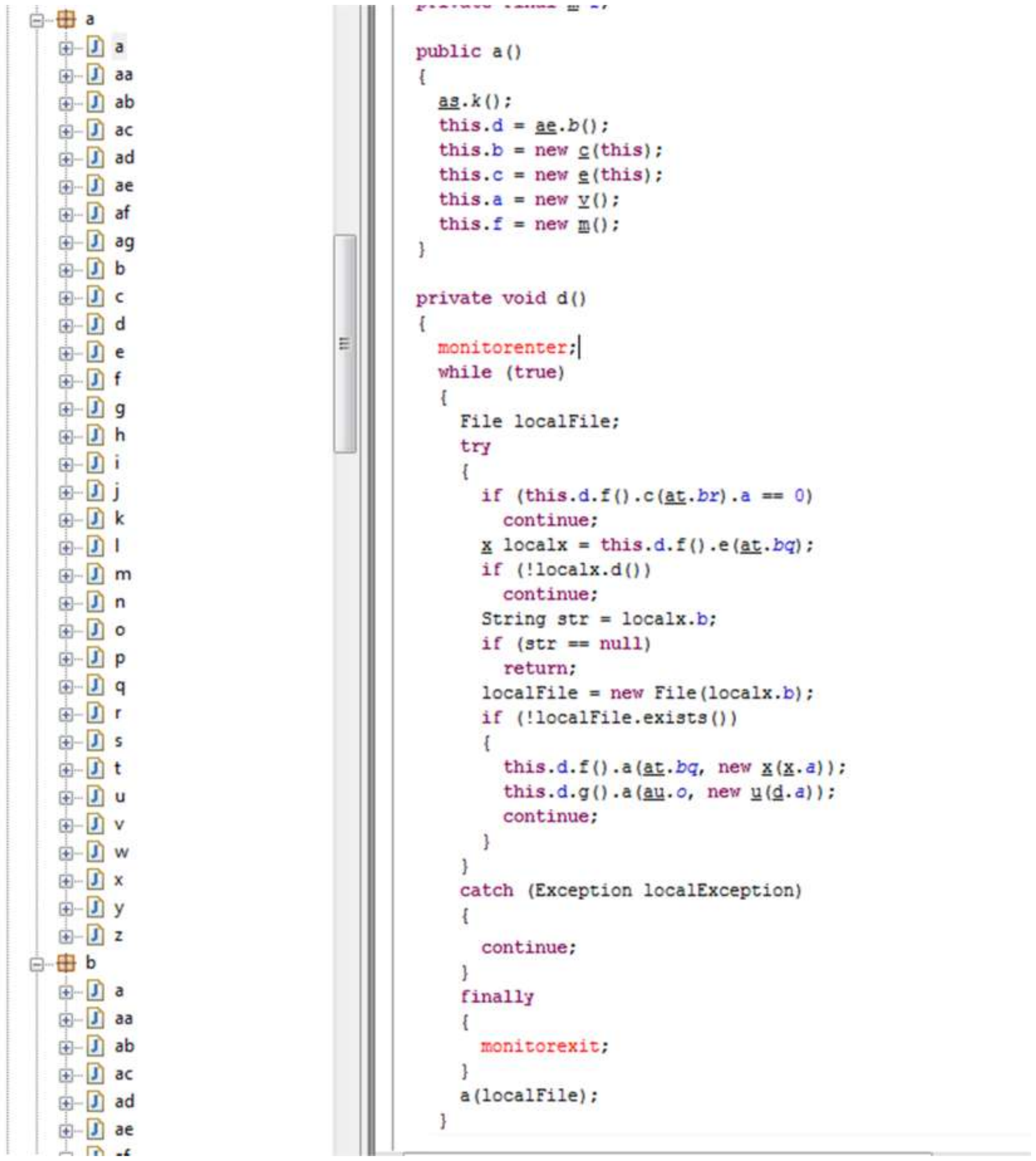
```java
PackageManager packetManager = getPackageManager();
PackageInfo packetInfo = packetManager.getPackageInfo(getPackageName(),
        PackageManager.GET_SIGNATURES);
Signature[] signatures = packetInfo.signatures;
```

FIGURE 27. Acquiring application signature.

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.



```
public a()
{
  as.k();
  this.d = ae.b();
  this.b = new c(this);
  this.c = new e(this);
  this.a = new v();
  this.f = new m();
}

private void d()
{
  monitorenter;
  while (true)
  {
    File localFile;
    try
    {
      if (this.d.f().c(at.br).a == 0)
        continue;
      x localx = this.d.f().e(at.bq);
      if (!localx.d())
        continue;
      String str = localx.b;
      if (str == null)
        return;
      localFile = new File(localx.b);
      if (!localFile.exists())
      {
        this.d.f().a(at.bq, new x(x.a));
        this.d.g().a(au.o, new u(d.a));
        continue;
      }
    }
    catch (Exception localException)
    {
      continue;
    }
    finally
    {
      monitorexit;
    }
    a(localFile);
  }
```

FIGURE 28. Reverse engineered large project.

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.

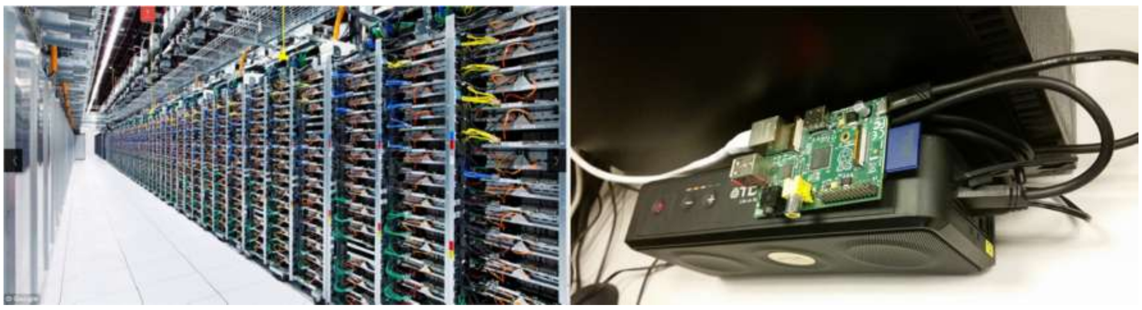


FIGURE 29. Google data center vs Raspberry Pi

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)

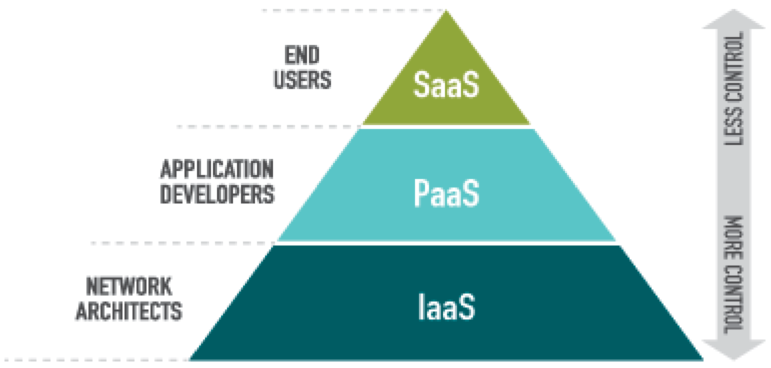
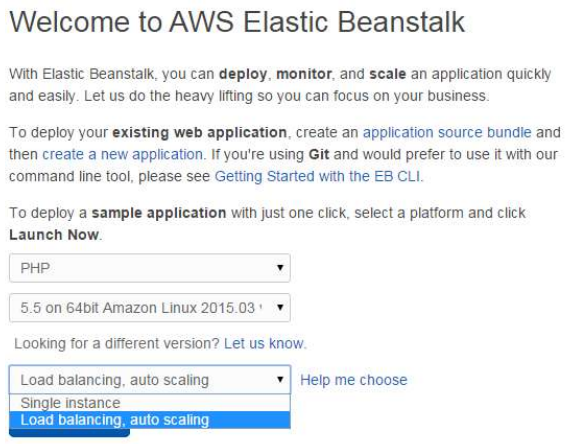


FIGURE 30. Different levels of cloud (Redcentric, 2015).

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.



Welcome to AWS Elastic Beanstalk

With Elastic Beanstalk, you can **deploy**, **monitor**, and **scale** an application quickly and easily. Let us do the heavy lifting so you can focus on your business.

To deploy your **existing web application**, create an application source bundle and then create a new application. If you're using **Git** and would prefer to use it with our command line tool, please see Getting Started with the EB CLI.

To deploy a **sample application** with just one click, select a platform and click **Launch Now**.

PHP

5.5 on 64bit Amazon Linux 2015.03

Looking for a different version? Let us know.

Load balancing, auto scaling — Help me choose

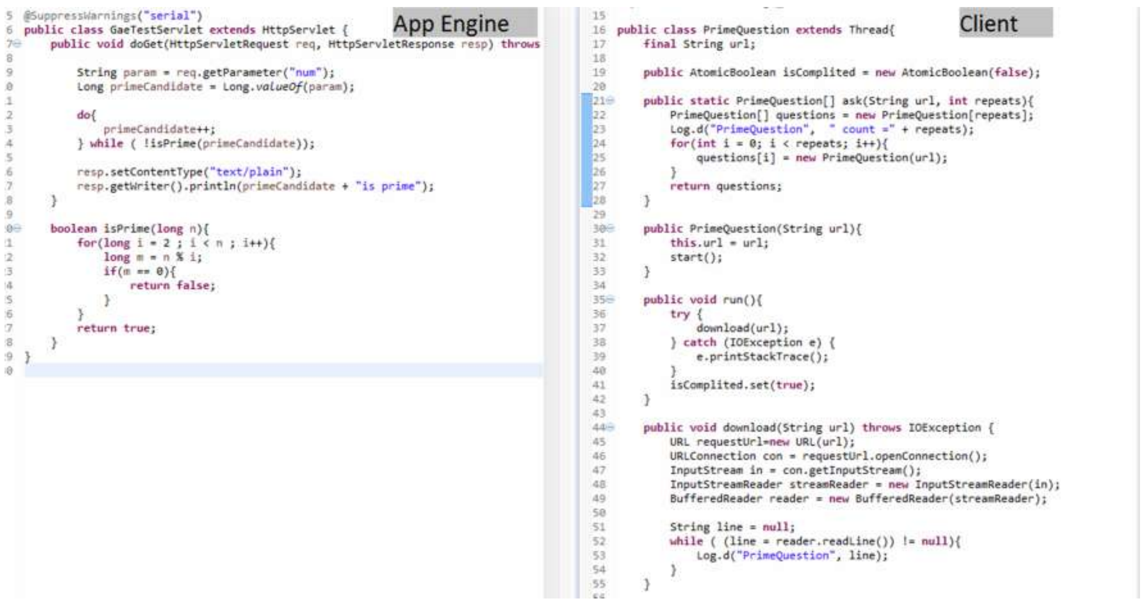
Single instance

Load balancing, auto scaling

FIGURE 31. AWS Elastic Beanstalk configuration

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.



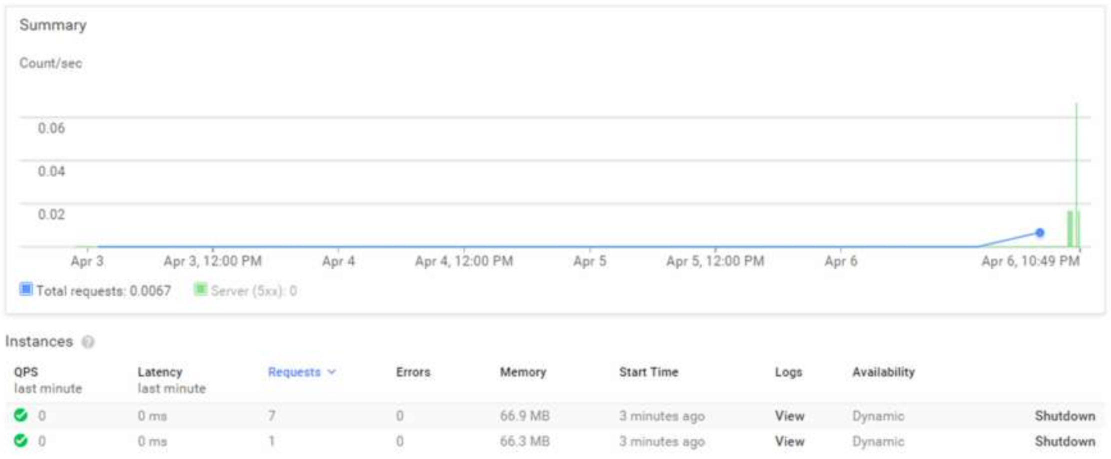
```java
@SuppressWarnings("serial")
public class GaeTestServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws

        String param = req.getParameter("num");
        Long primeCandidate = Long.valueOf(param);

        do{
            primeCandidate++;
        } while ( !isPrime(primeCandidate));

        resp.setContentType("text/plain");
        resp.getWriter().println(primeCandidate + "is prime");
    }

    boolean isPrime(long n){
        for(long i = 2 ; i < n ; i++){
            long m = n % i;
            if(m == 0){
                return false;
            }
        }
        return true;
    }
}
```

```java

public class PrimeQuestion extends Thread{
    final String url;

    public AtomicBoolean isComplited = new AtomicBoolean(false);

    public static PrimeQuestion[] ask(String url, int repeats){
        PrimeQuestion[] questions = new PrimeQuestion[repeats];
        Log.d("PrimeQuestion",  " count =" + repeats);
        for(int i = 0; i < repeats; i++){
            questions[i] = new PrimeQuestion(url);
        }
        return questions;
    }

    public PrimeQuestion(String url){
        this.url = url;
        start();
    }

    public void run(){
        try {
            download(url);
        } catch (IOException e) {
            e.printStackTrace();
        }
        isComplited.set(true);
    }

    public void download(String url) throws IOException {
        URL requestUrl=new URL(url);
        URLConnection con = requestUrl.openConnection();
        InputStream in = con.getInputStream();
        InputStreamReader streamReader = new InputStreamReader(in);
        BufferedReader reader = new BufferedReader(streamReader);

        String line = null;
        while ( (line = reader.readLine()) != null){
            Log.d("PrimeQuestion", line);
        }
    }
```

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.

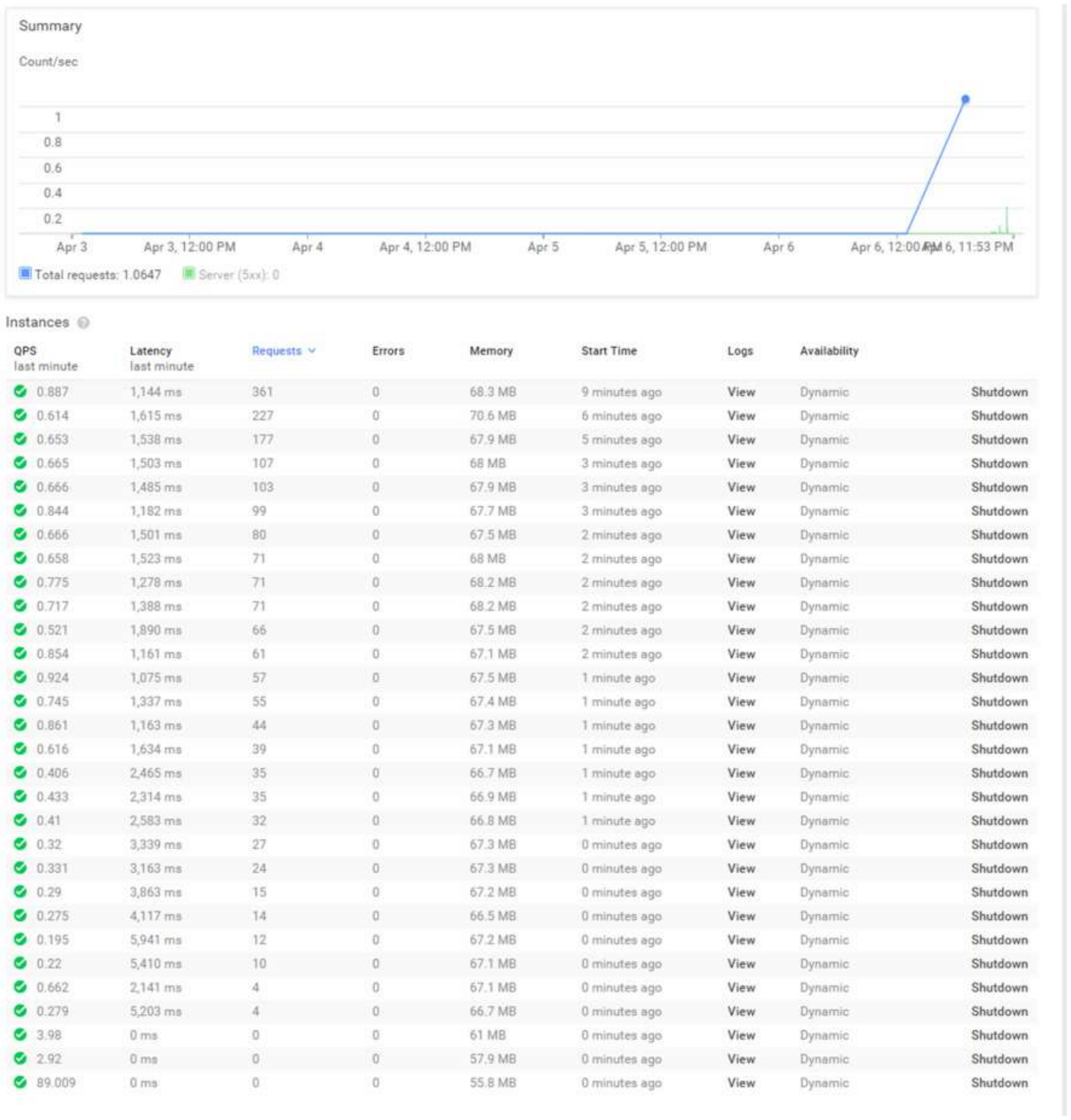
At the second step I selected a smaller starting number for the prime number search and started sending requests sending 100 requests simultaneously. In addition, I configured a timer which would resend all requests that had been completed. Usually, my client got the response it was looking for in a 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.



QPS last minute	Latency last minute	Requests	Errors	Memory	Start Time	Logs	Availability	
✓ 0	0 ms	7	0	66.9 MB	3 minutes ago	View	Dynamic	Shutdown
✓ 0	0 ms	1	0	66.3 MB	3 minutes ago	View	Dynamic	Shutdown

FIGURE 33. Google App Engine Dashboard after few CPU intensive requests.

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.

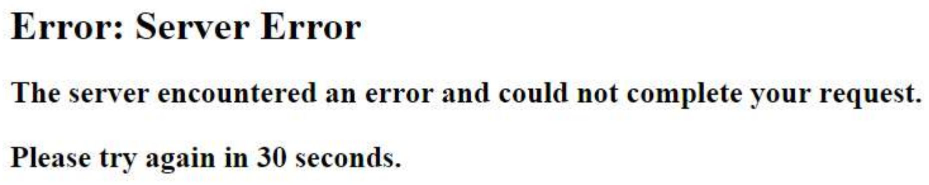


QPS last minute	Latency last minute	Requests	Errors	Memory	Start Time	Logs	Availability	
✓ 0.887	1,144 ms	361	0	68.3 MB	9 minutes ago	View	Dynamic	Shutdown
✓ 0.614	1,615 ms	227	0	70.6 MB	6 minutes ago	View	Dynamic	Shutdown
✓ 0.653	1,538 ms	177	0	67.9 MB	5 minutes ago	View	Dynamic	Shutdown
✓ 0.665	1,503 ms	107	0	68 MB	3 minutes ago	View	Dynamic	Shutdown
✓ 0.666	1,485 ms	103	0	67.9 MB	3 minutes ago	View	Dynamic	Shutdown
✓ 0.844	1,182 ms	99	0	67.7 MB	3 minutes ago	View	Dynamic	Shutdown
✓ 0.666	1,501 ms	80	0	67.5 MB	2 minutes ago	View	Dynamic	Shutdown
✓ 0.658	1,523 ms	71	0	68 MB	2 minutes ago	View	Dynamic	Shutdown
✓ 0.775	1,278 ms	71	0	68.2 MB	2 minutes ago	View	Dynamic	Shutdown
✓ 0.717	1,388 ms	71	0	68.2 MB	2 minutes ago	View	Dynamic	Shutdown
✓ 0.521	1,890 ms	66	0	67.5 MB	2 minutes ago	View	Dynamic	Shutdown
✓ 0.854	1,161 ms	61	0	67.1 MB	2 minutes ago	View	Dynamic	Shutdown
✓ 0.924	1,075 ms	57	0	67.5 MB	1 minute ago	View	Dynamic	Shutdown
✓ 0.745	1,337 ms	55	0	67.4 MB	1 minute ago	View	Dynamic	Shutdown
✓ 0.861	1,163 ms	44	0	67.3 MB	1 minute ago	View	Dynamic	Shutdown
✓ 0.616	1,634 ms	39	0	67.1 MB	1 minute ago	View	Dynamic	Shutdown
✓ 0.406	2,465 ms	35	0	66.7 MB	1 minute ago	View	Dynamic	Shutdown
✓ 0.433	2,314 ms	35	0	66.9 MB	1 minute ago	View	Dynamic	Shutdown
✓ 0.41	2,583 ms	32	0	66.8 MB	1 minute ago	View	Dynamic	Shutdown
✓ 0.32	3,339 ms	27	0	67.3 MB	0 minutes ago	View	Dynamic	Shutdown
✓ 0.331	3,163 ms	24	0	67.3 MB	0 minutes ago	View	Dynamic	Shutdown
✓ 0.29	3,863 ms	15	0	67.2 MB	0 minutes ago	View	Dynamic	Shutdown
✓ 0.275	4,117 ms	14	0	66.5 MB	0 minutes ago	View	Dynamic	Shutdown
✓ 0.195	5,941 ms	12	0	67.2 MB	0 minutes ago	View	Dynamic	Shutdown
✓ 0.22	5,410 ms	10	0	67.1 MB	0 minutes ago	View	Dynamic	Shutdown
✓ 0.662	2,141 ms	4	0	67.1 MB	0 minutes ago	View	Dynamic	Shutdown
✓ 0.279	5,203 ms	4	0	66.7 MB	0 minutes ago	View	Dynamic	Shutdown
✓ 3.98	0 ms	0	0	61 MB	0 minutes ago	View	Dynamic	Shutdown
✓ 2.92	0 ms	0	0	57.9 MB	0 minutes ago	View	Dynamic	Shutdown
✓ 89.009	0 ms	0	0	55.8 MB	0 minutes ago	View	Dynamic	Shutdown

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.

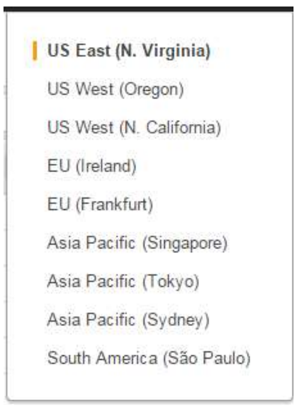
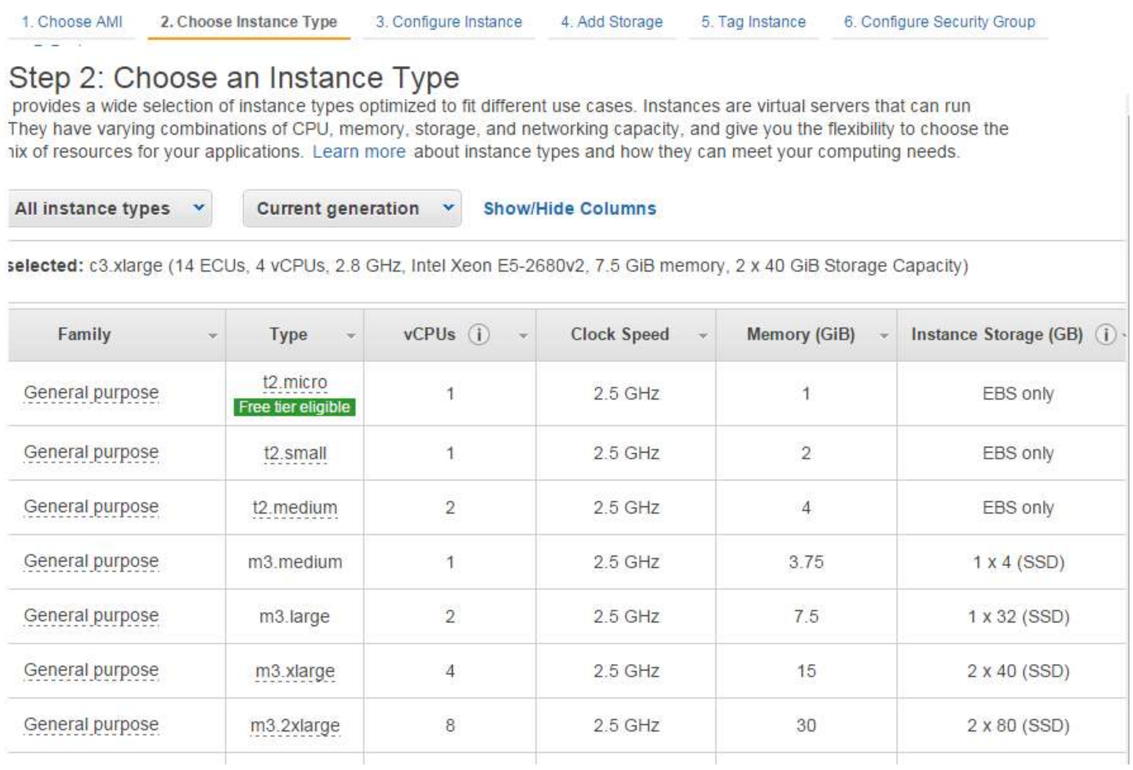


FIGURE 36. AWS region selection.

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.



Step 2: Choose an Instance Type

provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the ıix of resources for your applications. Learn more about instance types and how they can meet your computing needs.

selected: c3.xlarge (14 ECUs, 4 vCPUs, 2.8 GHz, Intel Xeon E5-2680v2, 7.5 GiB memory, 2 x 40 GiB Storage Capacity)

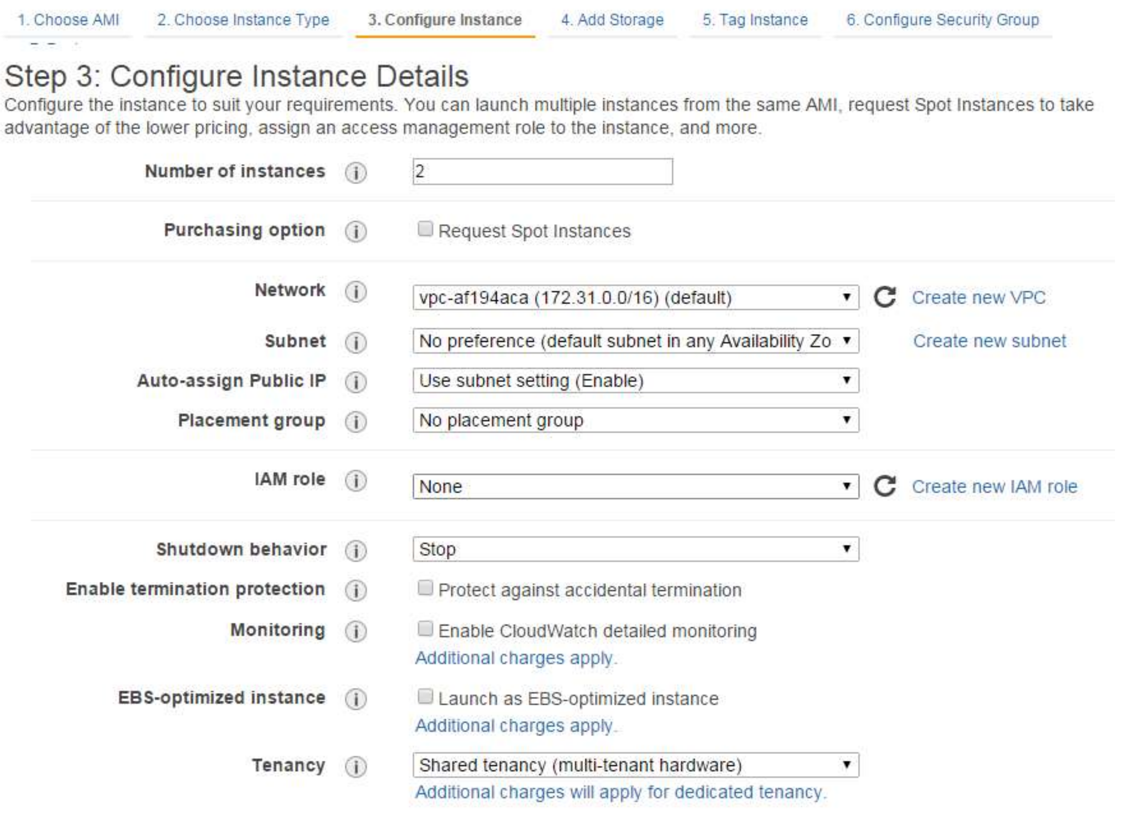
Family	Type	vCPUs	Clock Speed	Memory (GiB)	Instance Storage (GB)
General purpose	t2.micro Free tier eligible	1	2.5 GHz	1	EBS only
General purpose	t2.small	1	2.5 GHz	2	EBS only
General purpose	t2.medium	2	2.5 GHz	4	EBS only
General purpose	m3.medium	1	2.5 GHz	3.75	1 x 4 (SSD)
General purpose	m3.large	2	2.5 GHz	7.5	1 x 32 (SSD)
General purpose	m3.xlarge	4	2.5 GHz	15	2 x 40 (SSD)
General purpose	m3.2xlarge	8	2.5 GHz	30	2 x 80 (SSD)

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.



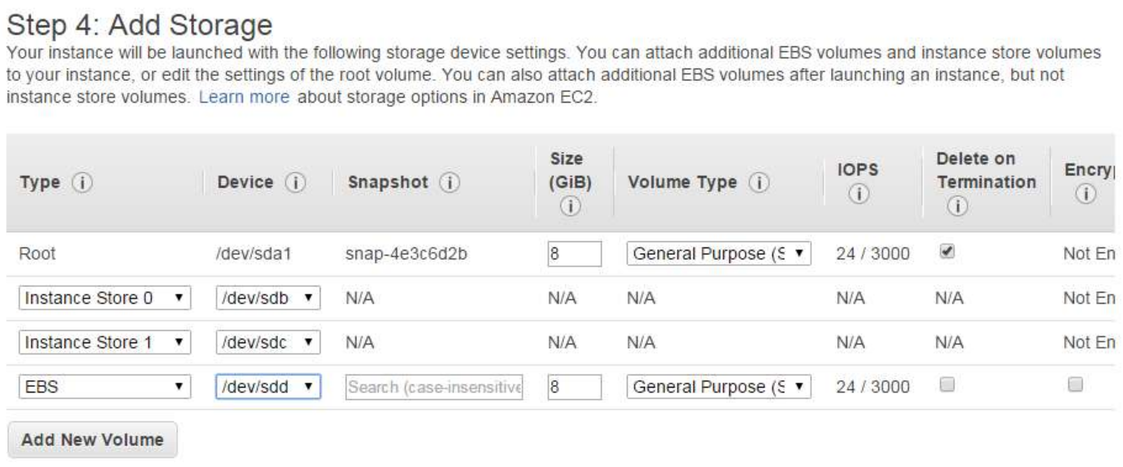
Step 3: Configure Instance Details

Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot Instances to take advantage of the lower pricing, assign an access management role to the instance, and more.

Number of instances	2
Purchasing option	☐ Request Spot Instances
Network	vpc-af194aca (172.31.0.0/16) (default) — Create new VPC
Subnet	No preference (default subnet in any Availability Zo — Create new subnet
Auto-assign Public IP	Use subnet setting (Enable)
Placement group	No placement group
IAM role	None — Create new IAM role
Shutdown behavior	Stop
Enable termination protection	☐ Protect against accidental termination
Monitoring	☐ Enable CloudWatch detailed monitoring Additional charges apply.
EBS-optimized instance	☐ Launch as EBS-optimized instance Additional charges apply.
Tenancy	Shared tenancy (multi-tenant hardware) Additional charges will apply for dedicated tenancy.

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'.



Step 4: Add Storage

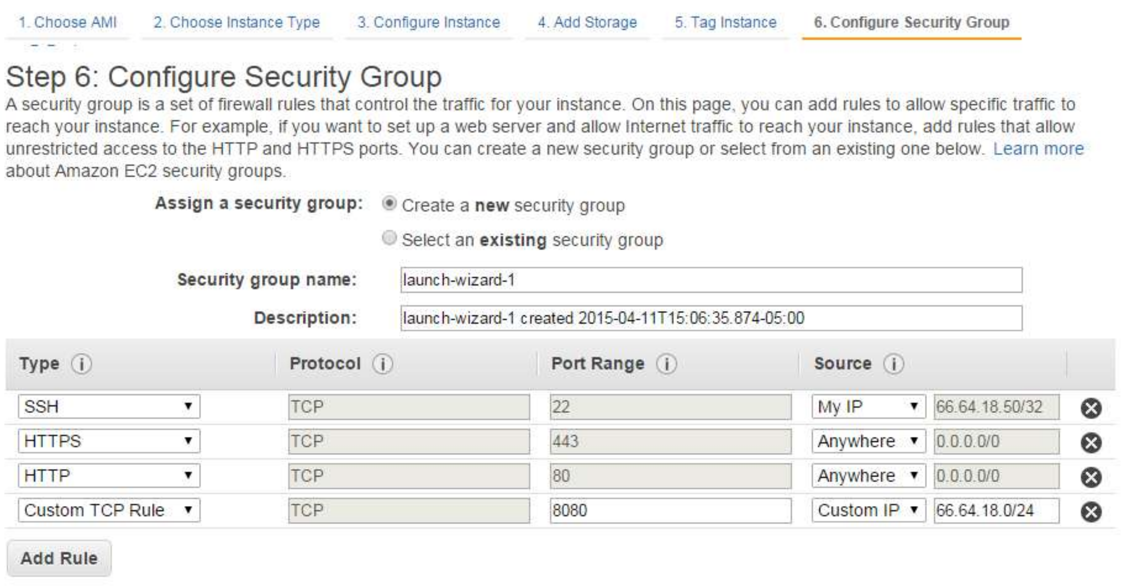
Your instance will be launched with the following storage device settings. You can attach additional EBS volumes and instance store volumes to your instance, or edit the settings of the root volume. You can also attach additional EBS volumes after launching an instance, but not instance store volumes. Learn more about storage options in Amazon EC2.

Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Delete on Termination	Encry
Root	/dev/sda1	snap-4e3c6d2b	8	General Purpose (S	24 / 3000	✓	Not En
Instance Store 0	/dev/sdb	N/A	N/A	N/A	N/A	N/A	Not En
Instance Store 1	/dev/sdc	N/A	N/A	N/A	N/A	N/A	Not En
EBS	/dev/sdd	Search (case-insensitive	8	General Purpose (S	24 / 3000	☐	☐

Add New Volume

FIGURE 39. AWS storage selection.

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.



1. Choose AMI | 2. Choose Instance Type | 3. Configure Instance | 4. Add Storage | 5. Tag Instance | 6. Configure Security Group

Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. Learn more about Amazon EC2 security groups.

Assign a security group: [✓ Create a **new** security group] Select an **existing** security group

Security group name: launch-wizard-1

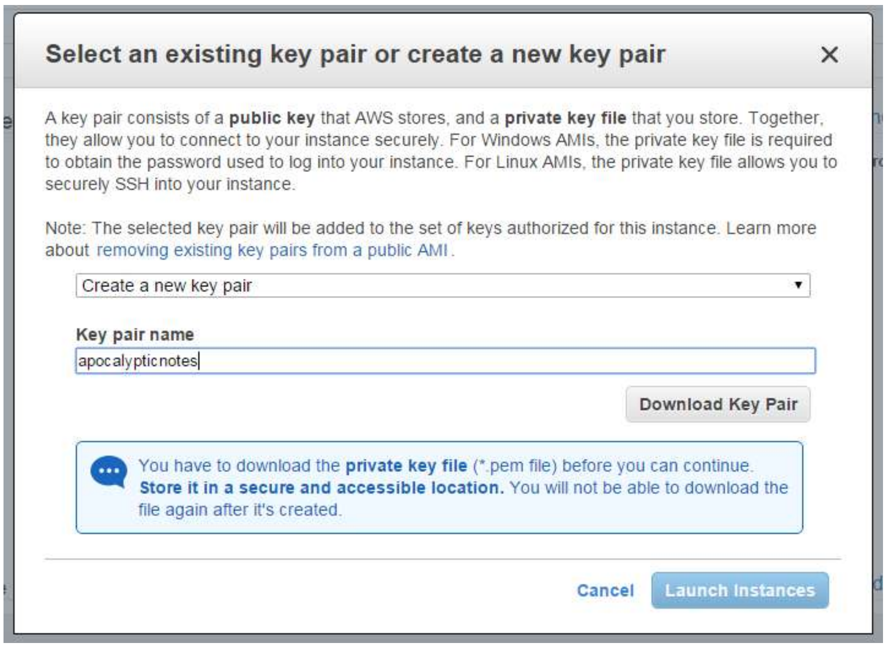
Description: launch-wizard-1 created 2015-04-11T15:06:35.874-05:00

Type	Protocol	Port Range	Source	
SSH	TCP	22	My IP	66.64.18.50/32
HTTPS	TCP	443	Anywhere	0.0.0.0/0
HTTP	TCP	80	Anywhere	0.0.0.0/0
Custom TCP Rule	TCP	8080	Custom IP	66.64.18.0/24

Add Rule

FIGURE 40. AWS firewall configuration.

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.


Select an existing key pair or create a new key pair

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about removing existing key pairs from a public AMI.

Create a new key pair

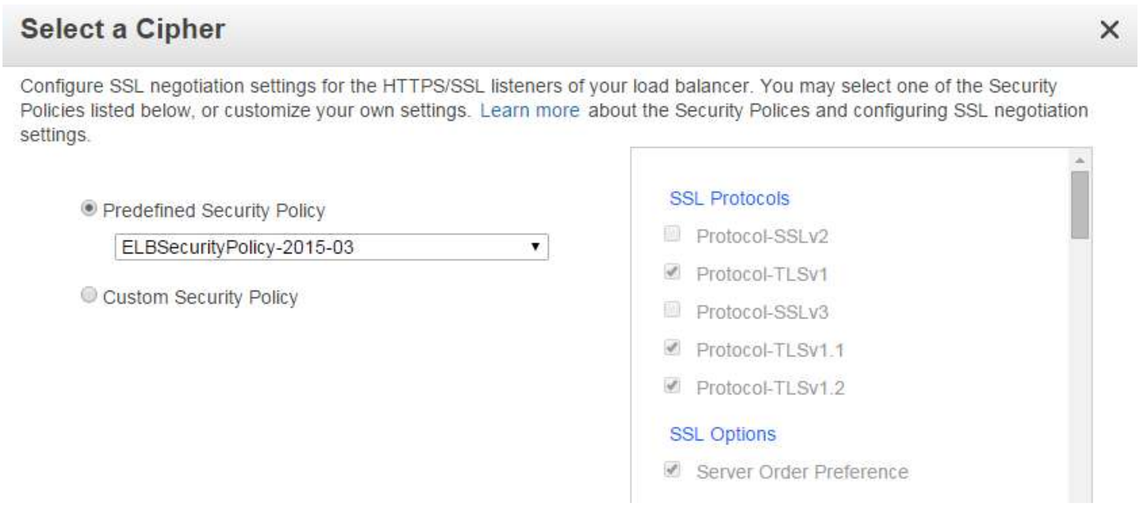
Key pair name: apocalypticnotes

You have to download the **private key file** (*.pem file) before you can continue. **Store it in a secure and accessible location.** You will not be able to download the file again after it's created.

FIGURE 41. AWS instance key pair setup.

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.


Select a Cipher

Configure SSL negotiation settings for the HTTPS/SSL listeners of your load balancer. You may select one of the Security Policies listed below, or customize your own settings. Learn more about the Security Polices and configuring SSL negotiation settings.

[● Predefined Security Policy] ELBSecurityPolicy-2015-03

○ Custom Security Policy

SSL Protocols

- ☐ Protocol-SSLv2
- ☑ Protocol-TLSv1
- ☐ Protocol-SSLv3
- ☑ Protocol-TLSv1.1
- ☑ Protocol-TLSv1.2

SSL Options

- ☑ Server Order Preference

FIGURE 42. AWS HTTPS supported cipher suite selection.

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 the balancers health check to fail. As a default a load balancer considers 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.

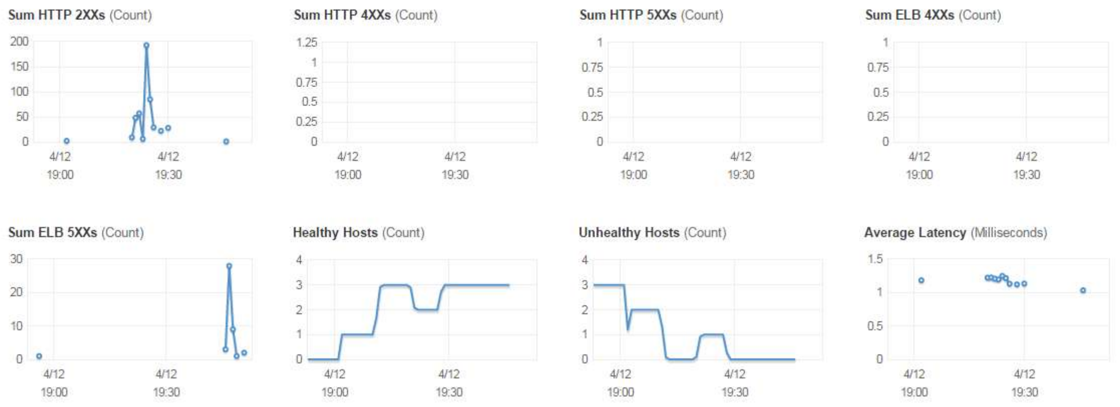


FIGURE 43. AWS load balancer status graphs.

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.

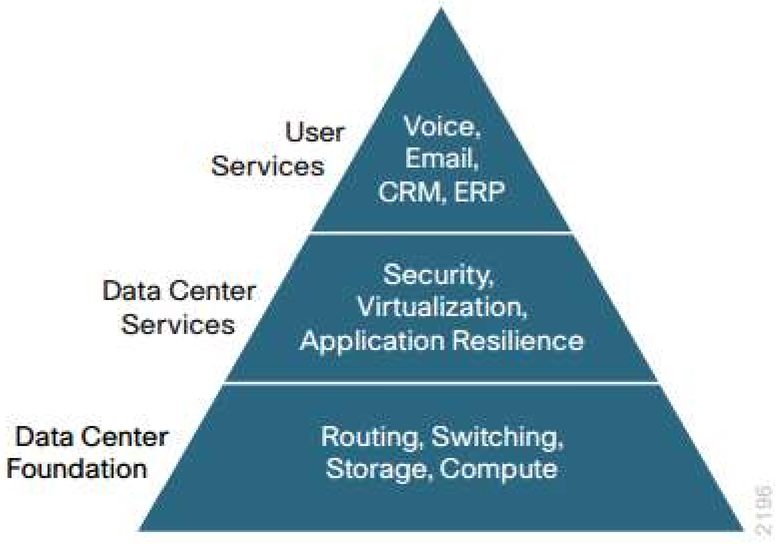

User Services
Voice, Email, CRM, ERP
Data Center Services
Security, Virtualization, Application Resilience
Data Center Foundation
Routing, Switching, Storage, Compute
2196

FIGURE 44. Data center pyramid of service layers (Cisco, 2014).

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.

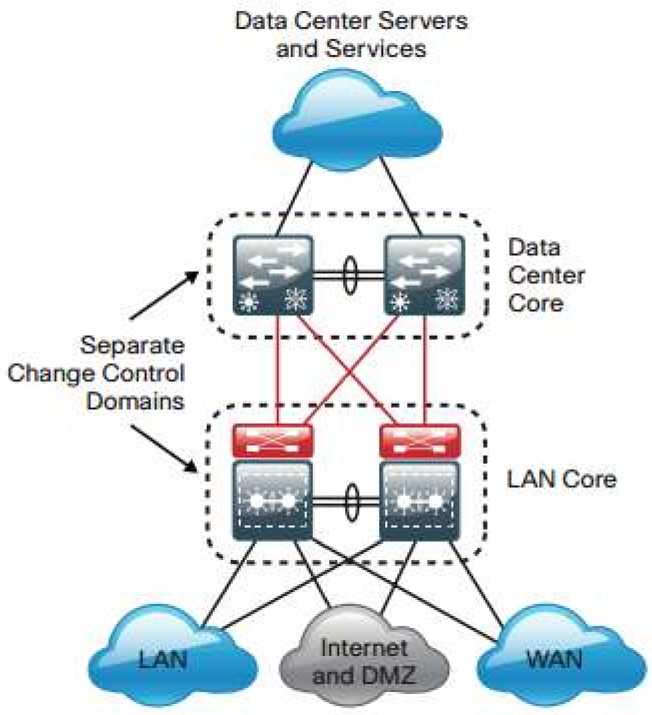

Data Center Servers and Services
Data Center Core
Separate Change Control Domains
LAN Core
LAN
Internet and DMZ
WAN

FIGURE 45. Data Center core and LAN core change control separation. (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).

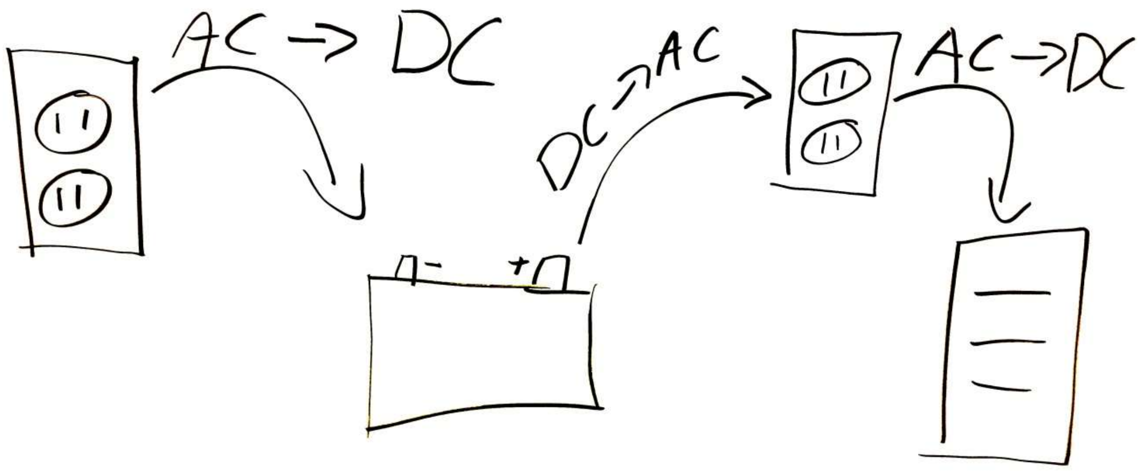


FIGURE 46. Conventional from outlet to ups to computer route.

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.

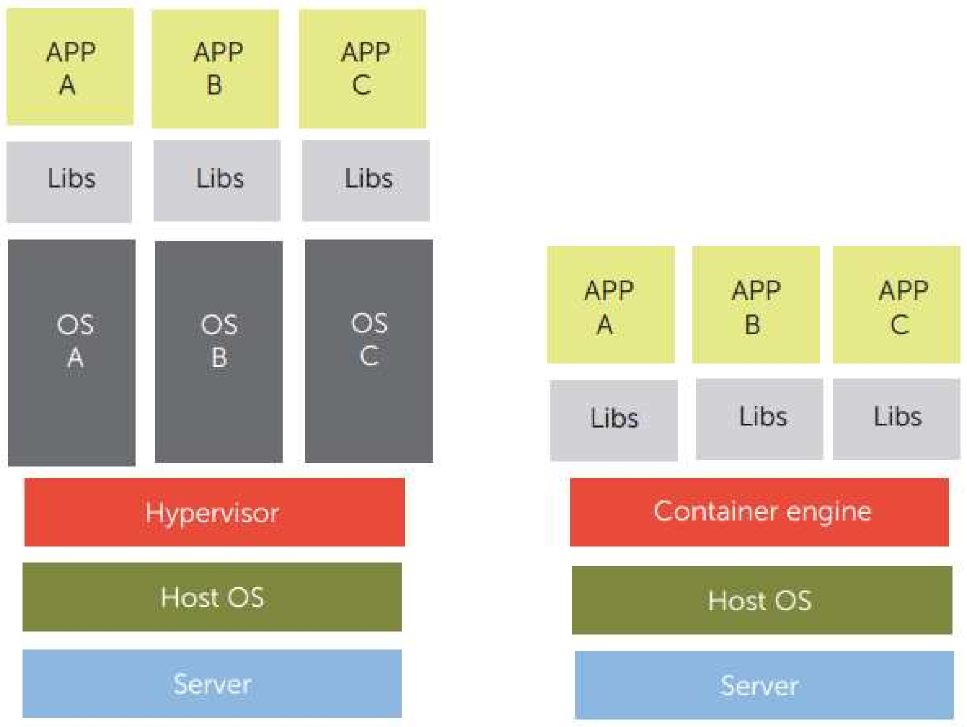


FIGURE 47. "Comparison of hypervisor and container-based deployments. A hypervisor-based deployment is ideal when applications on the same cloud require different operating systems or different OS versions; in container-based systems, applications share an operating system, so these deployments can be significantly smaller in size." (Bernstein, 2014)

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)

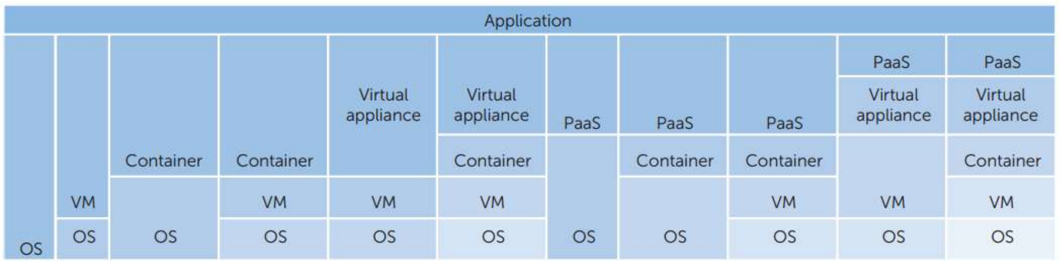


FIGURE 48. Possible layering combinations for application runtimes. (Bernstein, 2014)

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.

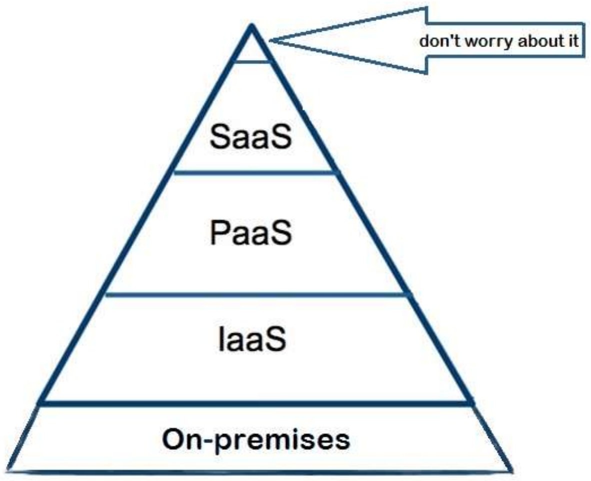


FIGURE 49. Five levels of stress over license server.

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

Amazon Web Services. Amazon Route 53 Developer Guide (API Version 2013-04-01). Date of retrieval 25.04.2015
http://docs.aws.amazon.com/Route53/latest/DeveloperGuide/routing-policy.html#routing-policy-geo

Amazon Web Services. Developer Guide (API Version 2010-12-01) Elastic Beanstalk Walkthrough. Date of retrieval 25.04.2015
http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/GettingStarted.Walkthrough.html

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

Atkins, R. 2011. Watching You. Date of retrieval 25.04.2015
https://www.youtube.com/watch?v=2uzK3VwzraM

Baker, J. 2014. The Register, Economics prof denies digital pirates plundered €20bn from EU coffers. Date of retrieval 25.04.2015
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
https://aws.amazon.com/blogs/aws/cross-region-read-replicas-for-amazon-rds-for-mysql/

BBC Worldwide. 2014. Submission in response to the Australian government's online copyright infringement discussion paper. Date of retrieval 25.04.2015
http://www.ag.gov.au/Consultations/Documents/OnlineCopyrightInfringement/OnlineCopyrightInfringement-BBCWorldwide.pdf

Berners-Lee, T., Fielding, R., Irvine, U.C., Gettys J., Mogul, J., Frystyk, H., Masinter, L., Leach P. 1999. IETF, Hypertext Transfer Protocol -- HTTP/1.1. Date of retrieval 25.04.2015
https://tools.ietf.org/html/rfc2616

Berners-Lee, T. 1992. HyperText Transfer Protocol. Date of retrieval 25.04.2015
http://www.w3.org/History/19921103-hypertext/hypertext/WWW/Protocols/HTTP.html

Bernstein, D., 2014, Containers and Cloud: From LXC to Docker to Kubernetes, IEEE Cloud Computing, Volume 1, Issue 3

Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., Moeller, B. 2006. IETF, Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS). Date of retrieval 25.04.2015
https://tools.ietf.org/html/rfc4492

BSA – The software Alliance. 2014. The Compliance Gap, BSA Global Software Survey. Date of retrieval 25.04.2015
http://globalstudy.bsa.org/2013/downloads/studies/2013GlobalSurvey_Study_en.pdf

Chatzakis, A. 2014. WordPress: Best Practices on AWS. Date of retrieval 25.04.2015
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=vyIap827rHs

Cisco. 2014. Data Center Technology Design Guide. Date of retrieval 25.04.2015
http://www.cisco.com/c/dam/en/us/td/docs/solutions/CVD/Aug2014/CVD-DataCenterDesignGuide-AUG14.pdf

Dierks, T., Rescorla, E. 2008. IETF, HTTP over TLS. Date of retrieval 25.04.2015
https://tools.ietf.org/html/rfc5246

Diffie, W., Hellman, M.E. 1976. New Directions in Cryptography, IEEE Transacactions, Volume 22, Issue 6.

Doherty, S., Gegeny, J., Spasojevic, B., Baltazar. J. 2013. Hidden Lynx – Professional Hackers for Hire. Date of retrieval 25.04.2015 http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/hidden_lynx.pdf

Far, P., Grigorik, I. 2014. Google I/O 2014, HTTPS Everywhere. 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

Gates, B. 1976. Homebrew Computer Club Newsletter Volume 2. Date of retrieval 25.04.2015 http://commons.wikimedia.org/wiki/File:Bill_Gates_Letter_to_Hobbyists.jpg

Gibbs, K. 2008. Google Campfire one. Date of retrieval 25.04.2015 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

Iltalehti. 2012. Jenna Lepomäen murha: Vankilasta löytyi järkyttävä salakirjoitus. Date of retrieval 25.04.2015
http://www.iltalehti.fi/uutiset/2012060515670142_uu.shtml

Johnson, B. 2010. The Guardian, When using open source makes you an enemy of the state. Date of retrieval 25.04.2015
http://www.theguardian.com/technology/blog/2010/feb/23/opensource-intellectual-property

Kahn D. 1973, The Codebreakers, The story of secret writing, The Macmillan Company, NY: The New American Library

Karaganis, J., Renkema, L. Copy culture in us and Germany. Date of retrieval 25.04.2015
http://americanassembly.org/sites/default/files/download/publication/copy_culture.pdf

Kerner, S.M. 2015. eWeek, Let's Encrypt. Date of retrieval 25.04.2015
http://www.eweek.com/security/lets-encrypt-becomes-linux-foundation-collaborative-project.html

Levy C.J. 2010. The New York Times, Russia Uses Microsoft to Suppress Dissent. Date of retrieval 25.04.2015
http://www.nytimes.com/2010/09/12/world/europe/12raids.html?_r=0

Marantis, D. 2013. Office of the United States Trade Representative, 2013 Special 301 Report. Date of retrieval 25.04.2015
http://www.mpaa.org/wp-content/uploads/2014/02/2013-Special-301-Report.pdf

McCullagh, D. 2002. Wired, Another punch for copy protection. Date of retrieval 25.04.2015
http://archive.wired.com/politics/law/news/2002/03/51400?currentPage=all

Mell, P., Grance, T. 2011. The NIST Definition of Cloud Computing. Date of retrieval 25.04.2015
http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

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
https://msdn.microsoft.com/en-us/library/bb397867(v=vs.110).aspx

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

NIST. 2001. Federal Information Processing Standards publication 197, Announcing the ADVANCED ENCRYPTION STANDARD (AES) . Date of retrieval 25.04.2015
http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

NIST. 2012. Secure Hash Standard (SHS) . Date of retrieval 25.04.2015
http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf

Nykänen, J. 2003. Tekniikanmaailma, Vialliset levyt. Date of retrieval 25.04.2015
http://tekniikanmaailma.fi/vanhat/vialliset-levyt

Parloff, R. 2007. Fortuen, Microsoft takes on the free world. Date of retrieval 25.04.2015
http://archive.fortune.com/magazines/fortune/fortune_archive/2007/05/28/100033867/index.htm

Pingdom. 2009. Wacky copy protection methods from the good old days. Date of retrieval 25.04.2015
http://royal.pingdom.com/2009/08/26/wacky-copy-protection-methods-from-the-good-old-days/

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

Rivest, R., Shamir, A., Adleman, L. 1978. A Method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM21.

RSA Laboratories. 2012. RSA Cryptography Standard. Date of retrieval 25.04.2015
http://www.emc.com/emc-plus/rsa-labs/pkcs/files/h11300-wp-pkcs-1v2-2-rsa-cryptography-standard.pdf

Schneier, B. 2005. Real Story of the Rogue Rootkit. Date of retrieval 25.04.2015
http://archive.wired.com/politics/security/commentary/securitymatters/2005/11/69601?currentPage=all

Shankland, S. 2009. Google uncloaks once-secret server. Date of retrieval 25.04.2015
http://www.cnet.com/news/google-uncloaks-once-secret-server-10209580/

Stallman, R. 2015. Free Software Is Even More Important Now. Date of retrieval 25.04.2015
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

Teosto. 2012. Chisu-gatessa sovinto. Date of retrieval 25.04.2015
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
http://www.teraconsultants.fr/en/issues/The-Economic-Contribution-of-the-Creative-Industries-to-EU-in-GDP-and-Employment

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

TIM. 2015. SSL Pulse – Survey of the SSL Implementation of the Most Popular Web Sites. Date of retrieval 25.04.2015
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