RSA authentication in Internet of Things

Technical limitations and industry expectations

FREDRIK JONSSON

MARTIN TORNKVIST
Abstract

The objective of this thesis is to evaluate if it is possible to run RSA authentication in a specified scenario. A Raspberry Pi with a limited CPU is used to simulate a low-performance device. A series of tests on this device shows that it is not possible to run RSA authentication in the provided scenario. A survey conducted on IT-professionals shows that there is a strong belief that this is possible. The results shows that there is a disparity between the tested RSA performance and the perception in the industry. However since ambiguity exists in the scenario it is hard to draw conclusions about the results.
Acknowledgement

We would like to thank Patrick Kall at Telenor for his guidance and feedback. We also want to thank all the companies and individuals who have answered the survey.
Glossary

AC  Asymmetric Cryptography, same as public key cryptography. 16

AES  Advanced Encryption Standard, a standard for encryption based on symmetric-keys. 15

CoAP  Constrained Applications Protocol, application layer protocol intended for constrained devices. 12

DES  Data Encryption Standard, a previous standard for encryption, replaced by AES. 15

DoS  Denial Of Service, an attack that aims at making a service or machine unavailable to its users. 12, 18, 25

DTLS  Data gram Transport Layer Security, a protocol designed to allow transport layer security. 12

ECC  Elliptic curve cryptography, a public key cryptosystem based on elliptic curves. 9, 13, 36, 37

IoT  Internet of Things. In this report defined as devices connected to the Internet with a limited computational capacity. 9–14, 17, 18, 37

IPsec  Internet Protocol Security, a protocol used for authentication and encryption. 12

LAN  Local Area Network, usually a limited network connected via ethernet. 17, 18

MFA  Multi-factor authentication, an authentication method where the user needs to provide several methods of authentication. 15

OSI  Open Systems Interconnection, a model for data communication in 7 layers. 17

PKC  Public Key Cryptography, a cryptosystem that uses public as well as private keys. Examples are RSA and ECC. 12, 13

RFID  Radio Frequency Identification, a wireless technology for identifying tags. 9, 11, 13, 18

RSA  A public key cryptosystem named after its inventors Ron Rivest, Adi Shamir and Leonard Adleman. 9, 10, 13, 16, 19, 21, 27–29, 34–37
SSL  Secure Sockets Layer, cryptography protocol for secure connections. 13

top  Displays information about the top CPU processes in a Unix system. 28, 29

WEP  Wired Equivalent Privacy, a security algorithm for wireless networks. 15

WSN  Wireless Sensor Network. A network of connected autonomous sensors. 12, 13, 18, 19

XOR  Exclusive or, a logical operation that only outputs true when one input is true. 15
Table Of Contents

Glossary .......................................................... 3

1 Introduction .................................................... 9
   1.1 News Value ................................................. 9
       1.1.1 Social and Ethical Aspects ....................... 10
   1.2 Scientific Question ...................................... 10
       1.2.1 Problem Definition ............................... 10
       1.2.2 Delimitations .................................. 10
       1.2.3 Expected Result ................................. 11

2 Related Work ................................................ 12
   2.1 Two IoT architecture approaches ..................... 12
   2.2 PKC Compared to Pre-shared Keys .................... 12
   2.3 RSA Compared to ECC .................................. 13
   2.4 IoT Security from a Layer Perspective ............... 13

3 Background .................................................... 14
   3.1 Defining IoT ............................................. 14
   3.2 Authentication .......................................... 14
       3.2.1 Authentication Factors ......................... 14
       3.2.2 Passwords .................................. 14
       3.2.3 Multi-Factor Authentication ................... 15
       3.2.4 Key Methods ................................ 15
   3.3 RSA ....................................................... 16
       3.3.1 History ........................................ 16
3.3.2 Algorithm ........................................... 16
3.3.3 RSA Authentication ............................... 16

3.4 IoT Device Architecture ............................. 17
3.4.1 Application Layer ................................. 17
3.4.2 Network Layer ...................................... 17
3.4.3 Support Layer .................................... 18
3.4.4 Perception/Recognition Layer .................. 18

4 Method - Survey ...................................... 19
4.1 Respondents ........................................ 19
4.2 Scenario ............................................. 19
4.3 Survey ................................................ 20
4.3.1 Question 1 ......................................... 20
4.3.2 Question 2 ......................................... 20
4.3.3 Question 3 ......................................... 21
4.3.4 Question 4 ......................................... 21
4.3.5 Question 5 ......................................... 21
4.4 Process ............................................... 21
4.5 Evaluation .......................................... 21

5 Results - Survey ..................................... 22
5.1 Demographics ....................................... 22
5.1.1 Gender ............................................ 22
5.1.2 Age Distribution .................................. 22
5.2 Education ............................................ 23
5.3 Profession ............................................ 23
8.2.2 Demographics ........................................ 35
8.2.3 Scenario ........................................... 35
8.3 Suggestions for Future work ............................... 36
8.4 Conclusion .............................................. 37

References .................................................. 38

A Appendix .................................................. 39
B Appendix .................................................. 46
C Appendix .................................................. 51
D Appendix .................................................. 57
1 Introduction

This report tests the RSA algorithm on a Raspberry Pi using the Go standard library. The central processing unit (CPU) of the Raspberry Pi is limited to emulate low performing device.

The growth of Internet of Things (IoT) has increased rapidly in recent years. Much is due to improvements in technology which makes computers cheaper, smaller and faster. Ericsson estimate that by 2020 there will be 16 billion connected devices whilst Information Handling Services (IHS) estimate about 30,7 billion [1]. The areas of use are temperature sensors, surveillance cameras and Radio Frequency Identifiers (RFID) sensors.

To ensure that no untrusted users can get access to the devices some form of authentication method has to be implemented. Normally this is achieved using public key cryptography, for example RSA or Elliptic Curve Cryptography (ECC). With IoT the implementation of these algorithms are more challenging due to their, often limited, computational capabilities. One solution is to use symmetric key cryptography instead, which is less resource heavy, but this presents another issue since the keys have to be distributed [2].

1.1 News Value

According to [3, 4] the following factors makes security in IoT especially difficult in comparison to regular computer security. Devices often have limited resources in the form of memory and CPU and they are often required to be fully autonomous without human interaction. Sometimes IoT are battery operated which limits computational power and they are required to communicate with many other devices without human monitoring. It is very important to solve these problems if IoT are going to be used on a large scale. The relatively new phenomenon of IoT and how it separates itself from normal computer security makes it an interesting subject to study.

The increased use of IoT is not only happening in the consumer market. Many companies have started to use IoT as a part of their daily operations. Security breaches in these devices could lead to unwanted and catastrophic events. One example is Lufthansa who is using IoT in its operations to monitor maintenance repair and overhaul (MRO) [1]. A business such as aviation maintenance will put a high demand on the security standards.

To make sure these IoT networks are secure some form of authentication method needs to be implemented to keep unwanted users from gaining access to the network. Since IoT often have limited CPU, memory and power the authentication algorithms could prove too be to demanding for the devices in
the network [2]. The goal of this paper is to find the minimum specifications required for implementation of secure authentication algorithms for different network specifications.

1.1.1 Social and Ethical Aspects

Since the report will focus on an area of computer security the results might show security vulnerabilities in IoT. This information could be used by people with bad intentions and this must be considered before publishing. Since the report will contain a survey it is important that the information about the respondents is kept confidential.

1.2 Scientific Question

Is it possible to use the RSA algorithm for authentication when using a device with limited computational capabilities?

1.2.1 Problem Definition

Previous studies suggest that the low computational capabilities in IoT is one of the main limitations for implementing good authentication. This paper aims to find and investigate two things. Firstly, to find the threshold where the authentication methods are no longer practically usable. Secondly, to investigate if there is a disparity between the way authentication issues are perceived in the industry and the results from performance tests in this paper.

1.2.2 Delimitations

The tests will be limited to the RSA algorithm since this is the most widely used public key algorithm. The RSA implementation will be imported from the Go standard library using the PKCS1v15 implementation. To simulate a low performing device, the CPU clock frequency will be limited.

There are many definitions of IoT. This paper will focus on devices with limited computational capabilities. To emulate these kind of devices a downclocked Raspberry Pi will be used.

There are many different situations where the question could be tested and the result could differ depending on the situation. A scenario have been created so the question could be evaluated in that specific environment.
1.2.3 Expected Result

Our hypothesis was that the difficulty to implement authentication methods would only be true for low performing IoT such as small sensors and RFID tags. As the number of authentication requests increase we expected the resource requirements to increase. We expected to find the limit where it no longer would be practical to use authentication in an IoT device such as a Raspberry Pi.
2 Related Work

2.1 Two IoT architecture approaches

Since IoT is still a relatively new phenomenon there is still much research to be done in the area of IoT security. Roman et al. [3] describes two IoT architecture approaches for connecting IoT to the users and how these approaches give way to different security issues [3].

The Centralized Approach In this approach the devices are connected-many-to one. Every interaction with a IoT device is done through centralized servers. This greatly simplifies the security implementation since it reduces the amount of entry points to the devices. The authentication and access control can be done on the central servers reducing the demands of the IoT devices themselves [3].

The Distributed Approach In this approach many-to-many-connections are possible. Users are able to connect directly to devices on a local network as well as through the internet, increasing the security demands since there are more entry points. In this approach every device must implement an authentication and access procedure putting more pressure on the devices and their computational capability [3].

The lack of computational resources is a large part of why the implementation of good authentication methods is not a trivial problem and not easily transferable from the already accepted authentication and access control methods. Since protocol and network security follows from successful authentication the same problems apply here. There are some approaches to adapting protocols to the more lightweight requirements, for example constrained application protocol (CoAP), datagram transport layer security (DTLS) and internet protocol security (IPsec) [3].

2.2 PKC Compared to Pre-shared Keys

Roman et al. [5] compares public key cryptography (PKC) and pre-shared keys as two ways to ensure authentication in a wireless sensor network (WSN). A WSN is a network of interconnected sensors generally connected to a common node. The limitation of PKC in that it requires more resources is emphasized. The PKC approach is viable in the case that the WSN node is used as a client only. When using the WSN node as a server it is possible to perform a DoS attack against the WSN by forcing the node to make many PKC calculations.
The time taken for these calculations is estimated from the algorithms and the specifications of the ATmega128L microcontroller. In the paper class II nodes are the primary focus. These types of nodes have 4-16 KB of RAM and a microcontroller of 4-8 MHz.

2.3 RSA Compared to ECC

In [6] two PKC-algorithms ECC and RSA for secure sockets layer (SSL) are evaluated. In this comparison they evaluated the two algorithms in three real world scenarios.

1. A peer-to-peer connection between two handheld Yopy PDA:s.
2. A connection between a Ultra-80 server and the Yopy PDA.
3. A connection between two Ultra-80 servers.

According to [6] ECC was shown to be considerably faster especially when considering higher encryption standards. The CPU performance of the Yopa PDA is 200 MHz and for the Ultra-80 server it is 450 MHz.[6]

2.4 IoT Security from a Layer Perspective

In [7] the security of IoT is evaluated from the perspective of three layers: Application, Transportation and Perception. The report focuses on RFID and WSN types of IoT networks. An overview is given of all the security issues and the current solutions, also here the problem with the low computational capabilities of IoT is identified.

Jing et al. (2014) [7] emphasize the need for development of computationally efficient algorithms for key management, authentication and access control. Public and symmetric key cryptography is compared. Both are seen as viable solutions to the problems but public key cryptography needs to be implemented in an efficient way in WSN and RFID to be a viable solution. The conclusion is that further research needs to be done in this area [7].
3 Background

3.1 Defining IoT

The term IoT is thought to be coined in 1999 by Kevin Ashton in a presentation he gave at Procter and Gamble (P&G) initially referring to Radio Frequency Identification (RFID). The term is used very differently depending on context. Gubbi et al. describes IoT as "A worldwide network of interconnected devices" [8]. This is a very broad term that encompasses everything connected to the internet. We will use this definition as a basis for our investigation but focus on devices with a limited computational capability. In this report the focus will be on devices with single core CPU performance of 100 MHz or below. This means that we are not interested in devices such as smart phones, personal computers or servers.

3.2 Authentication

According to the Oxford Dictionary, the definition of authentication is “the process or action of proving or showing something to be true, genuine, or valid”. Or in a more computer related sense, “the process or action of verifying the identity of a user or process”. This differs from identification where a person simply states that he or she is a certain person. A secure system can’t trust the user so there is a need for a verification of some sort.

3.2.1 Authentication Factors

Knowledge Something you know. This could be a password, PIN or pattern.

Possession Something you have. A physical object like your ATM card or smart phone.

Inherence Something you are. This is something associated with the user, usually a biometric like fingerprint or voiceprint.

3.2.2 Passwords

A common method to verify that a user is whom he or she claims to be, is the use of a password or passphrase. This is a simple method that even kids
can use: “only people that know the secret word is allowed in my room”. This example illustrate how effective passwords can be, a user will not be allowed into the system if he or she cannot verify that they are allowed to be there. This however has an obvious flaw. If a non-authorized user gets a hold of the password, the system would give the wrong person access. The password could be overheard or stolen from the user, and you could have a system that allowed “guesses” which would result that an intruder would guess the password, given unlimited guesses.

3.2.3 Multi-Factor Authentication

When using a multi-factor authentication (MFA) the user needs to provide several pieces of evidence that verifies their identity before gaining access.

Two factor authentication Most people use two-factor verification (2FA) on an everyday basis. A common situation is when they use their ATM card to withdraw money or make a purchase. This usually requires the user to provide the card as well as the PIN. This example shows the usage of knowledge and possession authentication types.

3.2.4 Key Methods

Symmetric cryptography This method uses the same key for encryption and decryption, meaning that both ends of the communication link needs to have the key. This key needs to be kept hidden from unauthorized users and can only be shared in a secure way. There are two types of symmetric key cryptography: Stream ciphers and block ciphers.

Stream ciphers Uses a fixed length key to produce a pseudo random stream of bits. To encrypt a message you XOR the key bits with the message bits. And to decrypt you XOR the key bits with the encoded message bits. The RC4 used in WEP for wireless network security and the a5/1 used to encrypt GSM phone data and conversations are two examples of stream ciphers.

Block ciphers Block ciphers encrypt a block of data instead of one bit at a time like the stream ciphers. Data Encryption Standard (DES) and Advanced Encryption Standard (AES) are examples of block ciphers. The AES encrypts 128-bits blocks and uses a key with a length of 128, 192 or 256-bits.
**Asymmetric cryptography**  Asymmetric cryptography (AC) allows one person to encrypt with one key and another person to decrypt with a different key. One key for encryption and one for decryption, these two form a key-pair. This has two uses:

1. Confidentiality - send secret messages to someone
2. Integrity - Prove who created the message

Every user has two keys, a public key and a private key.

- Public key - Not kept secret. Shared with everyone.
- Private key - Kept secret. Only the user can have it.

### 3.3 RSA

#### 3.3.1 History

The RSA algorithm is a public key encryption algorithm and the most common and well known of all public key algorithms. RSA was developed in 1970 by Ron Rivest, Adi Shamir and Len Adleman [9].

The algorithm is based upon the difficulty of finding the prime factors of a large number. For encryption and decryption the algorithm uses a public and a private key. To authenticate a session with a server and a client the following steps are performed:

#### 3.3.2 Algorithm

The algorithm utilizes a public key PU and a private key K. These keys in turn consist of two prime numbers, PU = e, n and K = d, n.

To encrypt a message M a cipher C is calculated using $C = M^e (mod \ n)$. The message can later be retrieved using the private key, $M = C^d (mod \ n)$ [9].

#### 3.3.3 RSA Authentication

1. The server starts by encrypting a message using the client’s public key and sends this message to the client.
2. The client decrypts this message using its own private key.
3. The client encrypts this message using the server’s public key and sends it back to the server.

4. When the server receives the message it is able to decrypt the message using its own private key.

5. If the message matches the original message the server knows that the client is authorized to login. [9, 10]

3.4 IoT Device Architecture

The IoT architecture can be divided into four key layers [4]. They are based on the Open Systems Interconnection (OSI) model. This section will cover an introduction for each layer and also the security features and security requirements for each of the layers.

3.4.1 Application Layer

The application layer is the topmost layer and is closest to the user. The user can communicate with the IoT device through an interface using a series of different devices e.g. smartphone, computer or TV [4]. The types of services that the IoT provides to the user will operate on this layer.

Depending on the application, the security needs are different. Data sharing is the security issue that is present in almost every device and creates some different security challenges e.g. access control, information safety and privacy of the data [4].

There are two different solutions to the security problems in this layer. The first is authentication and key agreement. This solution protects the data and the device by verifying that only the authorized persons can obtain or change the data. The second is the user’s awareness about information security. This is accomplished by educating the user and administrator about security e.g. password management [4].

3.4.2 Network Layer

The network layer handles the network communications and is responsible for securing the data transfer. This includes providing a dependable transmission for the information from the perceptual layer. The transfer of the information can use wireless technology such as Bluetooth, Infrared and Wi-Fi but also a wired connection to a local area network (LAN) [11].
The safe protection ability for this layer is relatively complete but there are some security issues that are common problems concerning Internet, Wi-Fi and LAN. This includes man-in-the-middle attack, eavesdropping, counterfeit attack and denial of service (DoS attack).

The problem with IoT in wireless networks are that nodes can move freely, and some need to have the possibility to join and leave without prior authentication [12], making the network more vulnerable. To prevent illegal nodes to connect to the network you need some kind of authentication process.

### 3.4.3 Support Layer

The support layer will support the application layer. This can be done by using cloud computing and network grids [4]. The ability to use a more powerful machine for computing is very useful for a IoT device which usually doesn’t have the processing power required. This layer works with both the application layer and the network layer.

To help with the security in the network layer this layer can do the mass data processing and make intelligent decision of what is suspicious behavior. Devices with limited resources could use this to generate keys for use in a secure authentication method.

### 3.4.4 Perception/Recognition Layer

This layer collects all kinds of information through sensing devices such as Radio Frequency Identification (RFID), GPS, Zigbee, Smart card and sensor networks. RFID technology uses microchips for wireless data communication which helps with automatic identification of the physical property they are attached to [11]. The information that is gathered identifies the physical world in the digital world.

The collected data is transmitted through WSN. WSN are vulnerable to attack like Node capture, Fake node, Malicious data and Denial of Service attacks [11]. The sensors are vulnerable to sensor attacks, sensor abnormalities and radio interference. The sensor data needs to be protected so its confidentiality, authenticity and integrity is not compromised.

The nodes in this layer often lack CPU power and memory capacity, mainly because they need to be simple, small and need to operate with less power [4]. This makes it hard to implement a security system and to apply the usual security responses like frequency hopping communication and key encryption algorithms. The solutions would be to use lightweight encryption algorithms.
4 Method - Survey

A survey was conducted on persons working in the IT-sector. The survey presented a scenario of a WSN. Based on this scenario the respondents answered a number of questions regarding computer security and RSA. Google Forms was used to create and host the survey. This platform was chosen due to the fact that it allowed seamless integration with the cloud service, Google Drive. The purpose of the survey was to collect data about how persons who work with IT view authentication in devices with limited computational resources.

The survey was designed to ask several questions related to different IT-security threats and solutions. The idea behind asking about other threats is that the answers could be compared in relation to the different threats. The survey is anonymous and the respondents got the choice to receive a copy of the report. The answers given by the respondent needed to still be anonymous even if they entered their contact information. To solve this problem, a new form was created. When the respondent was finished with the survey, they had a choice to go to the separate form and enter their contact information there.

4.1 Respondents

The respondents are persons working in the IT-sector. A number of companies that work with IT-security was chosen and they were asked to distribute the survey to their employees. The survey was also sent to individuals that work with IT-security. These people were chosen because they work with IT-security according to their LinkedIn page.

4.2 Scenario

This is the scenario that is presented to the respondents in the survey. In the survey the scenario was presented in Swedish but is here translated into English.

“We have a device that will be used as a gateway in a Wireless Sensor Network WSN. Communication between the user and the gateway is done through the internet. The specifications for the device is 100 MHz CPU and 512 MB RAM. 10 % of the device’s resources are reserved for the communication with the sensors. The network consist of 50 sensors which are spread out in an office building. Each sensor reads the temperature of the surrounding air and sends that data to the gateway once every minute.”
4.3 Survey

The survey was split into four sections. The first section presents information about us, the authors and our education. This section also presented an overview of the survey.

The second section asked questions about the respondent’s background. Year of birth, gender, level of education and professional category. These questions where asked to establish which types of people that answered the survey. The goal was to get a distribution of respondent that represent the distribution in the industry.

In the third section the scenario was presented. The scenario was designed to create context for the respondent. This specified scenario helps all respondents to answer the questions with the same set of delimitations.

The fourth and final section asked the computer security related questions. The first and second question was about security flaws related to the scenario. The third question asked about specific types of threats and how they would impact the security in the scenario. The fourth question asked about five security solutions and how they would improve the security in the scenario. The fifth question is a direct question if the respondent thinks it is possible to implement RSA in the scenario.

4.3.1 Question 1

The first question asked about how common the respondent perceived certain security threats. They where asked to estimate how often security threats are occurring by selecting from the following options: ”not occurring”, ”seldom occurring”, ”occurring” and ”frequently occurring”. These options where later given points of 1 to 4 where 4 is ”frequently occurring” and 1 means ”seldom occurring”. This allowed us to make a comparison of the options given and rank them according to the highest perceived threat.

4.3.2 Question 2

In question 2 the respondents where asked to estimate how much an existing security threat impacts security in the given scenario. The security threats in question 2 are the same as in question 1. The respondents where given the options ”no impact”, ”low impact”, ”high impact” and very ”high impact”. The options where given a number as in Question 1, ”no impact” gave a score of 1 while high a ”high impact” gave a score of 4. This allowed us to rank the options internally.
4.3.3 Question 3

The third question asked about external threats to our scenario. The respondents were given the options no threat, small threat, big threat and very big threat. The options were ranked from 1 meaning no threat to 4 meaning very big threat.

4.3.4 Question 4

Question 4 asked about what improvements can be achieved by implementing a security solution. The respondents were given the options "no improvement", "low improvement", "high improvement" and "very high improvement".

4.3.5 Question 5

Question 5 was directly related to our test of the RSA algorithm.

4.4 Process

The survey was sent out to companies that work with IT-security and they were asked to distribute it to their employees. The survey was also sent to individuals that work with IT-security. The survey was open to the respondents for two weeks. The answers was then collected and put together.

4.5 Evaluation

The answers from the survey was compiled and analyzed. The questions that relate to authentication and RSA are the main sources for then evaluation. To answer the scientific question, these answers was compared to the results from the performance tests. Conclusions was drawn from this comparison of how the perceived challenges correspond to the results from the performance testing.
5 Results - Survey

In this section the results from the survey are presented. Since the report is written in English and the survey was conducted in Swedish a choice was made to translate the survey to English to keep a language consistency throughout the report. The original formulations can be found in Appendix A.

5.1 Demographics

5.1.1 Gender

Of the respondents 90% are male, 7% are female and 3% has chosen not to disclose their gender. Compared to the average for the industry which according to SCB is 70% male and 21% female we have a slight oversampling of males [13].

5.1.2 Age Distribution

The age distribution is 17% in the age group 16-29, 45% in the age group 30-39, 28% in the age group 40-49, 3% in the age group 50-59 and 7% in the age group 60-69.
5.2 Education

A majority of the respondents, 86% have a university degree, 10% have vocational education (yrkeshögskola) and 4% have finished upper secondary school (gymnasium) as their highest form of education.

Figure 2: Education

![Pie chart showing education levels](chart)

5.3 Profession

A majority of the respondents, 90% work in the IT-sector.

Figure 3: Profession

![Pie chart showing professional sectors](chart)
5.4 Question 1

Figure 4 shows that bad password handling is the most highly perceived threat given the scenario. Bugs and non-updated software are ranked lowest when it comes to how often they are perceived to occur.

![Figure 4: Question 1](image)

5.5 Question 2

The result is shown in figure 5. According to the result the option with the highest perceived impact on security is when the manufacturer fails to change the default password. Low complexity passwords are perceived as having the lowest impact on security.

![Figure 5: Question 2](image)
5.6 Question 3

The results for question 3 can be seen in figure 6. Brute force attacks are perceived as the largest threat to security in the scenario whilst DoS attacks is the lowest ranked.

![Figure 6: Question 3](image)

5.7 Question 4

When the results are compared automatic updates are perceived as the most important implementation of security. Results are shown in figure 7.

![Figure 7: Question 4](image)
5.8 Question 5

The results from question 5 show that most of the respondents, 75%, believe that the resources are enough to authenticate with RSA according to the scenario. 7% answer no and 18% selected the option other.

Figure 8: Question 5

Are the given resources enough to run 1024-bit RSA authentication on the gateway on every data transfer?

- Yes: 73%
- No: 10%
- Other: 17%
6 Method - Performance Testing

To test the performance requirements of authentication algorithms, a series of tests have been run on a Raspberry Pi where CPU resources were restricted. The RSA algorithm was tested and evaluated on different CPU settings. The test was aimed to simulate a real world scenario, with a set number of sensors within the simulated network and how often they will send information to the Raspberry.

The test could show if the Raspberry is able to process all the authentication requests from the sensors before the next wave of data where sent. The memory usage of the process where also measured. The tests focused on the limitations of the RSA algorithm regarding CPU requirements. By doing this parallels could be drawn to how the algorithm would perform on real world devices.

6.1 Programming Language

The tests were implemented in the Go programming language. Go is relatively high performing whilst still being easy to use. The language also has an included RSA package which could be easily be implemented for the testing.

6.1.1 The RSA Package

This package implements RSA algorithms as defined in the Public Key Cryptography Standards (PKCS). It is possible to use RSA in this package to implement either public-key encryption or public-key signatures. For the testing PKCS was implemented.

6.2 Testing Machine

All the tests were run on a Raspberry Pi 3 model B. The Raspberry has a 4 core 2.4 GHz CPU and 1024 MB of RAM. During the tests the CPU was restricted to 100 MHz one core and 1024MB of RAM. Since no limit was reached with these settings the limit was estimated from the available data.

The Raspberry Pi was chosen as it is easily available, has relatively low performance and is easy to set up and run tests on. It would have been more optimal to choose a machine with even lower performance but this would also mean that the implementation of the testing would have been much more complicated.
6.3 Authentication Method

The authentication algorithm is based on the RSA PKCS. The authentication was performed by first signing then verifying a message. This was to emulate an authentication phase where the server, in this case a Raspberry Pi, first authenticates itself followed by a verification of the client’s signature. The time taken for sending, waiting for the client’s verification and the response of the client was disregarded in the testing.

6.4 CPU Limitation Methods

6.4.1 Cpulimit

To limit the CPU below the minimum required 100 MHz a program called Cpulimit was used. Cpulimit is a program that limits the CPU usage of a process. This is specified in percentage not in CPU time. The program pauses and resumes the process at the appropriate moments based on the given parameters from the user. This means that the program does not change the priority of the process.

6.4.2 CPU Governor

This method of limiting the CPU sets the maximum CPU by changing arm_freq value in /boot/config.txt. This file was run at boot and allows us to set the frequency of the CPU down to 100 MHz.

6.5 top

In order to monitor resource usage the built in task manager top was used for logging of CPU and memory data. This data was saved every 1 second and was used to verify that the resources used corresponded to the settings in Cpulimit.

6.5.1 Time Interval

The first intention was to log the data every 0.1 seconds, but this resulted in significant increase of the time that the test needed to complete. Several tests where made to find a threshold that could be used without compromising the results. It became clear that the test was unaffected or equally affected at 1-20
sec, but at 0.9-0.1 had a noticeable effect on the test. An conclusion was made that top have issues when using intervals more precise than a whole second.

### 6.6 Step-by-step Explanation of the Test

The tests were split into three parts. This was to test the accuracy of Cpulimit as performance limiting tool. Each part consisted of an authentication session, sign and verify, which was run 400 times for each setting. This test was in turn run 3 times each to ensure that no errors occurred during the test.

1. In the first part the governor was set to every 100 MHz between 1200 and 100 MHz and no Cpulimit was used.

2. The second part kept the CPU governor to 1000 MHz and lowered CPU resources using Cpulimit only. The reason for conducting this test was to be able to compare the results and in that way test the validity of Cpulimit as a performance limiting tool.

3. In the third part the CPU governor was set to the lowest value of 100 MHz and Cpulimit was used to lower resources further down to the lowest setting of 10%.

Each test was executed from a bash file to make sure that the RSA test and Cpulimit executed with minimal delay in between. The bash file started Cpulimit, the testing program and top.

1. In the bash file the % of current CPU resources to be used was set. This limited the amount of CPU % the test program could use. During the test top was used to log the resource usage of the process. This data was saved to a log file.

2. The test program ran the specified amount of tests sequentially in a for loop. Each test consisted of one sign-verify session using the PKCS RSA algorithm. Every test is timed and the times where used to calculate an average time for the authentication method.

3. After the tests the results were saved to a log file. With the average time for each authentication the limiting CPU or the limiting amount of authentication attempts for a specific CPU was able to be calculated.
7 Result - Performance Testing

This is an explanation of the headers in table 1 to 3.

**Average CPU usage:** This is the average CPU usage as measured during the test. This value is logged using top.

**CPU Clock setting:** This is the `arm_freq` value set in `/boot/config.txt`. This tells the CPU governor to limit the CPU MHz to the set value.

**Corrected Average CPU:** The calculated CPU usage in MHz of the test process, \( \text{AverageCPUusage} \times \text{CPUclocksetting} \).

**Average time:** This is the average time taken for one authentication, both signing and verifying a message.

**Cpulimit setting:** This is the value set by Cpulimit and puts an upper bound on the amount of resources available to the test process.

One should note that the value set by the Cpulimit tool differs somewhat from the value logged using top. This might be a limitation in top and/or a limitation of the logging. The value in the Corresponding CPU MHz column will be used for analysis since this allows a more fair comparison between the test both when using and not using Cpulimit.

The results are divided in three parts based on how the tests were conducted.
### 7.1 Results Without Using Cpulimit

By only using the CPU-governor setting in the Raspberry Pi the CPU clock frequency can be reduced down to 100 MHz. The lowest setting of 100 MHz resulted in an average CPU-usage of 89.127 % and an average time of 213.546 ms. This would be enough to authenticate 280 devices a minute without interruption.

<table>
<thead>
<tr>
<th>Average CPU [%]</th>
<th>Corr Average [MHz]</th>
<th>Average time [ms]</th>
<th>CPU Setting [MHz]</th>
</tr>
</thead>
<tbody>
<tr>
<td>87.394</td>
<td>1048.733</td>
<td>14.770</td>
<td>1200.000</td>
</tr>
<tr>
<td>84.700</td>
<td>931.700</td>
<td>16.200</td>
<td>1100.000</td>
</tr>
<tr>
<td>85.631</td>
<td>856.310</td>
<td>17.852</td>
<td>1000.000</td>
</tr>
<tr>
<td>85.599</td>
<td>761.388</td>
<td>19.887</td>
<td>900.000</td>
</tr>
<tr>
<td>91.456</td>
<td>731.644</td>
<td>22.386</td>
<td>800.000</td>
</tr>
<tr>
<td>90.772</td>
<td>635.405</td>
<td>25.624</td>
<td>700.000</td>
</tr>
<tr>
<td>90.883</td>
<td>545.300</td>
<td>30.200</td>
<td>600.000</td>
</tr>
<tr>
<td>91.877</td>
<td>459.384</td>
<td>36.290</td>
<td>500.000</td>
</tr>
<tr>
<td>91.967</td>
<td>367.867</td>
<td>45.734</td>
<td>400.000</td>
</tr>
<tr>
<td>91.287</td>
<td>273.861</td>
<td>62.331</td>
<td>300.000</td>
</tr>
<tr>
<td>92.205</td>
<td>184.410</td>
<td>96.466</td>
<td>200.000</td>
</tr>
<tr>
<td>89.127</td>
<td>89.127</td>
<td>213.546</td>
<td>100.000</td>
</tr>
</tbody>
</table>
7.2 Results With Cpulimit and CPU set to 1000 MHz

By setting the CPU governor to 1000 MHz and limiting resources using Cpulimit we are able to compare the results when limiting processor speed using Cpulimit and limiting by the CPU - governor only. This allows us to ensure that Cpulimit works as expected. In figure 9 the results from the two tests are plotted against each other. Both tests show similar results on how time increases with lower CPU setting.

The lowest setting we used on this test was 10 % using Cpulimit. This resulted in an average CPU usage of 13.999 % and an average time of 134.883 ms per authentication.

<table>
<thead>
<tr>
<th>Average CPU [%]</th>
<th>Corr Average [MHz]</th>
<th>Average time [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>88.657</td>
<td>886.571</td>
<td>19.725</td>
</tr>
<tr>
<td>80.150</td>
<td>801.500</td>
<td>21.962</td>
</tr>
<tr>
<td>70.852</td>
<td>708.519</td>
<td>24.743</td>
</tr>
<tr>
<td>65.260</td>
<td>652.600</td>
<td>27.069</td>
</tr>
<tr>
<td>55.408</td>
<td>554.083</td>
<td>31.949</td>
</tr>
<tr>
<td>45.982</td>
<td>459.822</td>
<td>38.950</td>
</tr>
<tr>
<td>35.988</td>
<td>359.877</td>
<td>50.037</td>
</tr>
<tr>
<td>25.594</td>
<td>255.936</td>
<td>70.385</td>
</tr>
<tr>
<td>13.990</td>
<td>139.896</td>
<td>134.883</td>
</tr>
</tbody>
</table>

Figure 9: Comparison Of Authentication Times With and Without Cpulimit
7.3 Results With Cpulimit and CPU set to 100 MHz

This test uses the lowest available CPU setting at 100 MHz in `boot/config.txt`. We also use Cpulimit to reduce performance further. The lowest setting is 10% Cpulimit which gives us an average authentication time of 1597.131 ms. With this performance we would be able to authenticate about 37 devices per minute using the RSA algorithm.

In figure 10 a comparison is made between the estimated authentication times vs the measured authentication times using Cpulimit. The results show that the authentication time when using Cpulimit is slightly below that of the projected times.

Table 3: 100 MHz, 90-10% with cpulimit

<table>
<thead>
<tr>
<th>Average CPU [%]</th>
<th>Corr Average [MHz]</th>
<th>Average time [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>81.721</td>
<td>81.721</td>
<td>234.636</td>
</tr>
<tr>
<td>75.960</td>
<td>75.960</td>
<td>253.171</td>
</tr>
<tr>
<td>68.521</td>
<td>68.521</td>
<td>281.435</td>
</tr>
<tr>
<td>60.198</td>
<td>60.198</td>
<td>319.303</td>
</tr>
<tr>
<td>51.311</td>
<td>51.311</td>
<td>376.265</td>
</tr>
<tr>
<td>42.936</td>
<td>42.936</td>
<td>451.552</td>
</tr>
<tr>
<td>33.251</td>
<td>33.251</td>
<td>584.344</td>
</tr>
<tr>
<td>23.783</td>
<td>23.783</td>
<td>819.999</td>
</tr>
<tr>
<td>12.337</td>
<td>12.337</td>
<td>1597.131</td>
</tr>
</tbody>
</table>

Figure 10: Comparison Of projected vs measured authentication times
8 Discussion

8.1 Performance Testing

The results show that RSA authentication is feasible depending on use case. The scenario requires that the gateway is able to handle 50 RSA authentication attempts per minute. With an average of 12.3 MHz used, the authentication process needs 1.6 seconds to handle one authentication. This restricts the device to handle 37 authentications per minute. This is 13 less than what the scenario requires.

This result fits well with the hypothesis on how the performance requirements would scale with the number of authentication requests. For devices who are very limited RSA authentication will be difficult even with a low number of authentication requests.

8.2 Survey

8.2.1 Number of Answers

Since the number of respondents is low, 29, it is difficult to draw far reaching conclusions from the answers. As the survey was anonymous there is no way to verify how many respondents came from certain companies or if the respondents are who they say that they are. Different security policies at different companies might therefore skew the results in a certain direction if employees from one company are highly represented in the study.
### 8.2.2 Demographics

**Gender**  The gender distribution in the survey is 90% male, 7% female, and 3% chose not to disclose their gender. These values are compared to the distribution in the sector according to statistics from Statistiska Centralbyrån SCB [13]. The sectors compared are chosen to be similar to the sectors surveyed in the study. From SCB, the distribution in the sector is 79% male and 21% female. This would mean that this study has a slight oversampling of males.

**Age**  The age group distribution in this study compared to the SCB statistics can be seen in Table 4. The comparison shows that the mean age in our study is lower than in the general target population.

<table>
<thead>
<tr>
<th>Age Group</th>
<th>% in Our Study</th>
<th>% SCB Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-29</td>
<td>17%</td>
<td>11%</td>
</tr>
<tr>
<td>30-39</td>
<td>45%</td>
<td>29%</td>
</tr>
<tr>
<td>40-49</td>
<td>28%</td>
<td>35%</td>
</tr>
<tr>
<td>50-59</td>
<td>3%</td>
<td>21%</td>
</tr>
<tr>
<td>60-69</td>
<td>7%</td>
<td>5%</td>
</tr>
<tr>
<td>Total</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

**Education**  The education of the respondents are generally collage or university which is to be expected from the target group.

**Profession**  Of all the respondents, 90% work in the IT-sector, 7% chose not to disclose, and one person is a student. Since the survey only was sent to IT-professionals, it is assumed that the one student works in the IT-sector, and therefore decided to keep that person’s answer in the study.

### 8.2.3 Scenario

The answers from the first question shows that the respondents rate poor password handling as the most common security flaw. The reason for when a user chooses an easily guessable password is probably because they have a hard time remembering it otherwise. This is the same reason for when a user writes down their password. Both of these behaviors could be solved with a multi-factor authentication system or an implementation of RSA. A hacker would then need to get a hold of the private key to use the stolen or guessed password.
Brute force attacks are the attack with the highest threat level for the given scenario according to the respondents. This also indicates that the authentication system with passwords is a valid attack point to gain access to the system. This is another reason to use a more advanced authentication system. The second highest threat is "Man in the middle" attacks which also could be solved with RSA. This means that the system would need to run one additional authentication turn. This would of course require more resources than doing this once.

In question four the respondents answered that if the system forces the user to use complex passwords or forces them to change password regularly would only give a low improvement of the security. This is probably because these security implementations do not help against brute force attacks, which was the attack with the highest threat level in our scenario. This also strengthens the need for a stronger authentication system.

The final question asks "Are the given resources enough to run 1024-bit RSA authentication on the gateway on every data transfer?". The results from the performance testing suggests that this would not be possible to do continuously. With 10% resources each authentication request would take too long time to complete and the authentication requests would eventually stack up.

The majority of the respondents believe that the implementation would be possible. This could be explained by the fact that it might be possible to implement the authentication algorithm more efficiently using more efficient versions of the RSA algorithm or a more low level programming language such as C or Assembly. Since the choice of programming language is not specified in the question this leaves room for differing interpretations.

Other ambiguities in the questions that respondents might interpret differently is the protocol used for data transfer and the type and size of the data sent.

An alternative to using RSA in the given scenario is to use ECC which is faster than RSA for a given key size. Another implementation is to use symmetric cryptography which is also faster. If the performance is very limiting an other option is to use a hardware implementation of RSA or ECC [6].

8.3 Suggestions for Future work

In the survey bad password handling by users where ranked high both in terms of commonality and in severity. This area might be investigated further to see if this might be the main reason for lacking security in IoT and how this would be solved.
In regards to authentication algorithms ECC could be tested in the same manner as in this study. It would also be interesting to see if the performance of the RSA algorithm could be improved using a more low level programming language such as Assembly or C.

A more extensive survey could shed more light on the most commonly experienced security threats. This would allow future work to be focused on the areas that are most important for IoT security.

8.4 Conclusion

"Is it possible to use the RSA algorithm for authentication when using a device with limited computational capabilities?"

It is not possible to use RSA in devices with limited computational capability that operate within the scenario provided. In this paper the threshold was shown to be 37 authentication attempts per minute with the lowest tested performance setting.

"Is there a disparity between the way authentication issues are perceived in the industry and the results from the performance tests in this paper?"

This paper are not able to conclude that a disparity exists between the results and the expectation in the industry due to a limited number of respondents. There is a disparity in the actual performance measured and the answers given in the survey. This could be due to different interpretations of the scenario provided in the survey.
References


A Appendix
Säkerheten inom Internet of Things

Vi är två studenter som läser Civ.ing inom Industriell ekonomi med inriktning datateknik vid KTH. Den här enkäten syftar till att undersöka vad professionella inom IT har för uppfattning om säkerheten i Internet of Things.

Enkäten består av två sektioner. Den första sektionen handlar er bakgrund. Dessa inledande frågor kommer att användas för att jämföra vårt urval med branschen i helhet.

Den andra sektionen handlar om datasäkerhet sett utifrån ett scenario.

*Required

Bakgrund

Alla era svar kommer att vara anonyma och kommer enbart att användas i forskningssyfte.

Dessa inledande frågor kommer att användas för att jämföra vårt urval med branschen i helhet.
1. När är du född? *

Mark only one oval.

- [ ] 1949 eller tidigare
- [ ] 1950
- [ ] 1951
- [ ] 1952
- [ ] 1953
- [ ] 1954
- [ ] 1955
- [ ] 1956
- [ ] 1957
- [ ] 1958
- [ ] 1959
- [ ] 1960
- [ ] 1961
- [ ] 1962
- [ ] 1963
- [ ] 1964
- [ ] 1965
- [ ] 1966
- [ ] 1967
- [ ] 1968
- [ ] 1969
- [ ] 1970
- [ ] 1971
- [ ] 1972
- [ ] 1973
- [ ] 1974
- [ ] 1975
- [ ] 1976
- [ ] 1977
- [ ] 1978
- [ ] 1979
- [ ] 1980
- [ ] 1981
- [ ] 1982
- [ ] 1983
- [ ] 1984
- [ ] 1985
- [ ] 1986
- [ ] 1987
- [ ] 1988
- [ ] 1989
Scenario

Datasäkerhetsfrågor
Svara på frågorna nedan efter utgångsläget i följande scenario.

100 MHz cpu samt 512 MB ram, där 10 % av enhetens resurser är reserverat för kommunikation med sensornerna. Nätverket består av 50 sensorer utspridda i en kontorsbyggnad. Varje sensor läser av temperaturen på luften och skickar data till vår gateway en gång varje minut.

5. Hur vanligt förekommande är det att ett företags implementation av vårt scenario har följande brister? *

Svara generellt sett utifrån givet scenario och svara vad ni TROR även om ni inte är helt säkra. *Mark only one oval per row.*

<table>
<thead>
<tr>
<th></th>
<th>Ej förekommande</th>
<th>Sällan förekommande</th>
<th>Förekommande</th>
<th>Vanligt förekommande</th>
<th>Vet ej/Obestämd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buggar och säkerhetshål på grund av ej uppdaterad mjukvara.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Låg komplexitet på lösenorden (Ej stora/små bokstäver och/eller inga specialtecken.)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dålig lösenordshantering av användarna (ex: sparas synligt eller lösenord som är lätt att gissa ).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ingen autentisering mellan gateway och sensor på grund av begränsade resurser i gateway.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ej krypterad kommunikation med vår gateway på grund av begränsad beräkningskapacitet i gateway.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tillverkarens standardlösenord har ej ändrats på gateway.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6. Givet att följande säkerhetsbrister existerar i scenariot, hur stor påverkan har de på säkerheten? *  
Svara generellt sett utifrån givet scenario och svara vad ni TROR även om ni inte är helt säkra. *Mark only one oval per row.*

<table>
<thead>
<tr>
<th>Låg komplexitet på lösenorden (Ej stora/små bokstäver och/eller inga specialtecken.)</th>
<th>Ingen påverkan</th>
<th>Låg påverkan</th>
<th>Hög påverkan</th>
<th>Väldigt hög påverkan</th>
<th>Vet ej/Obestämd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tillverkarens standardlösenord har ej ändrats på gateway.</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
</tr>
<tr>
<td>Ingen autentiserings på grund av begränsade resurser i gateway.</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
</tr>
<tr>
<td>Dålig lösenordshantering av användarna (ex: sparas synligt eller lösenord som är lätt att gissa.).</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
</tr>
<tr>
<td>Ej krypterad kommunikation på grund av begränsade resurser (CPU, minne) i gateway.</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
</tr>
<tr>
<td>Buggar och säkerhetshål pga ej uppdaterad mjukvara.</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
</tr>
</tbody>
</table>

7. Hur stora är dessa hot för säkerheten för givet scenario? * 
Svara generellt sett utifrån givet scenario och svara vad ni TROR även om ni inte är helt säkra. *Mark only one oval per row.*

<table>
<thead>
<tr>
<th>Malware</th>
<th>Ingen hot</th>
<th>Litet hot</th>
<th>Stort hot</th>
<th>Väldigt stort hot</th>
<th>Vet ej/Obestämd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute force attacks</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
</tr>
<tr>
<td>Man in the middle och eavesdropping</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
</tr>
<tr>
<td>DoS attacker</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
</tr>
</tbody>
</table>
8. Hur förbättras säkerheten i vår gateway av följande lösningar? *
Svara generellt sett utifrån givet scenario och svara vad ni TROR även om ni inte är helt säkra.
Mark only one oval per row.

<table>
<thead>
<tr>
<th>Sätt för att förbättra säkerheten</th>
<th>Ingen förbättring</th>
<th>Låg förbättring</th>
<th>Hög förbättring</th>
<th>Väljligt Hög förbättring</th>
<th>Vet ej/Obestämd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Användning av assymetrisk kryptering mellan gateway och sensorer.</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Tvinga användaren att byta lösenord vid setup av gateway.</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Krav på komplexa lösenord.</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Automatiska uppdateringar av mjukvaran (Detta för att kända säkerhetshål stängs snarast möjligt).</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Tvinga användaren att regelbundet byta lösenord.</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

9. Är de angivna resurserna tillräckliga för att driva 1024-bits RSA autentisering vid varje datamottagning i gateway?
Svara generellt sett utifrån givet scenario och svara vad ni TROR även om ni inte är helt säkra.
Mark only one oval.

- Ja
- Nej
- Other:
B Appendix
Säkerheten inom Internet of Things

29 responses

Bakgrund

När är du född? (29 responses)

Vilket kön har du? (29 responses)

Ange högsta utbildning (29 responses)
Scenario

Datasäkerhetsfrågor

Hur vanligt förekommande är det att ett företags implementation av vårt scenario har följande brister?

Givet att följande säkerhetsbrister existerar i scenariot, hur stor påverkan har de på säkerheten?
Hur stora är dessa hot för säkerheten för givet scenario?

- Malware
- DoS attacker
- Man in the middle och eavesdropping
- Brute force attacks

Hur förbättras säkerheten i vår gateway av följande lösningar?

- Användning
- Tvinga an...
- Automatis...

Är de angivna resurserna tillräckliga för att driva 1024-bits RSA autentisering vid varje datamottagning i gateway?

(29 responses)
Number of daily responses

This content is neither created nor endorsed by Google. Report Abuse · Terms of Service · Additional Terms

Google Forms
C Appendix

package main

import (
    "bufio"
    "crypto"
    "crypto/rand"
    "crypto/rsa"
    "crypto/sha256"
    "crypto/x509"
    "encoding/pem"
    "flag"
    "fmt"
    "io/ioutil"
    "log"
    "os"
    "os/exec"
    "runtime"
    "runtime/pprof"
    "strconv"
    "strings"
    "sync"
    "time"
)

//Constants, used for test settings and for storing data
const (
    RSAEncryptionStrength int = 1024
    RSAPubkeyFilename string = "RSA_pubkey.pub"
    RSAPrivkeyFilename string = "RSA_privkey.key"
    Message string = "golang"
    RSAtests = 400
    processName string = "rsa"
    crypt = crypto.SHA256
)

// Only used for starting and stopping logging of data.
var startOfTest = ""
var endOfTest = ""

/*
Runs setup and the tests.
Generation of keys can be omitted but must in that case be loaded from file.
*/
func main() {
    reader := bufio.NewReader(os.Stdin)
    fmt.Print("Enter to start: ")
    reader.ReadString('
')
    //pub, priv :=
    //generatePublicPrivateKey(RSAEncryptionStrength)
    //Only needs to run once to create keyes
    //saveKeyesToFile(pub, priv)
    pub, priv := getKeyes()
    startOfTest = time.Now().Format("15:04:05")
    testAuthentication(pub, priv, RSAtests)
    endOfTest = (time.Now().Add(time.Duration(1) *
        time.Second)).Format("15:04:05")
}

// Generates a signature and verifies using PKCS
func signVerifyPKCS(p *rsa.PublicKey, k *rsa.PrivateKey, mess []byte) {
    rng := rand.Reader
    hash := sha256.Sum256(mess)
    sig, err := rsa.SignPKCS1v15(rng, k, crypt, hash[:])
    if err != nil {
        log.Fatal(err)
    }
    err = rsa.VerifyPKCS1v15(p, crypt, hash[:], sig)
    if err != nil {
        log.Fatal(err)
    }
}

// Generates keyes and saves to file, RSA only. Takes key length as argument
func generatePublicPrivateKey(bit int) (interface{}, *rsa.PrivateKey) {
    k, err := rsa.GenerateKey(rand.Reader, bit)
    if err != nil {
        log.Fatal(err)
    }
    return k.Public(), k
}

// Saves generated keyes to a file
func saveKeyesToFile(pub interface{}, priv *rsa.PrivateKey) {
    privkey := x509.MarshalPKCS1PrivateKey(priv)
    pubkey, err := x509.MarshalPKIXPublicKey(&priv.PublicKey)
if err != nil {
    log.Fatal(err)
}

privBytes := pem.EncodeToMemory(&pem.Block{
    Type: "RSA PRIVATE KEY",
    Bytes: privkey,
})

pubBytes := pem.EncodeToMemory(&pem.Block{
    Type: "RSA PUBLIC KEY",
    Bytes: pubkey,
})
ioutil.WriteFile("data/"+RSAPubkeyFilename, pubBytes, 0644)
ioutil.WriteFile("data/"+RSAPrivkeyFilename, privBytes, 0644)

//Reading keyes from file and returns private and public key, RSA only
func getKeyes() (interface{}, *rsa.PrivateKey) {
    pubfile, err := ioutil.ReadFile("data/"+RSAPubkeyFilename)
    if err != nil {
        log.Fatal(err)
    }
    pubDecoded, _ := pem.Decode(pubfile)
    pub, err := x509.ParsePKIXPublicKey(pubDecoded.Bytes)
    if err != nil {
        log.Fatal(err)
    }

    privfile, err := ioutil.ReadFile("data/"+RSAPrivkeyFilename)
    if err != nil {
        log.Fatal(err)
    }
    privDecoded, _ := pem.Decode(privfile)
    priv, err := x509.ParsePKCS1PrivateKey(privDecoded.Bytes)
    if err != nil {
        log.Fatal(err)
    }

    return pub, priv
}

/*
Runs tests, times the encryption and decryption each time.
*/
Number of tests run is decided by RSAtests constant.

```go
func testAuthentication(pub interface{}, priv *rsa.PrivateKey, num int) {
    timeData := make([]float64, 0, num)
    pubKey := pub.(*rsa.PublicKey)
    mess := []byte(Message)
    for i := 0; i < RSAtests; i++ {
        startAuthentication := time.Now()
        //encryptDecryptOAEP(pubKey, priv, mess)
        signVerifyPKCS(pubKey, priv, mess)
        timeData = append(timeData,
                          time.Since(startAuthentication).Seconds())
    }
    storeData(timeData)
}
```

// Returns process pid from process name.
```go
func getPidFromProcessName(nameOfProcess string) string {
    out, err := exec.Command("pgrep", processName).Output()
    if err != nil {
        log.Fatal(err)
    }
    pidArr := strings.Split(string(out), "\n")
    if len(pidArr) != 2 {
        log.Fatal("Warning mer en process med namnet: ", processName)
    }
    return pidArr[0]
}
```

/*
Stores data to log file data/log.txt
Takes time data from each test case and calculates total and average time.
Gets PID of the program and uses this to parse data/top_data.txt
*/
```go
func storeData(timeData []float64) {
    system := runtime.GOOS
    numCores := runtime.NumCPU()
    timeDataString := ""
    var totTime float64
    var averageTime float64
    pid := getPidFromProcessName(processName)
    fmt.Println("PID")
    avgCPU, avgMem := parser(pid, startOfTest, endOfTest)
```
for _, td := range timeData {
    totTime += td
}
if err != nil {
    log.Fatal(err)
}
for _, v := range timeData {
    timeDataString = timeDataString +
        strconv.FormatFloat(v, 'f', 20, 64) + " "
}

averageTime = totTime / RSAtests
if err != nil {
    log.Fatal(err)
}
log.SetOutput(logfile)
log.Println(system, numCores, strings.TrimSpace(string(cpuP)), RSAEncryptionStrength, avgCPU, avgMem, totTime, averageTime, timeDataString)

fmt.Println("PID=", pid)
var cpuStats []float64
var memoryStats []float64

for scanner.Scan() {
    line := scanner.Text()
    words := strings.Fields(line)
    if len(words) > 0 {
        if words[0] == "top" && words[2] == clockStart && !activated {
            activated = true
        }
        if words[0] == "top" && words[2] == clockEnd && activated {
            activated = false
        }
        if words[0] == pid && activated {
            number, _ := strconv.ParseFloat(strings.Replace(words[8], ",", ".", -1), 64)
            cpuStats = append(cpuStats, number)
            memNumber, _ := strconv.ParseFloat(strings.Replace(words[9], ",", ".", -1), 64)
            memoryStats = append(memoryStats, memNumber)
        }
    }
}

var totalCPU float64
for i := 0; i < len(cpuStats); i++ {
    totalCPU += cpuStats[i]
}

avgCPU :=
    strconv.FormatFloat(totalCPU/float64(len(cpuStats)), 
    'f', -1, 64)

var totalMem float64
for i := 0; i < len(memoryStats); i++ {
    totalMem += memoryStats[i]
}
avgMem :=
    strconv.FormatFloat(totalMem/float64(len(memoryStats)),
    'f', -1, 64)

if err := scanner.Err(); err != nil {
    log.Fatal(err)
}

fmt.Println(avgCPU, avgMem)
return avgCPU, avgMem

D Appendix

#!/bin/bash

top -u pi -d 0.1 -b > data/top_data.txt &
go run rsa.go &
cpulimit -e rsa -l 10