KU LEUVEN

FACULTY OF ENGINEERING TECHNOLOGY

**TECHNOLOGY CAMPUS DE NAYER**

# Design of a pairing protocol for the AR.Drone 2.0

**Thomas BERTELS**

Promotor(en): Pr.Dr.Ir. G. De Samblanx

Co-promotor: E. Marin

S. Iraklis

Master Thesis submitted to obtain the degree of Master of Science in Engineering Technology: Elektronica-ICT afstudeerrichting ICT

Academic Year 2015 - 2016

# Acknowledgements

I look back at the process of the creation of this thesis with enormous gratitude and satisfaction. It was not always easy to find the motivation I needed after we encountered a lot of unforeseen problems. Therefore, I want to grab this opportunity to thank the people who supported and helped me creating a thesis I can be proud of.

At first, I want to thank my two co-promoters, Eduard and Iraklis. They provided me with literature, connected me with experts in the field and made notes on my text. I want to thank COSIC for providing me with the drone I needed for this research.
I also want to express my gratitude to my promotor Gorik De Samblanx for his support, his insights on the topic and for reading my text.

Finally I want to thank my family and my girlfriend who motivated and supported me from the sideline.

Thomas Bertels

# Abstract

This thesis treats the security aspect of commercial drones controlled over WIFI. More specifically, the research concentrates on the AR.Drone 2.0 from Parrot. The first part starts with an overview of the research on the drone that has been done in academic circles, with respect to the security aspect. The lack of security had already been revealed, but the effort made to solve this issue was small. In the third and fourth chapter, a brief introduction on cryptography is presented, together with a review of existing pairing protocols and analysis for their suitability on our drone. Respecting the available I/O on the drone, a protocol has been proposed that enables the user to setup a secure connection with the controller. In the last chapter of the thesis, a description of the implementation process is given.

# Abstract

Deze thesis behandeld het veiligheidsaspect van commerciële drones bestuurd met WIFI. Meer specifiek is ons onderzoek gericht op de AR.Drone 2.0 van Parrot. Het eerste deel begint met een overzicht van onderzoek op vlak van security, dat al verricht is in academische kringen. Het gebrek aan veiligheid was al blootgelegd maar er waren nog maar weinig pogingen ondernomen om dit op te lossen. In het derde hoofdstuk geven we een korte inleiding over cryptografie. In het vierde hoofdstuk geven we een overzicht van bestaande pairing protocollen en gaan we na of deze implementeerbaar zijn op onze drone. In functie van de aanwezige in- en uitvoer op de drone hebben we een protocol voorgesteld dat de gebruiker in staat stelt om een beveiligde verbinding tot stand te brengen tussen de drone en de controller. In het laatste hoofdstuk beschrijven we het voorloop van deze implementatie.

# Samenvatting

## Hoofdstuk 1: Inleiding

Drones zijn tegenwoordig alom aanwezig in onze maatschappij. Het is een populair geschenk voor onder de kerstboom maar ook bedrijven maken er meer en meer gebruik van. Dit succes heeft er toe geleid dat bedrijven steeds betere drones op de markt brengen tegen een lagere prijs. We moeten vaststellen dat de beveiliging niet op dezelfde manier geëvolueerd is, en dit heeft o.a. gevolgen naar onze privacy. We proberen in deze thesis een analyse te maken van de problemen en de gevolgen daar aan verbonden, en zullen een voorstel doen om deze drones beter te beveiligen. We specificeren on op de AR.Drone 2.0 van Parrot.

## Hoofdstuk 2: Literatuurstudie

Enkele onderzoeken naar dit type van drone waren reeds uitgevoerd. Wetenschapper brachten reeds aan het licht dat er verschillende kwetsbaarheden aanwezig waren aan de drone. Specifiek kaarten ze twee gebreken aan. Het eerste gebrek ontstaat doordat de drone zich enkel bind aan zijn controller door middel van het controleren van het fysieke adres. Dit adres is echter makkelijk te imiteren, wat tot gevolg heeft dat een derde partij de controle van de drone kan overnemen. Een tweede gebrek ontstaat door het feit dat de drone Root toegang verschaft via het Telnet protocol zonder wachtwoordbeveiliging. Dit heeft als gevolg dat een derde partij de software op de drone kan aanpassen, maar ook de data afkomstig van de camera's kan onderscheppen zonder medeweten van de eigenlijke

gebruiker. Deze technieken werden reeds geautomatiseerd en gedeeld op Github. Om al deze problemen te voorkomen wordt voorgesteld om het data verkeer te versleutelen op fysiek niveau.

# Hoofdstuk 3: Cryptografie: Een korte inleiding

In dit hoofdstuk geven we een korte inleiding over cryptografie, die nodig is om de rest van de tekst te begrijpen. We maken onderscheid tussen symmetrische cryptografie, waar gebruikt maakt van één sleutel, en asymmetrische cryptografie, waar we gebruik maken van een sleutelpaar. Als laatste wordt er verder in gegaan op hash functies.

# Hoofdstuk 4: Pairing protocols

Om het data verkeer te versleutelen hebben beide partijen een identieke sleutel nodig waar enkel zij kennis van hebben. Voor de uitwisseling van deze sleutel gebruiken we pairing protocollen. In dit hoofdstuk bespreken we enkele bestaande protocollen, alsook het OOB kanaal. Dit is een extra communicatiekanaal waar deze protocollen gebruik van maken, en waarvan we zeker zijn dat deze vertrouwelijk en/of authentiek zijn.

# Hoofdstuk 5: Ontwerp van ons pairing protocol

Om zelf een pairing protocol te ontwerpen moeten we eerst op de hoogte zijn van de aanwezige hardware op de drone. De belangrijkste eigenschappen van de drone zijn de ARM processor, de WIFI interface en de aanwezige sensors, zoals 2 camera's, accelerometers en hoogtemeters. De aanwezige hardware bepaald de keuze van het OOB, wij hebben voor het visuele kanaal gekozen. Hier gaan we uit dat de camera van de drone het scherm van de controller kan lezen. We stellen voor om de sleutel volledig over dit OOB te versturen, hierdoor vermijden we het gebruik van asymmetrische cryptografie. Voor de te verdelen sleutel gebruiken we de key derivation functie gedefinieerd door de WPA stan-

daard. Dit beperkt de over te dragen data tot 8 - 13 byte. Verder stellen we voor om deze sleutel te encoderen in een QR code, dit zal de beeldherkenning vergemakkelijken en voorziet ingebouwde error correctie.

# Hoofdstuk 6: Implementatie

In dit hoofdstuk bespreken we hoe de implementatie van ons protocol tot stand is gekomen. We delen deze implementatie op in 3 delen; we bespreken eerst het opzetten van een beveiligd WIFI access point op de drone, daarna bespreken we de applicatie op de controller en we eindigen met de software die de QR code moet herkennen.

Op een linux systeem gebruiken we de software 'hostapd' om een beveiligd access point op te zetten. Door problemen met de drivers zijn we hier echter niet in gelukt. Het was echter wel mogelijk om de drone te laten verbinden met een beveiligd access point. We hebben hierdoor in ons protocol de rol van client en server omgedraaid.

Waar we voor onze controller zelf een smartphone gebruikten, hebben we een android applicatie gemaakt. Deze applicatie is in staat om een willekeurige sleutel te genereren en deze te coderen in een QR-code. Verder begeleid de applicatie de gebruiker om een access point op te zetten.

Het laatste onderdeel van onze implementatie moest de decodering van de camera verzorgen, hier ondervonden we echter enkele problemen. Waar de firmware van de drone ons de toegang tot de camera ontzegde, konden deze wel bereiken via een FTP stream bedoeld voor de applicatie van Parrot. Hiervoor gebruikten we opencv. We waren in staat om enkele beelden te schieten van onze QR code en hier simpele bewerkingen op uit te voeren. Bij de integratie van een bibliotheek voor het decoderen van QR codes liep het echter mis. We waren niet in staat om deze bibliotheken te cross compileren voor gebruik op het ARM platform.

# Hoofdstuk 7: Conclusie

De drone van Parrot vertoont enkele belangrijke beveiligings gebreken. We hebben in ons onderzoek een voorstel gedaan om de AR.Drone 2.0 beter te beveiligen. Ons protocol zal de privacy van de gebruiker en de integriteit van de drone zeker ten goede komen. We zijn er echter niet in geslaagd om dit te implementeren op de drone zelf.

# Contents

# List of Figures

# List of Tables

# List of symbols and Acronyms

## Symbols

| | |
|---|---|
| $\dashrightarrow$ | data sent via a wireless insecure channel |
| $\longrightarrow$ | data sent via an out-of-band channel |
| $CV_K(D)$ | check-value function computed on the message m using the key k |
| $X \in_U \{0,1\}^k$ | random bitstring X of length n |
| $a\|\|b$ | concatenation of the bitstrings $a$ and $b$ |
| $a \oplus b$ | exclusive OR (XOR) of a and b |
| $h(D)$ | a cryptographic hash function computed on the data string D |
| $E_K(D)$ | symmetric encryption of the message m using the key $K$ |

# Acronyms

| | |
|---|---|
| AES | Advanced Encryption Standard |
| DES | Data Encryption Standard |
| FTP | File Transfer Protocol |
| GPS | Global Positioning System |
| IP | Internet Protocol |
| IV | Initial Value |
| NFC | Near Field Communication |
| OOB | Out-of-Band |
| PIN | Personal Identification Number |
| RSA | Rivest-Shamir-Adleman |
| SHA | Secure Hash Algorithm |
| SSH | Secure Shell |
| SSID | Service Set Identifier |
| WLAN | Wireless Local Area Network |
| WPA | WI-FI Protected Access |
| XOR | Exclusive OR |

# Chapter 1

# Introduction

Drones are unmanned aerial vehicles that are remotely controlled from short to long distances depending on the application. They typically contain a navigation system (GPS), a visual and audio interface, and a variety of different sensors. Historically, drones have been used for military purposes, such as surveillance. However, nowadays drones can also be used to support critical services such as forest fire and illegal hunting detection, search and rescue operations, or delivering medical supplies to remote/inaccessible regions.

The AR.Drone 2.0 is one of the most popular commercial drones. Previous research indicates that there is a serious security issue to this device. First, the images captured by the camera of the drone are not encrypted and can be easily intercepted by a third party. Second, by spoofing your mac-address, you can take over the controls of the drone. Both issues are the consequence of an insecure connection between the device and the controller.

In the scope of this thesis, we would like to introduce you to pairing protocols and use this to encrypt the communication between device and controller. We will evaluate them on their security level and implement them on the drone.

# Chapter 2

# Related work

The AR.Drone 2.0 is a widely available commercial drone with great specifications for his price and has a big community. The SDK[1] offered by ParrotParrot (2016b) make it even more suitable for research and education purposes. For this reasons, thorough security analysis has already been done by a few people. Their findings indicates that Parrot didn't made security a priority. We will discuss their findings here.

The lack of security was already widely discussed on the Parrot community forums, but was first academically publicized by Samland et al. (2012) While they did their research on the AR.Drone (and not the AR.Drone 2.0), their results are representative to our use because of the minor changes Parrot implemented. For their security analysis approach, they used the BSI standard 100-2 (Federal, 2008) adapted for embedded devices. They just listed the different components of the drone with their potential security aspect (availability, integrity, ...) and protection requirements. They proposed security enhancements for the components, ranging from increasing the redundancy of the sensors to dedicated partition of the memory. More interesting is their analysis about WLAN and Telnet and FTP protocols. They discover that the WLAN is not encrypted and the Telnet and FTP port are left unprotected.Therefore, every person with access to the drone's network can exploit this to upload and execute malicious code. To solve this problem, they suggest to use WPA2

[1] Software development kit

encryption over the WLAN to prevent unauthorised access to the drone. Furthermore, they state that the Telnet and FTP protocol should be replaced by the SSH protocol in case the WPA2-key gets leaked. Securing Telnet and FTP with a password will not suffice as these passwords are transmitted in plain text.

They experience that the drone is secured by remembering the MAC-address of the controller first connected to it. As it is easy to alter your MAC-address[2], this results in another vulnerability. Note that an attacker must not change his MAC-address to access the drone's network. They give three examples of exploitations of these vulnerabilities.

The first is hijack the drone, they do this as follows:

- The attacker can connect to the drone's network and setup a root-shell using Telnet. This is possible because there is no access control implemented in the drone, no encryption on the link layer and no passwords on the Telnet. However, in this stage it's still impossible to control the drone because the drone and the controller are paired. The attacker needs to make his device look like the controller's device.
- To imitate the controller, we need to obtain his MAC- and IP'address. This can be easily done because we have root access to the drone's Linux operating system. We can also use a network sniffer, for example Wireshark, to obtain this information.
- The attacker has to alter his MAC-address into the obtained MAC-address of the controller's network adapter. By doing this, the drone will identify the attacker as the paired control device when a new connection is established.
- To establish a new connection, the attacker can change the SSID[3] of the drone's network using his root privileges. This will result in the disconnection of the paired control device. If he connect his device now to the drone's access point with a new SSID, it will be recognized as the original controlling device.

[2] In linux, with the command "ifconfig 'interface' hw ether xx:xx:xx:xx:xx:xx", you can alter your MAC-address.

[3] Session ID, the public name of an access point

- All the attacker has to do now is change his IP-address into the address of the original controller, with the drone's incapability to see the difference between the original controller and the attacker's device as result.

In the second attack they explain, they try to intercept the video signals sent by the drone. The methods are quite similar. First they connect to the drone's network, but instead of disconnecting the authentic controller, they upload malicious code to the drone using the open FTP connection. This code can be executed via Telnet, to access the camera's interface and stream the images to the attacker without raising suspicion.

The third attack states that the drone can be equipped with a smartphone. Their tests revealed that the drone can carry an additional 120 grams without the controls being influenced. The drone sees the smartphone as primary controller, but the actual commands come from a computer, connected to a proxy on the smartphone. In this setup, the computer can send commands to the drone, depending on the received video-stream from the drone and the GPS coordinates from the smartphone. With smart software on the computer, the drone can, for example, be used to track people and map their movements. This is an attack on our privacy using the drone, but not an attack on the drone itself.

The paper concludes that the most of the security issues can be addressed by implementing link-layer encryption.

Another paper, wrote by Pleban et al. (2014), perform similar attacks on the AR.Drone 2.0. They started their investigation with a port scan on the unencrypted network using Nmap[4]. Except for the already known ports 21 (FTP) and 23 (telnet), six other open ports where discovered in the 5xxx range. They are shown in table 2.1.

The AT commands are part of the protocol, custom built by Parrot, to steer the drone. Furthermore they listed the critical shell scripts that can be altered, and they explained

[4]Nmap is an open-source network scanner used to map all services and hosts on a network.

**Table 2.1:** Active ports on the AR.Drone 2.0, reproduced from Pleban et al. (2014)

| Port | Explanation |
|---|---|
| 21 (TCP) | FTP Server which serves video and image files recorded by the drone |
| 23 (TCP) | Telnet Server offering a root shell |
| 5551 (TCP) | FTP access to the update folder for the purpose of firmware updates |
| 5553 (TCP) | VIDEO: The H265-720p frames of the camera are available here if the phone application is recording |
| 5554 (UDP) | NAVDATA: Current telemetry data (status, speed, rotor speed) is sent to the client here (15cmds/s demomode, 200 cmds/s full debug mode). |
| 5555 (TCP) | VIDEO: The video stream of the drone is available to clients here |
| 5556 (TCP) | ATCMD: The drone is controlled in the form of AT commands. These control commands are send periodically to the drone (30 cmds/s). |
| 5559 (TCP) | CONTROL port: Some critical data, sush as configurations are transferred here. |

why UDP is not a secure choice to send commands to the drone. The attacker scenarios described are similar to the ones previous seen. They propose (and implement) a way to secure the drone's network. To achieve this, they let the controller be the server of the network and let the drone connect to this network as a client. The controller device (smartphone, tablet or computer) has the ability to set up a secured access point implementing WPA or WPA2. For the drone to be able to connect to the secure network, a wpa_supplicant (Malinen, 2013) is needed. They cross-compiled this software for the ARM platform on the drone. To make the connection set up from the drone to the controller, they need a third party. When the drone powers up and sets up an insecure access point, the third device connects to the drone, uploads the wpa_supplicant, gives the parameters of the secure network of the controller (password, SSID) to the drone, and executes the code. This results in the deactivation of the insecure network and connection to the secure network owned by the controller. A secure connection is then established. Note that the connection between the third device and the drone is still insecure. If an attacker can sniff this communication, he is able to deduce the WPA2 key set by the controller's access point and listen to further communication.

For future work, they state that the third device used to setup the secure communication, should be eliminated. They propose that this can be done by pairing the controller to the

drone by touching it using Near Field Communication (NFC).

Finally we would like to introduce the SkyJack-Hack (Kamkar, 2013) designed by Samy Kamkar. They made a drone that automatically detects all the Parrot drone's in his range and use the known vulnerabilities to take over their controls. They achieve this by attaching a Raspberry Pi with an extra network adapter to a drone. While this network adapter is in monitoring mode[5], they sniff the network for MAC-addresses that belongs to Parrot. The range of MAC-addresses owned by Parrot can be found at `http://standards-oui.ieee.org/oui/oui.txt`. After they identified all the vulnerable drone's, they can use the attack scenarios described above. Note that his software can also be used from a ground station. All the code and demos can be found on his GitHub.

[5] When a network adapter is set to 'monitoring mode', it is capable to sniff packets without being connected to an access point.

# Chapter 3

# Cryptography, a brief introduction

Before explaining pairing protocols, we will give a brief introduction on cryptography. Cryptography is the art of concealing a message so that only the designated party can read it. Classic cryptography goes back to the ancient Egypt, where they used deviated hieroglyphs. Caesar used the Caesar cipher named after him. This cipher rotates the alphabet with the key, a fixed number from 0 to 25, and replace every letter with the corresponding one. For example: if the key $k = 1$, 'a' will be replaced by 'b'. Table 3.1 shows an example of $k = 5$.

With the beginning of the first and mainly the second world war, the rotor machine became more popular. This typing machine lets you set a more complex key and transfers the plaintext you type in ciphertext with a more advanced cryptographic algorithm. Famous examples are the enigma and the Lorentz-machine. With the advent of the first computers, these machines and almost every form of encryption up until that point were broken. There was a need for better algorithms that provided resistance to the always increasing computer power. In modern cryptography, we like to divide these algorithms into three

**Table 3.1:** Caecar cipher with k=5

| plain: | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| cipher: | v | w | x | y | z | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u |

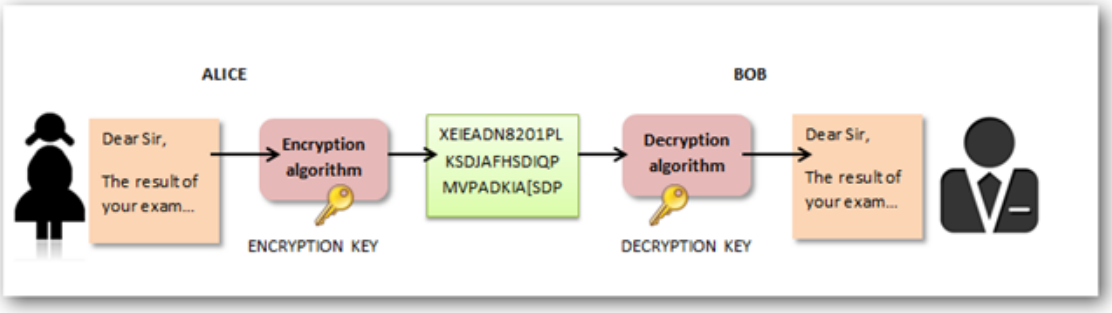


**Figure 3.1:** Symmetric encryption

categories.

## 3.1 Symmetric cryptography

With symmetric cryptography, two partys can exchange encrypted messages if they share the same key. This key will be used for both encrypting and decrypting, illustrated in figure 3.1. These algorithms are generally fast and can easily be applied to a bulk of data. The size of the key give a indication about the resistance against brute-force attacks[1]. The actual key strength is often lower than the key size because cryptanalysts find ways to reduce the number of combinations to be tested. It is safe to say that nowadays a key strength of 128bit is secure. The main issue with using symmetric cryptography is that we need a secure way to share a key with both parties. This need to be done authentically[2], confidentially[3] and the integrity[4] of the key needs to be preserved. AES (Daemen and Rijmen, 1999) is a symmetric algorithm that is widely distributed, has hardware support on popular platforms, and is still considered secure. The algorithm is also validated by NIST[5] (Nechvatal et al., 2000). DES and RC4 are examples of algorithms that are considered broken.

[1] A brute-force attack is a way to decipher a ciphertext by trying every possible key.
[2] Authentic: both parties need to be sure of the identity of the other party.
[3] Confidential: Both parties need to be sure that nobody else has can read the key.
[4] Both parties need to be sure that no one change the key
[5] the United States Government's National Institute of Standards and Technology

## 3.2 Asymmetric cryptography

With asymmetric encryption, we make use of a key pair. This key pair exists of a public key and a private key. This key pair has a relation to each other so that we can decrypt a message with one key and decrypt it with the other. If Alice want to send a message to Bob, Alice encrypts this message with Bob's public key and Bob can decrypt this message with his private key. Alice can also encrypt a message with her own private key and Bob can encrypt this with Alice's public key, we call this a signature. Both are illustrated in figure 3.2.

As long as we cannot derive the private key from the public key, we can say that the encryption is secure. This way, we can make our public key public and we only need to hide our private key. This is true because the key generation of the pairs is based on mathematical problems without an efficient solution. Examples are integer factorization, discrete logarithm and elliptic curve relationships. These mathematical problems are nevertheless easier to solve than brute forcing all possible options. Therefore, asymmetric keys are usually longer than symmetric keys, ranging from 1024bit (equivalent to 80bit symmetric key) to 15360bit (equivalent to 256bit symmetric key). Examples of asymmetric algorithms are RSA, ElGamal and Diffie-Hellman.

Whereas asymmetric cryptography require a lot more computing power to transfer a message, we usually use it to establish a session key that can be used as a symmetric key for further communication between two (or more) parties making use of a symmetric algorithm. This reduces the computational cost and provide a secure way to communicate. Another method to share a symmetric key is to make use of an OOB channel. This will be explained in section 4.1.

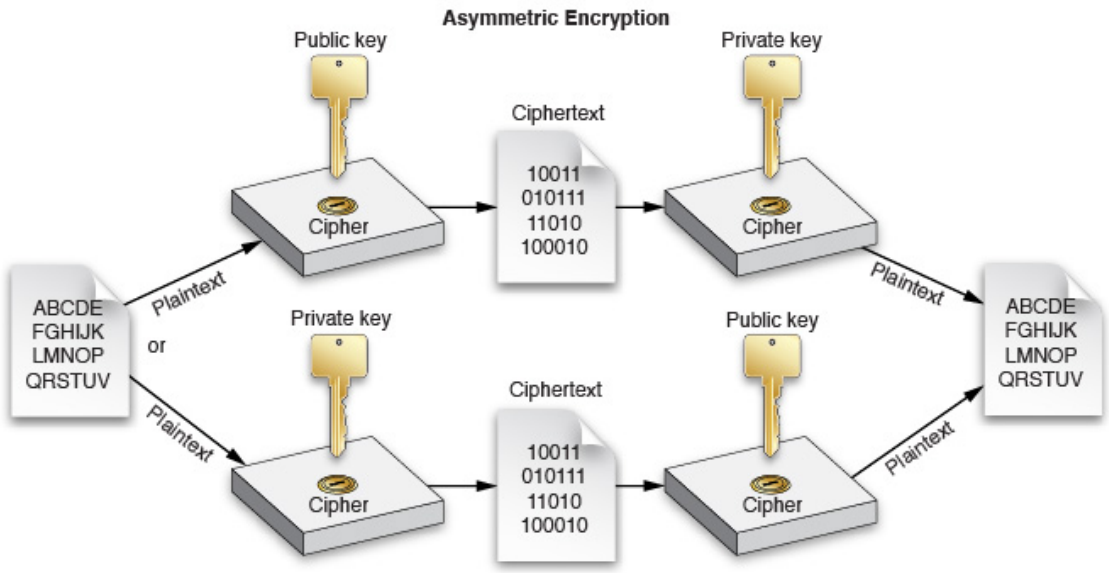


**Figure 3.2:** Asymmetric encryption

## 3.3 Hash functions

A hashing function is a function that create a fixed length output for a variable length input. The output of this function, called the hash of the input, appears to be random and is totally different from an input with a minor change. However, a hash function will always produce the same hash for identical inputs. Hashing functions are designed to compute fast and can be used to ensure the integrity of data. We give two examples:

- If Alice wants to send a message to Bob, she can calculate the hash of this message and deliver the hash and the message to Bob. Bob then calculates the hash of the message (with the same hashing function), and compares it with the hash received from Alice. If the hashes match, they possess the same message. Note that in this example it is important that Alice and Bob share the hash in a secure way.

- When you enter your password to log in into an online service, most of the time your password will get hashed before sending it to the webserver. The webserver that only possesses the hash of your password can then compare these two values and grant you access if they are identical. This has as advantage that we don't need to send our password in plain text over the internet and that the server does not have

to store our passwords in plain text. In case of a security breach, only the hash of our password can be stolen.

If an attacker can alter a message in a way that produces the same hash, we speak about a collision. The likeliness of this depends on the output size and the design of the hash function. Examples of hashing functions are MD5 and SHA.

# Chapter 4

# Pairing protocols

In this chapter we will first discuss what we understand by an OOB channel as this is an important part of pairing protocols. Secondly, we will give an overview of OOB channels and how they can be used for communicating. At last we will explain some pairing protocols

## 4.1 What is an OOB channel?

An Out-Of-Band (OOB) channel is a secondary channel between 2 devices with low bandwidth were they can exchange a number of bits in a private and/or authentic way. This data can be used sharing a public key, sharing a symmetric key or for key confirmation.

- **Sharing a public key:** If one device wants to share his public key to another device to setup a secure communication, they can do this over their insecure main communication channel. However, there is a risk that a third party will alter this key into his own. To prevent this, both devices can exchange their public key over the OOB. A public key does not need to stay confidential, so the OOB only needs to be authentic in this case.

- **Sharing a symmetric key:** If two devices want to share a symmetric key, they cannot do this over the insecure main communication channel. This is because this channel is not confidential and a third party can be eavesdropping. The symmetric

key needs to stay a secret between the two devices. If the OOB channel is authentic and confidential, both parties can exchange a key in plain text.

- **Key confirmation:** If both devices share a key, symmetric or asymmetric, they can check its integrity by sending a hash of this key over the OOB. This can also be used to reduce the bits send over the OOB while sharing their public key. They share this key over the insecure main communication channel and verify it by sending a hash of their key over the OOB. In both cases, the OOB only needs to be authentic if we use a secure hashing algorithm.

## 4.2 OOB channels in practice

We will now give an overview of different channels and how we can use them.

### 4.2.1 The resurrecting duckling

A simple solution to transfer a message between two devices is to make use of cabled connection, as suggested by Stajano and Anderson (2000). If a wire makes a direct electrical contact between two devices, it can be used to exchange information. This information is private, authentic and cannot be altered. We can also gain a high bandwidth. The disadvantage is that we need extra hardware (the wire).

### 4.2.2 User aided key exchange

The user can also be a medium. He can enter and compare values in both devices. We will now look at 2 protocols that need user interaction.

- **The MANual Authentication protocols**

  The first three MANA protocols where presented by Gehrmann and Nyberg (2004, 2001); Gehrmann et al. (2004) and later a fourth was added. The purpose of MANA protocols is to agree to a public set of bits, for example a public key. MANA1 was

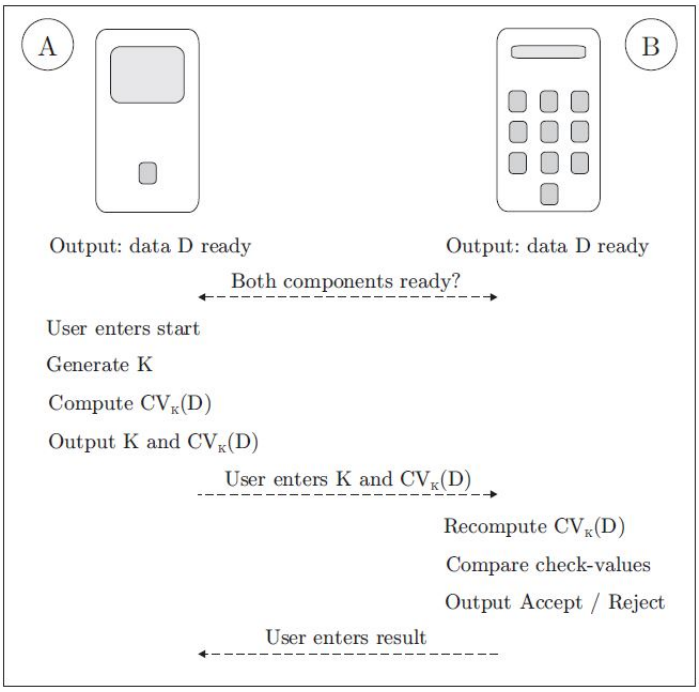


**Figure 4.1:** The MANA 1 protocol (Singelee, 2008)

designed specifically for one device with a screen and one device with a keyboard, whereas MANA2 requires both devices to have a screen. Finally MANA3 requires both devices to have a keyboard. We will briefly discuss the MANA 1 protocol in more detail.

Device A and B want to agree about data D. The former has a screen and the latter has a keyboard. Device A generates a key K and uses this key to compute a check-value function of D. The display shows K, and the result of this function. The user now needs to introduce these 2 values on device B. Device B then recomputes the check-value function and compares the value with the values keyboard-entered by the user. Finally, device B decides whether this value is correct, and the user enters the output into device A. Note that device A needs a small input and device B needs a small output for the last step.

- **Short authenticated string (SAS)**

  SAS was proposed by Vaudenay (2005). It is very similar to the MANA protocol but has some advantages. More specifically, the output that the user manually needs to

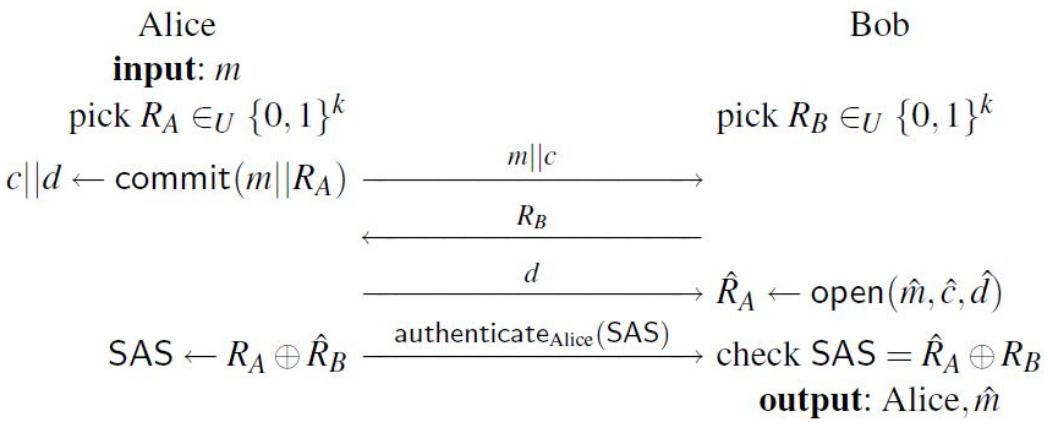


**Figure 4.2:** The SAS protocol (Vaudenay, 2005)

check is divided into two (without compromising the security of the scheme). The requirements of the authentication channel are also lower, where the MANA protocols needs a stall-free transmission[1], SAS does not.

In this scheme, input $m$ represents the message that will become authenticated and could represent a public key. Alice will first make a commitment of the message $m$ concatenated with a random generated key $R_A$ and sends this together with the message $m$ to Bob. When Alice receives a random generated key $R_B$ from Bob, she replies with the key $d$ to open her commitment. With this key, Bob can retrieve $R_A$. Both parties now possess $R_A$, $R_B$ and $m$. In the final step, Alice computes the SAS value ($R_A \oplus R_B$) and sends this over an authenticated channel to Bob. If Bob computed the same SAS value, he verified the identity of Alice.

### 4.2.3 Visual channel

Using the visual channel as a OOB channel was first proposed by McCune et al. (2009) They proposed to use the screen of a device to visually display the hash of a public key as a 2-D barcode and use a camera on a second device to read this information. This protocol is better known as SIB (seeing-is-believing). It provides a unidirectional authentication. If both devices have a camera and a display, there can be bidirectional authentication

[1] A transmission of data without any delay

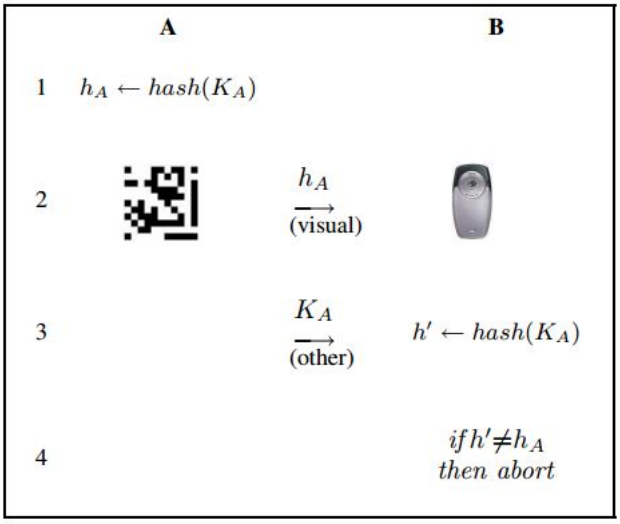


**Figure 4.3:** The SIB protocol (McCune et al., 2009)

1. $A \longrightarrow B$ (insecure channel): $K_A$
2. $A \longleftarrow B$ (insecure channel): $K_B$
   $A$ calculates $h_A$ as $h(K_A|K_B)$ and $B$ calculates $h_B$ as $h(K_A|K_B)$
3. $A \longrightarrow B$ (visual channel): $h_A$
   $B$ compares $h_A$ and $h_B$. If they match, $B$ accepts and continues. Otherwise $B$ rejects and aborts. In either case, $B$ indicates accept/reject to the user.
4. $A$ prompts user as to whether $B$ accepted or rejected. $A$ continues if the user answers affirmatively. Otherwise $A$ rejects.

**Figure 4.4:** The VIC protocol (Saxena et al., 2006)

achieved by executing SIB twice, one time in each direction. A scheme of SIB can be found in figure 4.3.

An improvement of this protocol was presented by Saxena et al. (2006) and is called Visual authentication based on Integrity Checking (VIC). This includes a way to achieve bidirectional authentication over an unidirectional channel. In this way, only one device needs a screen; the second device only needs a camera to achieve mutual authentication. VIC is explained in figure 4.4.

The paper also investigates how to apply this on devices with a small screen (i.e; low resolution) and even reduces the resolution to a blinking led. To make this work practically,

1. $A$ chooses a long random bit string $R_A$ and calculates $h_A$ as $h(R_A)$.
   $A \longrightarrow B$ (insecure channel): $h_A, K_A$
2. $B$ chooses its own long random bit string $R_B$
   $A \longleftarrow B$ (insecure channel): $R_B, K_B$
3. $A \longrightarrow B$ (insecure channel): $R_A$
   $B$ now computes $h'_A$ as $h(R_A)$ and compares it with the $h_A$ received in message 1. If they do not match, $B$ aborts. Otherwise $B$ continues.
4. $A$ calculates $hs_A$ as $hs(R_A, R_B, K_A, K_B)$ and $B$ calculates $hs_B$ as $hs(R_A, R_B, K_A, K_B)$
   $A \longrightarrow B$ (visual channel): $hs_A$
   $B$ compares $hs_A$ and $hs_B$. If they match, $B$ accepts and continues. Otherwise $B$ rejects and aborts. In either case, $B$ indicates accept/reject to the user.
5. $A$ prompts user as to whether $B$ accepted or rejected. $A$ continues if the user answers affirmatively. Otherwise $A$ rejects.

**Figure 4.5:** The VICsh protocol (Saxena et al., 2006)

they first need to reduce the number of transferred bits over the visual channel. For this reason they introduced a second protocol called VICsh (VIC with short checksum). This protocol is a merger between VIC and MA-3, an improved version of the MANA and SAS protocol. With this protocol, they can reduce the bits that needs to be exchanged to 15 to 20. The protocol is explained in figure 4.5.

### 4.2.4 Movement channel

Holmquist et al. (2001) suggest to use the presence of accelerometers to benefit from the common context between devices. They used the accelerometer data of two devices and compared them to establish if they are worn by the same person. Mayrhofer and Gellersen (2009) extended this research by developing an authentication and a pairing protocol based on movement data. In their first protocol, they try to authenticate two devices while shaking them together in one hand. First, they need to preprocess the data on both devices. They take the data acquisition, temporal alignment and special alignment into account. With this preprocessed data, each device executes a feature extraction to be-

come a bit-string of data. Now, both devices can compare this string and if these matches within a given error-rate, we can state the authentication was a success. The higher the error-rate, the more secure, but less user-friendly this protocol is, and vice versa. The second protocol try's to pair the both devices instead of only authenticating. They use they extracted features of the accelerometer data to negotiate a symmetric key. This is called the CKP (candidate key protocol). Their experiments show that we need to shake both devices for about 20 seconds to achieve a decent security level. This is not really user friendly.

### 4.2.5 Others

Of course there are many more OOB channels with the capacity to communicate. We can use IR sensors and align two devices to exchange information. This is not completely immune against a Man-In-The-Middle (MITM) attack and this technology is not widely present, because of its age. The audio channel also presents a few opportunities, two of them are described by Goodrich et al. (2005) and Soriente et al. (2007). The proposal from Pleban et al. (2014) to use NFC to pair with the drone is interesting because almost every smartphone nowadays has the ability to use NFC, but we will have to add a receiver for these signals since the drone is not equipped with it. An example of using NFC as pairing protocol is proposed by Amariucai et al. (2011).

## 4.3 Examples of pairing protocols

The goal is to share a key between two parties in a confidential and authentic way. This key can be used for encryption in further communication between the devices. If we exchange a key with a second party, one needs to be assured that this second party is the only one that can determine the key and that it is the party we think it is. We can make use of two communication channels. Firstly we have the high-bandwidth insecure channel. In the case of the AR.Drone this is the insecure WIFI. Secondly, there needs to be an auxiliary channel, called the out of-band channel, described in 4.1. This channel can be either

| Step | Alice | Bob |
|---|---|---|
| 1 | Parameters: $p$, $g$ | |
| 2 | $A = \text{random}()$<br>$a = g^A \pmod p$ | $\text{random}() = B$<br>$g^B \pmod p = b$ |
| 3 | $a \longrightarrow$<br>$\longleftarrow b$ | |
| 4 | $K = g^{BA} \pmod p = b^A \pmod p$ | $a^B \pmod p = g^{AB} \pmod p = K$ |
| 5 | $\longleftarrow E_K(data) \longrightarrow$ | |

**Figure 4.6:** Diffie Hellman key exchange (McCallum, 2014)

authentic, private or both. Typically this channel has a low-bandwidth. Without the use of a OOB channel, we can only be sure about the confidentiality and not about the authenticity. We will now give some examples of existing pairing protocols that can be relevant in our context.

### 4.3.1 Diffie-Hellman key exchange protocol

The Diffie Hellman (DH) key exchange is the basis for a lot of protocol's so we need a good understanding of this. Figure 4.6 explain how this works. The parameters in lowercase are public. These can be transmitted over an insecure channel without compromising the security, whereas the parameters in uppercase needs to stay secret. Alice and Bob just need to agree on two large prime numbers, $p$ and $g$. Both party's generate their private and public key and then exchange their public keys. In the example of figure 4.6, the private key $A$ is randomly chosen and the public key $a$ can derived from it, as $a = g^A \pmod p$. They can now calculate the same key with their private key and the other party's public key. Finally, they can use this key to communicate with a symmetric cypher.

Note that the exchange of the public keys is not authenticated, and therefore this scheme is vulnerable to MITM attacks. To solve this, we can make use of a Password-authenticated key agreement.

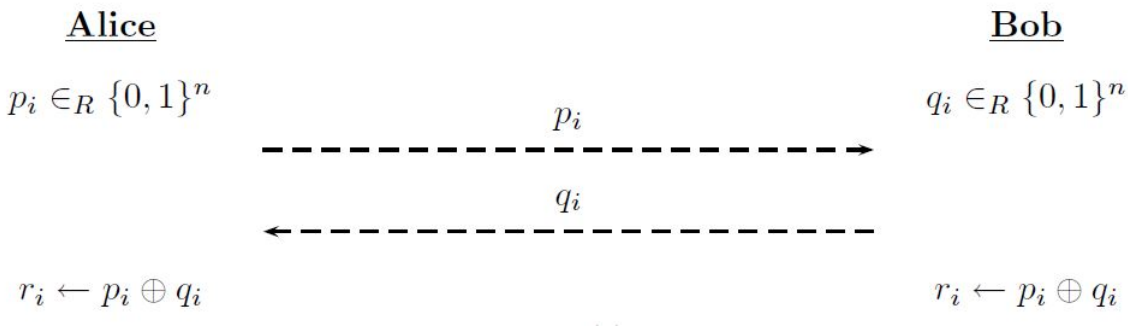


**Figure 4.7:** Mutual authentication over OOB, adapted from Singelee (2008)

### 4.3.2 Password authenticated key agreement (PAKE)

If Alice and Bob share a secret, then they can use this information to authenticate themselves. This secret is often a short password and can be exchanged over an OOB. There are two ways a password can be exchanged. If Alice chooses a password and sends it to Bob, they both agree on the same password, but there is only unidirectional authentication. If Alice and Bob both choose a password and send them to each other over the OOB, they can XOR their passwords and achieve mutual authentication. This is illustrated in figure 4.7. In this figure, the dashed lines represent a private and authentic OOB channel.

We will now examine two protocols that use a shared password in combination with a DH key exchange protocol.

### 4.3.3 Diffie-Hellman encrypted key exchange

Unlike DH key exchange, with this protocol the difference is that the public keys gets encrypted before transmitted. They use a hash of a shared password as the key for the encryption of the public key. Due to this encryption, a third party cannot alter the public key, this results in both confidentiality but more important, authenticity of the public key. There are three variants where either one or both of the public keys get encrypted. All three variants are equal in strength. The protocol is illustrated in figure 4.8.

| Step | Alice | Bob |
|---|---|---|
| 1 | Shared Secret: $S = H(password)$ | |
| 2 | Parameters: $p$, $g$ | |
| 3 | $A = \text{random}()$<br>$a = g^A \pmod p$ | $\text{random}() = B$<br>$g^B \pmod p = b$ |
| 4a | $E_S(a) \longrightarrow$<br>$\longleftarrow E_S(b)$ | |
| 4b | $a \longrightarrow$<br>$\longleftarrow E_S(b)$ | |
| 4c | $E_S(a) \longrightarrow$<br>$\longleftarrow b$ | |
| 5 | $K = g^{BA} \pmod p = b^A \pmod p$ | $a^B \pmod p = g^{AB} \pmod p = K$ |
| 6 | $\longleftarrow E_K(data) \longrightarrow$ | |

**Figure 4.8:** Diffie Hellmann encrypted key exchange (McCallum, 2014)

Simple Password Exponential Key Exchange

| Step | Alice | Bob |
|---|---|---|
| 1 | Parameter: $p$ | |
| 2 | $G = H(password)^2$ | $H(password)^2 = G$ |
| 3 | $A = \text{random}()$<br>$a = G^A \pmod p$ | $\text{random}() = B$<br>$G^B \pmod p = b$ |
| 4 | $a \longrightarrow$<br>$\longleftarrow b$ | |
| 5 | $K = G^{BA} \pmod p = b^A \pmod p$ | $a^B \pmod p = G^{AB} \pmod p = K$ |
| 6 | $\longleftarrow E_K(data) \longrightarrow$ | |

**Figure 4.9:** Simple Password Exponential Key Exchange (McCallum, 2014)

### 4.3.4 Simple password exponentional key exchange (SPEKE)

Instead of encrypting the public keys, with SPEKE we will use the shared password to generate the second parameter g. This parameter will be calculated by squaring the hash of the password. Parameter G will now become private and does not need to be exchanged over the insecure channel.

Both SPEKE and DH-EKE are resistant against MITM and offline dictionary attacks[2]. If there is a transmission error or an attacker alters a message, Alice and Bob won't share the same key K and the messages won't be readable to the other party. To solve this issue, these protocols can be followed by a key confirmation protocol, which SPEKE and DH-EKE do not provide.

[2] An online dictionary attack can be performed when an attacker can try to solve the key without interaction with one of both parties trying to setup a secure connection.

# Chapter 5

# The design of our pairing protocol

In these chapter we will make a proposal for a pairing protocol that we think is best for pairing the drone to a controller. Before we can specify a pairing protocol, we need to have an understanding of the technical abilities of the drone. These are explained in the first section of this chapter. Since our proposal depends on the use of QR codes, we will give a background of this in the third section.

## 5.1 Drone Specifications

Before we can specify a pairing protocol that we think is best for the AR.Drone 2.0, first we need to known all the technical specifications. These specs are listed in figure 5.1. This figure also illustrates the placement of the main components and sensors on the drone. In the scope of our thesis, we will focus on the on-board electronics.

We note that the drone is built on a ARM platform with a Cortex A8 processor supported by a DSP[1] and 1Gb of DDR2 RAM. To make this work, they use a Linux operation system with a 2.6.32 kernel. Furthermore, the drone is equipped with WIFI for communication with the controller and a USB port. This USB port can be used to attach a storage device, for example to store a video feed from one of the cameras, or a GPS module provided by

[1] Digital Signal Processor, supports the CPU with video processing tasks.

Parrot. The position data of this module can be used to improve the quality of the flight. Unnoticed in the specifications of Parrot (Parrot, 2016a), is the serial port on the main board. This gives us the ability to open a serial console using a wired connection.

Lastly, there are a lot of sensors attached on the main board. The data from the gyroscope, accelerometer, magnetometer, pressure sensor and ultrasound sensor are used for the precise movement of the drone. The two cameras provide the ability to record movies and take pictures, and helps the person operate the controller to steer the drone. The front facing camera has a resolution of 720p and can record movies at 30fps, where the down facing camera only has a resolution of 240p but has the ability to record at 60fps.

## 5.2 Our Design

### 5.2.1 The form of the symmetric key

The goal of our pairing protocol is to achieve a secure connection over the WIFI channel of the drone. Where the initial WIFI connection is not encrypted, the communication over this channel is non confidential, non-authentic, and the integrity of the data cannot be assured. If we can establish a symmetric key that satisfies to the last named primitives of security, we can use this key to setup an encrypted WIFI connection. For this encryption, we want to use the WPA2 protocol as it is advised by the WI-FI Alliance (Alliance, 2003). Another widely supported protocol, WEP, is outdated and badly broken (Cam-Winget et al., 2003).

As described in 802.11i (2004) ,the key size used for the WPA2 protocol is set to 256bit. To simplify the key sharing process for the user, the key can also be transmitted as 64 hexadecimal characters or a 13-63 character password, ASCII encoded. The limit of 63 characters is set to avoid confusion with a 64 character key. To convert the password into

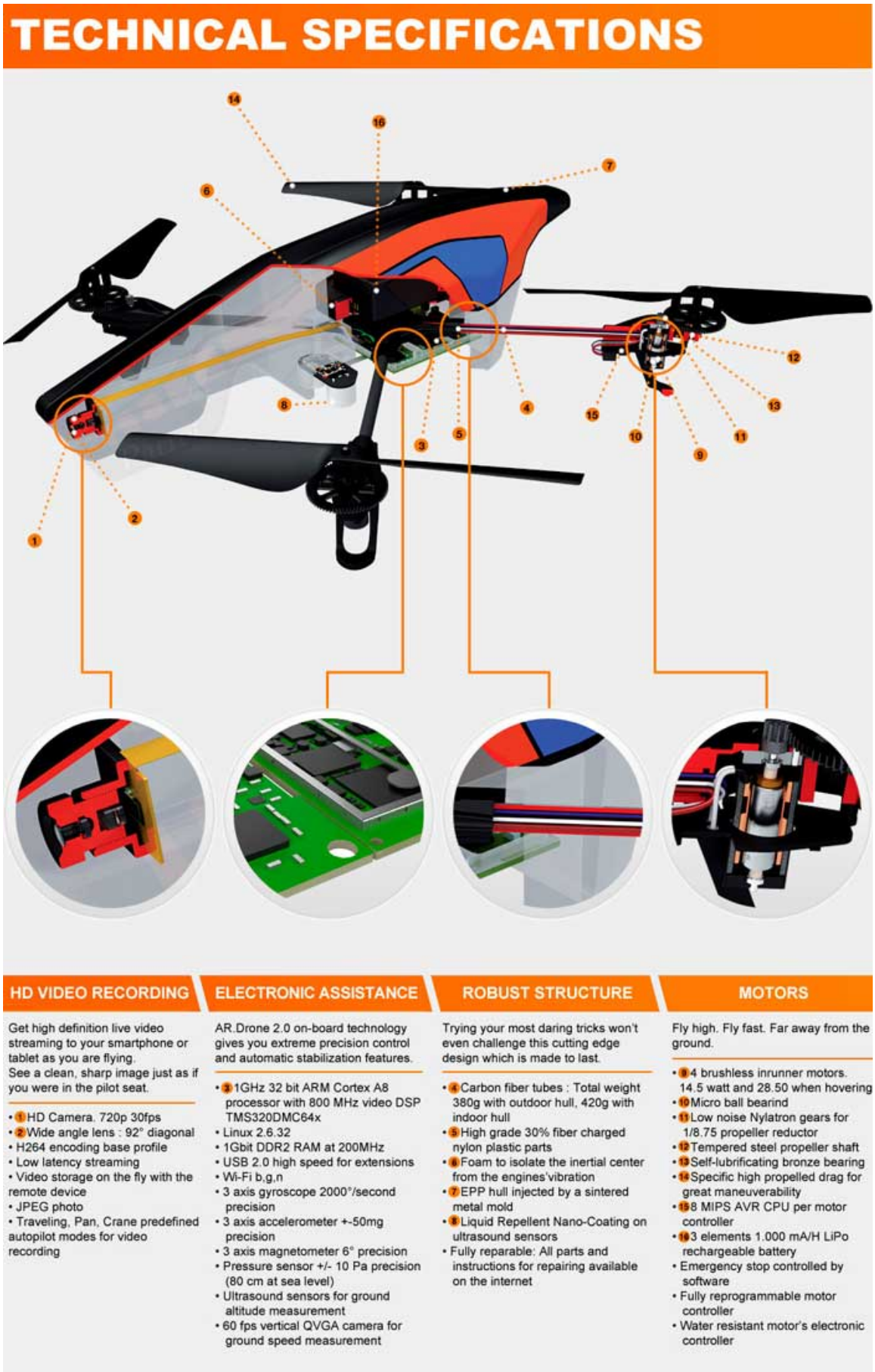


**Figure 5.1:** Technical specifications of the AR.Drone 2.0 (Parrot, 2016a)

a 256bit key, a key derivation function[2] is used, described as follows:

$$PSK = PBKDF2(Password, ssid, ssidLength, 4096, 256)$$

where $Password$ stands for the 13 to 63 bit password used by the user, $SSID$, the name of the access point, $SSIDLength$, the length of the SSID in bytes, $4096$, the number of times the password got hashed and $256$, the desired length of the key. This results in the creation of $PSK$, a 256bit key that can be used for WPA2 encryption. Note that the password can only contain readable ASCII codes, ranging from 32 to 126.

### 5.2.2 Choosing an OOB

As we cannot only use the WIFI to share a key between the drone and the controller, we will make use of an OOB channel. We will select an OOB channel listed in section 4.2 that we think is best for our application.

We will start by eliminating the ones that are not possible to implement on the drone. Due to the fact that the drone has no ability to communicate directly to people, for example a keyboard or a screen, we wont be able to implement any of the user-aided protocols like SAS or the MANA protocols. Furthermore, the drone has no microphone or speakers. For this reason it will be impossible to implement one of the sound based pairing protocols.

To choose one out of the remaining protocols, we made a comparison between them based on the papers of Kobsa et al. (2009) and Uzun et al. (2007). We evaluate them on their security level, user friendliness and available bandwidth. This comparison is visualized in table 5.1.

For the electric contact, we can establish a high bandwidth in a secure way. The necessity of a data cable for communication with the drone translates in a minor score for user

[2] A pseudo random function that stretches a variable length password into a fixed lenght key.

friendly. Using the accelerometers to track the movement of the drone results in a secure channel without the use of extra hardware. Because of the very low bandwidth achieved by this method, it will take a person too much time transferring a message to be stamped as user friendly. The visual channel is easy to use and provide us with a moderate bandwidth and security. As the drone has two cameras equipped, we won't need extra hardware. At last we have NFC, which is easy to use, can provide us with a high bandwidth, and assures a moderate level of security. However we would need to attach a NFC receiver to the drone.

If we compare our different options, we conclude that using the visual channel to setup a secure connection will be the best way.

### 5.2.3 Communication over the OOB

The protocols using the visual channel described in 4.2.3, will only provide authentication between 2 devices. For it to become a pairing protocol, we need to add a key exchange, for example the DH key exchange explained in section 4.3.1. If we integrate SIB with DH key exchange, this will results in figure 5.2. In the third step, Alice first sends a hash of her public key over the OOB. Bob can then check if it corresponds with the hash of Alice's public key received over the insecure network. This method requires the OOB to be authentic.

As this protocol provides a secure pairing method, two issues implementing it will rise. Firstly, will we will need to use the unencrypted WIFI of the drone. In the timeframe before the pairing is completed, an attacker will have full access to the drone due to the open Telnet and FTP ports, this will endanger the integrity of the software en the drone and the secrecy of the exchanged key. We can solve this problem by closing all the unnecessary ports on the drone but this will jeopardize any future communication with the drone if the protocol fails to pair. The second issue rises by the fact that we are obligated to use public cryptography. In contradiction to symmetric cryptography, this will require a significant amount of computing power and will therefore limit the available flight time.

We propose to exchange the full symmetric key suited for WPA2 over the visual OOB in

| Step | Alice | Bob |
|---|---|---|
| 1 | Parameters: $p$, $g$ | |
| 2 | $A = \text{random}()$<br>$a = g^A \pmod p$ | $\text{random}() = B$<br>$g^B \pmod p = b$ |
| 3 | $\text{h}(a) \dashrightarrow$<br>$a \longrightarrow$<br>$\longleftarrow b$ | |
| 4 | $K = g^{BA} \pmod p = b^A \pmod p$ | $a^B \pmod p = g^{AB} \pmod p = K$ |
| 5 | $\longleftarrow E_K(data) \longrightarrow$ | |

**Figure 5.2:** SIB protocol integrated in the Diffie Hellman key exchange

plain text. This is possible since the visual channel gives us a moderate bandwidth. To communicate over the visual channel, we can encode the key into a 2D barcode. We assume the controller has the ability to generate and show a 2D barcode and the drone can decode it using it's camera. Note that in this case our OOB needs to be authentic and confidential. Our implementation goes as follows:

- Instead of setting up an unencrypted network, the drone will only search for 2D barcodes with it's camera
- The controller will generate a random key suited for WPA2 and will encode this key into a 2D barcode to show to the drone.
- The drone will retrieve the key by decoding the barcode and set up a secure WPA2 network.
- The controller can now access this network as he has knowledge of the key.
- A secure connection has been established.

**Table 5.1:** Comparison of OOB channels usable for the drone.

| | Electric contact | Shaking | Visual | Nfc |
|---|---|---|---|---|
| Bandwith | ++ | – | +- | + |
| User Friendly | +- | - | ++ | + |
| Security | ++ | + | +- | +- |
| Extra hardware | Yes | No | No | Yes |

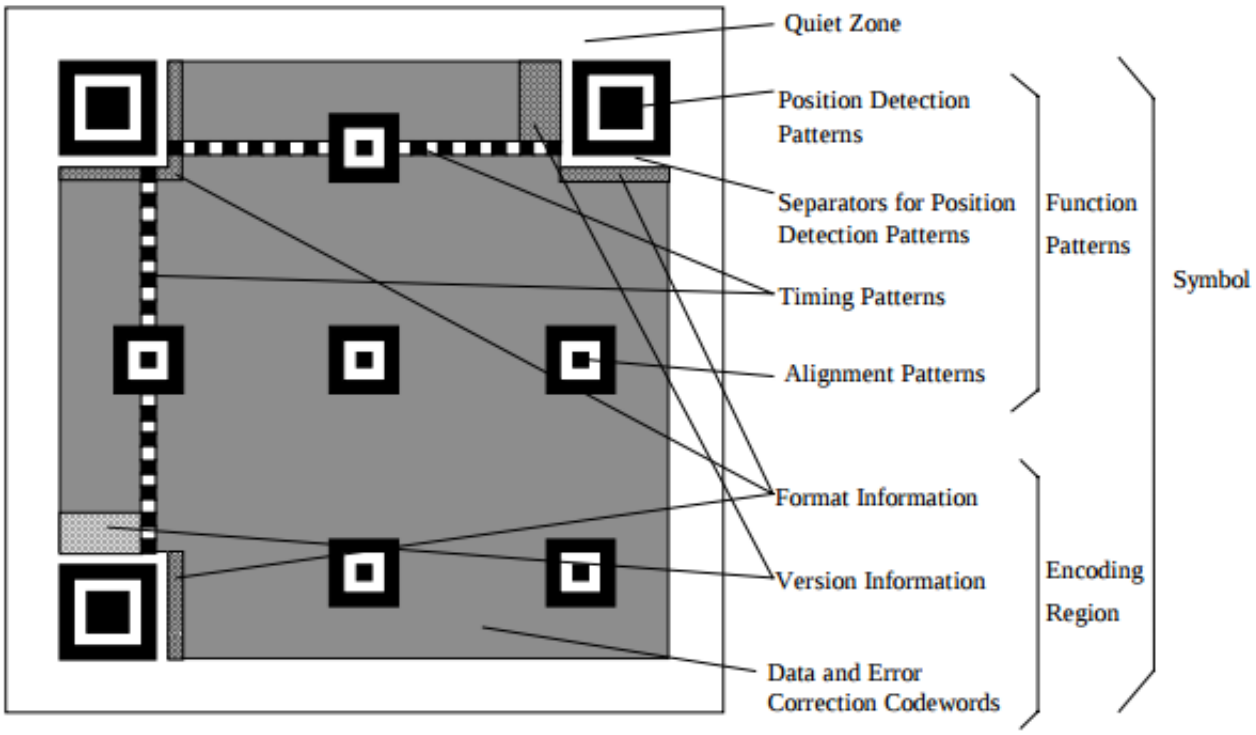


**Figure 5.3:** Structure of a QR code symbol (International Organization for Standardization, 2006)

## 5.3 QR-code, a 2D-barcode

To encode data over the visual channel, we will make use of a 2D barcode. As there are different standards available, we will choose to implement the Quick Response (QR) code as this provides us with different advantages. This protocol described by International Organization for Standardization (2006) can provide us with enough data capacity to encode a 256bit key. Furthermore, the orientation towards the decoding device is free to choose and it has integrated error correction. Support and open source libraries are widely available. Figure 5.3 shows the structure of a QR code.

There are 40 different versions of QR codes, starting from 1 up to 40. Each increasing version has the ability to store more data as the number of modules increases. A module represents one black or white square. The number of modules is defined by

$$Total\ modules = (17 + (version\ number\ x\ 4))^2$$

, so version 1 has 21 x 21 modules and version 40 has 177 x 177 modules.

The format information contains the level of error correction. There are 4 levels included in the ISO standard: L, M, Q, and H. These provide respectively 7, 15, 25 and 30 percent data restoration rate. It is obvious that a higher level of error correction results in a smaller capacity for data storage.

As described in section 6.3, we will only make use of QR code version 2 in our context. Figure 5.4 illustrates the detailed structure of this version. We notice that this version has only one alignment pattern. The actual data followed by the error correction (EC) words are stored in the gray squares. The sequence starts from the bottom right corner going up and follows a snake pattern till al the modules are used. Every gray square contains 1 codeword and consists of 8 bytes.

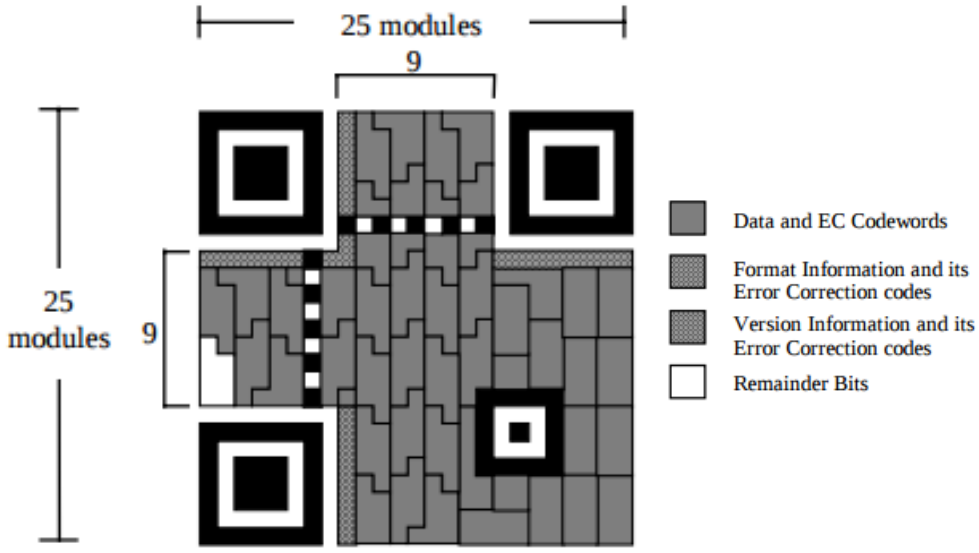


**Figure 5.4:** QR code version 2 with legend (International Organization for Standardization, 2006)

# Chapter 6

# Implementation

In this chapter we will discuss the process implementing our pairing protocol on the drone. We split this implementation into three sections. Firstly, we will discus the conversion of the open access point to an encrypted one using WPA2. In the second section we will discuss the application needed on the controller to pair with the drone. At last we explain how we implement QR decoding software on the drone. For every section we will state our objectives, the methods we used, and problems that raised.

## 6.1 Secure WIFI access point

Out of the box, the drone will setup an open WIFI network when powering up. The drone will act as the host of the network and a controller can connect as a client. The drone will allocate ip address $'192.168.1.1'$ to itself. The mac-address of the first device connecting to this network will be stored by the drone. The drone will check this address before accepting any commands from a controller.

In our proposed model, the drone will remain the host of the network by setting up an access point. However, this access point will not be open to connect, but use WPA2 encryption instead. The encryption of the network traffic is hardware-supported by the network interface. As the drone did not have the option to enable WPA2, we had to add

```
1 interface=wlan0
2 driver=nl80211
3 ssid=ARDroneWPA2securedhotspot
4 hw_mode=g
5 channel=6
6 macaddr_acl=0
7 auth_algs=1
8 ignore_broadcast_ssid=0
9 wpa=3
10 wpa_passphrase=Parrot
11 wpa_key_mgmt=WPA-PSK
12 wpa_pairwise=TKIP
13 rsn_pairwise=CCMP
```

**Figure 6.1:** Our hostapd configuration file

it ourselves. The designated way to handle this is, is to run host access point daemon (hostapd). This software acts as a daemon[1] and will tell the the network interface to act as an access point with the provided configurations. As Parrot did not provide this software for the drone, we had to cross-compile it. This cross-compilation was a long, struggling process without any success. We will list a few of the methods we tried to make it work.

- We first tried to cross-compile hostapd by following the tutorial of BeyondLogic (2014). In this tutorial, they first cross-compile Netlink Protocol Library Suite (libnl) and open Secure Sockets Layer (openSSL) as this will provide the needed libraries we will need to cross-compile the hostapd software. They use the 'arm-linux-gnueabi' cross-compiler that we installed from the official Ubuntu repository on our Ubuntu 14.04 release. After the compilation we became a hostapd executable that we transferred using FTP into the drone together with our 'hostadp.conf' file containing the configurations to setup a WPA2 secured hotspot. This file is illustrated in figure 6.1.

  Trying to execute hostapd on the drone raised a number of errors. As we did not specify to our compiler to compile the needed libraries statically, they were dynamically linked and not included in the executable. We solved this problem by transferring these libraries to the drone. These included the 'libnl' and 'libcrypto' libraries, already created by cross-compiling libnl and openSSL. At last, the drone told us that

[1] A daemon is a computer program that runs in the background.

```
Configuration file: hostapd.conf
nl80211: Could not configure driver mode
nl80211 driver initialization failed.
```

**Figure 6.2:** Error while trying to execut hostapd

we needed a newer version of 'GLIBC'. This is a core library present at any linux operation system, containing basic systemcalls. We downloaded, cross-compiled and replaced it with the existing GLIBC library on the drone. This resulted in the crash of our drone. After rebooting, all the drone's leds were blinking red and the drone did not create a WIFI hotspot. We tried to save our drone by accessing it trough the serial port using a uart-to-usb converter, without success. Since we recieved a kernel panic error in the booting process, we could not restore the original libraries, and there was no option for a factory reset. Our drone was bricked. We contacted Parrot for support but they could only advise us to replace the motherboard of the drone. After an easy replacement, the drone was operating fine again.

What went wrong and how could we solve this? We think that by replacing the GLIBC library, vital parts of the Linux operating system stopped working. The version of a GLIBC library is dependent on the kernel version it's running on. Replacing or updating this library is strongly discouraged. The solution for this issue is to build against the same kernel version as our drone, version $2.6.32$. This can be done by using u cross-compiler from the official repository on a platform with the same kernel version, e.g. Ubuntu 10.4, or by installing a custom compiler with a compatible toolchain. We went for the first solution, installed Ubuntu 10.4 and repeated the compiling process. We now got rid of all the library mismatches when trying to execute hostapd. However, the next error occurred, as shown in figure 6.2. This indicated a driver mismatch. The driver interface, nl80211, could not successfully communicate with the driver of the wireless interface of the drone. We tried to solve this problem by building hostapd with madwifi support and drivers for atheros interface (Leffler, 2010), without success. We decided to try it another way.

- In our second attempt, we tried to run Ubuntu on top of the drone's operating system following the tutorial of Mindego (2013). He proposes to use debootstrap, a tool to install an operation system into a directory of your current system, to install Ubuntu 10.4 ported for ARM. When installed, it can be accessed by using qemu, a tool to execute arm executables. Copying this to an usb stick and plugging it into the drone will provide us with al the capabilities of Ubuntu. We could now install software packages from the official repositories. We successfully installed some basic software as 'nano', but when we tried to execute hostapd after installing it, we obtained the same error as in figure 6.2. Installing and activating other drivers for our drones Atheros interface did not provide a solution for our problem.

- At last, we tried to implement a WEP secured access point. This can be done by using the 'iwconfig' software, already available on the drone. No other software, like hostapd, is needed. Using the manpages of iwconfig (Tourrilhes, 2004), we executed the following commands.

  - $iwconfig\ wlan0\ mode\ managed$ - to let our interface act as an access point.
  - $iwconfig\ wlan0\ enc\ restricted$ - to turn on WEP encryption
  - $iwconfig\ eth0\ key\ s : password$ - to set the password

  This resulted in the creation af a WEP secured access point created by the drone with the same SSID. However, we were not able to connect to this network using our chosen password. We looked for any format errors in the password but couldn't fix this issue.

After spending a significant amount of time trying the previous options without success, we needed to look for another solution to setup a secure connection. One of these solutions was already proposed and implemented by Pleban et al. (2014). Instead of adapting the access point on the drone to a secure one, he cross-compiled and installed 'WPA supplicant' on the drone. This will give our drone the ability to connect to a WPA2 secured hotspot. The controller now has the responsibility to setup a secured network with the key

it generated itself. After reading the key from the controller in the form of a qr code, the drone now needs to seek a connection with the controllers network. The controller will act as the server and the drone as client. This, however, will have a serious disadvantage towards the original proposed model.

As open source software for most used operating systems exists (e.g. Linux, windows, android), we used the official Parrot application using our android device. This application will try to access the drone on the IP address 192.168.1.1, but it won't find the drone as the DHCP server of our android device only hands out ip addresses in the range of 192.168.42.1-256. Android does not offer to adapt these settings and we cannot tell the application to look for the drone on another network address. The only option to make it work with the official Parrot application is to gain root access to the phone and adjust the DHCP setting manually. Other open source applications to fly the drone have the option to set the drone's ip address manually and provide us with the ability to control our drone with the secure connection.

## 6.2 Application on the controller

The application we are going to build for our controller will need to handle to pairing process with the drone. We will build an application for the controller that we use, a smartphone running android. With regard to our pairing model described in section 6.1, this application needs to generate a random key and encode it into a QR code. After we showed this code to the drone, the application will need to connect our phone to the drone's secure network.

Since we had no experience in building applications for android, we inspected a few ways to handle this. The obvious choice would be to use 'Android studio'. This is a software packet provided by android including an IDE and all the needed libraries to develop applications for its platform. As it takes time to master the environment and the application interfaces, we chose for a different, simpler solution. We used an advanced drag-and-drop

system that is developed by Massachusetts Institute of Technology (MIT). According to their website:

> ”MIT App Inventor is an innovative beginner’s introduction to programming and app creation that transforms the complex language of text-based coding into visual, drag-and-drop building blocks. The simple graphical interface grants even an inexperienced novice the ability to create a basic, fully functional app within an hour or less.” MIT (2012)

We will use the beta of the second version of app inventor (AI) as this provides us the ability to add third party extensions. AI has two modes to work with, the designer mode and the block mode. In the designer mode, we can drag-and-drop al the visual components of our application, like buttons and text areas. In the block mode, we can use logic blocks to set the behavior of these components. AI provides an online emulator to test our application but it also offers the possibility to download the app directly to our phone.

Our first application had 2 screens. The first showed a random generated key and a canvas with the corresponding QR code. If you were not satisfied with the key, there was an option to generate another random key. However, it was not possible to manually enter a personal key. People tend to be bad at choosing random keys, and the process is automated so the user would never have to enter the key manually. On the bottom of our first screen is a button called ’connect now’ that brings us to our second screen. Note that before switching screens, the QR code has to be shown at the drone. Our second screen will auto enable the phone’s WIFI and list all of the available access points. The user will now have to select the drone’s SSID out of the list and the application will make a connection with that network using the previous generated key. Accessing the WIFI configuration is not a built-in part of AI, but comes with an extension provided by PuraVidaApps (2015).

As we were obligated to change our pairing protocol and make our controller the server of the network (described in 5.2.3), we also had to alter our application. The second screen of our application will now be replaced by the option to redirect to the hotspot setting menu.

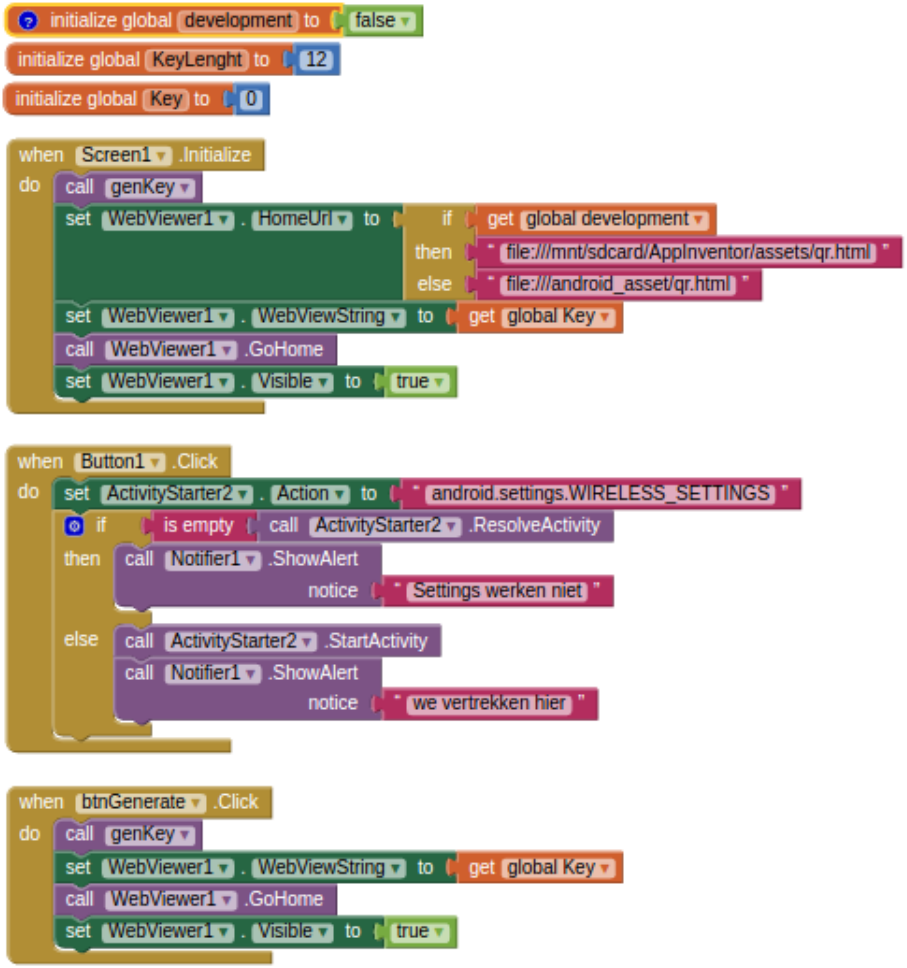


**Figure 6.3:** Variables and behavior of buttons in our application

As we could not configure the setting for the hotspot from within the application, the generated key got stored on the clipboard for easy entering in the options menu.

Our application is built with the block illustrated in figure 6.3 and 6.4. The first figure shows the declaration of the variables, event listeners on the buttons, and the initialisation sequence when starting the application. The second figure illustrates the key generation function and a decimal-to-ASCII-conversion function. For the generation of the QR code, we use a javascript provided by Etienne (2012). This script will automatically choose the needed version for the QR code, using a 30% error correction.

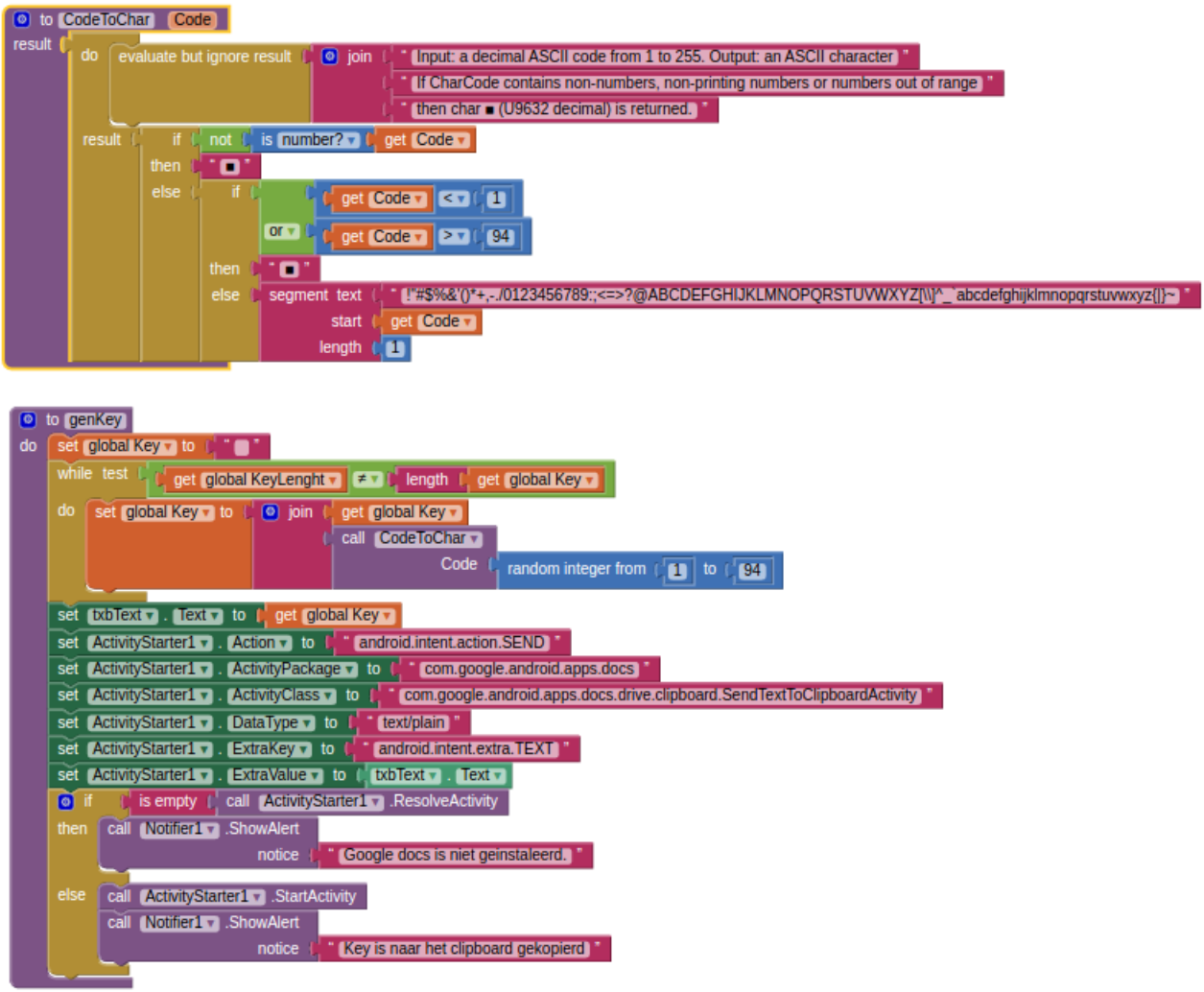


**Figure 6.4:** implemented routines in our application

```
1 #!/bin/bash
2
3 m2_gpio_init(181, 1); // Red LED on
4
5 essid=ar2
6 password=`/script/readqr`
7
8 succes=`/script/connect $ar2 -p $password`
9
10 if[succes] then m2_gpio_init(180, 1); // Green LED on
```

**Figure 6.5:** Our script for WIFI initialization

## 6.3 Application on the drone

The application we are going to develop on our drone will need the ability to scan and decode the QR code, and pass the value to the WPA supplicant. This value will represent the key needed to connect with the secure access point from our controller. We need to make sure that the drone does not endanger itself opening an insecure access point before pairing with the controller. It will also be helpful that the drone indicates a success/failure trying to read the 2D barcode.

Every time when we power up the drone, the script 'rcS' located at '/etc/inet.d/rcS' will be executed. This script will initialize all the hardware and software needed to fly the drone. This includes setting up an insecure access point, executed by '/bin/wifi_setup.sh'. As we do not want our drone to endanger itself, we will remove this line from the initialization script and replace it by the script we developed ourselves. This script receives the key from the QR reader executable, and present it together with the SSID of the controller's hotspot to the connection script provided by Araos (2013). If the connection is a success, the green LED's will turn green. You can find our script in figure 6.5. Note that since we were only planning to encode our WPA2 key in the QR code, and the access point is configured by the controller, the SSID of the access point is hard coded in the script.

The last part we had to implement was the QR reader itself. At first, we used a software suite called 'Zbar'. Zbar is an open source project that provides several applications. It provides two executables. 'Zbarcam' for scanning and decoding a barcode from a em-

bedded camera device and 'Zbarimg', for decoding a barcode provided as a picture. It also provides us with an interface we can use in combination with python and C++. We first tested it on our computer and the results were promising, but when we transferred and tried to execute 'Zbarcam' on the drone, the software failed to communicate with the camera of the drone. This was because the camera was already occupied by the drone's firmware. This firmware takes the output of the camera and produces a TCP stream on port 5555. If we want to process the images captured by the camera we need, to get them from $tcp://192.168.1.1:5555$ instead of tapping into the source. Unfortunately, this was not supported by 'Zbarcam'.

In our second approach, we tried to access the camera feed over the TCP stream using opencv. We cross-compiled the opencv libraries following the tutorial of Hassan (2015). We first wrote some code that took a picture from the drone's camera, and converted it to a gray image and stored this on the drone. After we altered our code by skipping the first images received from the camera, our code executed fine and stored a gray image on the drone. You can find a listing of this code on the next page.

Since we used precompiled binaries in our first approach, we now had to cross-compile a library that we could use for QR decoding within opencv. We experienced with 'Zbar', 'Zxing' and 'libdecodeqr', but we failed to compile a working library out one of these three open source projects. At this moment, we were not able to solve these issues and therefore we cannot demonstrate a working prototype of our pairing protocol.

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <string>
#include <stdio.h>
#include <unistd.h>
#include <iostream>
#include <ctime>

using namespace cv;
using namespace std;

int main(int argc, char** argv){
   char fileName[30];
   char url[]="tcp://192.168.1.1:5555";

   cv::Mat foto;

   std::cout<<"Trying to access the camera ...\n";
   VideoCapture vc(url);


   // Clear pipeline
   for(int i=0;i<100;i++) vc.read(foto);


   if(vc.read(input)){

        cvtColor( input, input, CV_BGR2GRAY );
        sprintf(fileName,"gray.jpg");
        cv::imwrite(fileName, foto);
        std::cout<<"Capture saved"<<std::endl;
        }
        else{
        std::cout<<"something went wrong"<<std::endl;
        return 1;
        }

   return 0;
}
```

# Chapter 7

# Conclusions and Future Research

## 7.1 Conclusions

We learned from the available literature that there was a security issue with our drone. In this thesis, we present a solution to overcome this problem. Since cryptography is the key to solve this issue, we gave a brief introduction and zoomed in on different pairing protocols. We made a comparison between the different OOB channels that we have available on the drone and we proposed a protocol to setup a secure communication. We evaluated the visual channel as the best fit for our application, as this would minimize the loss of usability for the user. We suggested to encode a WPA2 key into a QR barcode and transfer it by presenting the controller's display to the camera of the drone.

In chapter six, we described our process trying to implement our proposed pairing protocol. We failed to setup a secure access point on the drone due to miscommunication with the drivers. However, we were able to prevent the drone from creating an insecure network and let the drone make a connection with a secure access point. This ensures the integrity of the software on the drone.
We successfully developed an android application to help the controller pair to the drone. This application is capable of creating a WPA2 conform key and decode it into a QR barcode.

In the process of creating an application that is able to scan and decode a barcode we encountered library and dependency issues that we were not able to resolve. The application we created was able to scan the barcode, but the decoding part was still missing. In the absence of this application, we are not able to present a working proof of work.

## 7.2 Future work

As drones are becoming more advanced and affordable, the need for secure communication expands. In our thesis, we only focused on one specific drone. It is likely that there are many more drones with the same security issue. A comparative analysis would reveal the magnitude of this problem.

We proposed a protocol to pair our drone with a controller. We failed to implement it, but we are convinced that it should be possible. A demonstration of a user-friendly implementation would be a major added value to our thesis. If this implementation has been made, it is also valuable to analyse what implications it has on the drone. It is probable that the flight time gets decreased and the latency gets increased due to the encryption process.

As the data decoded in the QR code will always represent an equal amount of bytes, there is a possibility to speed up the decoding process by presetting the version and format information of the QR code. To make our protocol safer without endangering the usability, we propose to add the SSID of the network to the content of the QR code. In this setup, the SSID can be random every flight. This will improve the strength of the key derivation function used by WPA.

# Bibliography

802.11i, I. S. (2004). IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications Amendment 6: Medium Access Control (MAC) Security Enhancements.

Alliance, W.-F. (2003). Wi-Fi Protected Access: Strong, standards-based, interoperable security for today's Wi-Fi networks. *White paper, University of Cape Town*, page 492–495.

Amariucai, G. T., Bergman, C., and Guan, Y. (2011). An Automatic, Time-Based, Secure Pairing Protocol for Passive RFID. In Juels, A. and Paar, C., editors, *RFIDSec*, volume 7055 of *Lecture Notes in Computer Science*, page 108–126. Springer.

Araos, D. (2013). AR.Drone WPA/WPA2 support. `https://github.com/daraosn/ardrone-wpa2`.

BeyondLogic (2014). Cross Compiling iw wpa supplicant hostapd rfkill for ARM. `http://wiki.beyondlogic.org/index.php?title=Cross_Compiling_iw_wpa_supplicant_hostapd_rfkill_for_ARM`.

Cam-Winget, N., Housley, R., Wagner, D., and Walker, J. (2003). Security flaws in 802.11 data link protocols. *Communications of the ACM*, 46(5):35–39.

Daemen, J. and Rijmen, V. (1999). AES Proposal: Rijndael.

Dutta, R., Barua, R., and Sarkar, P. (2004). Pairing-Based Cryptographic Protocols : A Survey. *IACR Cryptology ePrint Archive*, 2004:64.

Etienne, J. (2012). jquery.qrcode.js. https://github.com/jeromeetienne/jquery-qrcode.

Federal (2008). *BSI-Standard 100-2 IT Baseline Protection Methodology*. Bonn, version 2.0 edition.

Gehrmann, C., Mitchell, C. J., and Nyberg, K. (2004). Manual authentication for wireless devices. *Cryptobytes*, 7(1):29–37.

Gehrmann, C. and Nyberg, K. (2001). Enhancements to Bluetooth baseband security. In *Proceedings of Nordsec 2001*.

Gehrmann, C. and Nyberg, K. (2004). Security in personal area networks. *IEE TELECOMMUNICATIONS SERIES*, 51:191–229.

Goodrich, M. T., Sirivianos, M., Solis, J., Tsudik, G., and Uzun, E. (2005). Loud and Clear: Human-Verifiable Authentication Based on Audio. *IACR Cryptology ePrint Archive*, 2005:428.

Hassan, N. (2015). How to Cross-Compile OpenCV with FFmpeg (x264 and xVid) for AR Drone (ARM Processor). http://hassannadeem.com/blog/2015/04/29/cross-compile-opencv-with-ffmpeg-ar-drone-arm/.

Holmquist, L., Friedemann, M., Schiele, B., Alahuhta, P., Beigl, M., and Gellersen, H. (2001). Smart-Its Friends: A Technique for Users to Easily Establish Connections between Smart Artefacts. *Lecture Notes in Computer Science*, 2201:116.

International Organization for Standardization (2006). Information Technology — Automatic Identification and Data Capture Techniques — QR Code 2005 Bar Code Symbology Specification. ISO/IEC 18004:2006.

Kamkar, S. (2013). SkyJack. https://github.com/samyk/skyjack.

Kobsa, A., Sonawalla, R., Tsudik, G., Uzun, E., and Wang, Y. (2009). Serial hook-ups: a comparative usability study of secure device pairing methods. In *Proceedings of the 5th Symposium on Usable Privacy and Security*, page 10. ACM.

Leffler, S. (2010). The MadWiFi project. Available at http://madwifi-project.org.

Malinen, J. (2013). Linux WPA/WPA2/IEEE 802.1X Supplicant. https://w1.fi/wpa_supplicant/.

Mayrhofer, R. and Gellersen, H. (2007). Shake Well Before Use: Authentication Based on Accelerometer Data. In LaMarca, A., Langheinrich, M., and Truong, K. N., editors, *Pervasive*, volume 4480 of *Lecture Notes in Computer Science*, page 144–161. Springer.

Mayrhofer, R. and Gellersen, H. (2009). Shake Well Before Use: Intuitive and Secure Pairing of Mobile Devices. *IEEE Trans. Mob. Comput.*, 8(6):792–806.

McCallum, N. (2014). Authenticated Key Exchange with SPEKE or DH-EKE. http://www.themccallums.org/nathaniel/2014/10/27/authenticated-key-exchange-with-speke-or-dh-eke/.

McCune, J. M., Perrig, A., and Reiter, M. K. (2009). Seeing-Is-Believing: Using Camera Phones for Human-Verifiable Authentication. *International Journal of Security and Networks Special Issue on Secure Spontaneous Interaction*, 4(1–2):43–56.

Mindego (2013). Ubuntu on AR.Drone 2.0. https://www.drone-forum.com/forum/viewtopic.php?t=7094.

MIT (2012). MIT application inventor. http://appinventor.mit.edu/explore/about-us.html.

Nechvatal, J., Barker, E., Bassham, L., Burr, W., and Dworkin, M. (2000). Report on the development of the Advanced Encryption Standard (AES). Technical report, DTIC Document.

Parrot (2016a). Parrot AR.Drone 2.0. http://www.parrot.com/nl/producten/ardrone-2/.

Parrot (2016b). Parrot for developers. https://developer.parrot.com/.

Pleban, J.-S., Band, R., and Creutzburg, R. (2014). Hacking and securing the AR.Drone 2.0 quadcopter: Investigations for improving the security of a toy.

PuraVidaApps (2015). Pura Vida Apps - WiFi Manager Extension. http://puravidaapps.com/wifi.php.

Samland, F., Fruth, J., Hildebrandt, M., Hoppe, T., and Dittmann, J. (2012). AR.Drone: security threat analysis and exemplary attack to track persons.

Saxena, N., Ekberg, J.-E., Kostiainen, K., and Asokan, N. (2006). Secure Device Pairing based on a Visual Channel. *IACR Cryptology ePrint Archive*, 2006:50.

Singelee, D. (2008). *Study and design of a security architecture for wireless personal area networks*. PhD thesis, KUL.

Soriente, C., Tsudik, G., and Uzun, E. (2007). HAPADEP: Human Asisted Pure Audio Device Pairing. *IACR Cryptology ePrint Archive*, 2007:93.

Stajano, F. and Anderson, R. (2000). The resurrecting duckling: Security issues for ad hoc wireless networks. *7th International Workshop*, 1796 of Lecture Notes in Computer Science:172–194 Springer–Verlag.

Stallings, W. (2014). *Cryptography and network security - principles and practice (6. ed.).* Prentice Hall.

Tourrilhes, J. (2004). iwconfig. http://linuxcommand.org/man_pages/iwconfig8.html.

Uzun, E., Karvonen, K., and Asokan, N. (2007). Usability Analysis of Secure Pairing Methods. In Dietrich, S. and Dhamija, R., editors, *Financial Cryptography*, volume 4886 of *Lecture Notes in Computer Science*, page 307–324. Springer.

Vaudenay, S. (2005). Secure Communications over Insecure Channels Based on Short Authenticated Strings. In Shoup, V., editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, page 309–326. Springer.

FACULTY OF ENGINEERING TECHNOLOGY
TECHNOLOGY CAMPUS DE NAYER
Jan De Nayerlaan 5
2860 SINT-KATELIJNE-WAVER, Belgium
tel. + 32 15 31 69 44
fet.denayer@kuleuven.be
www.fet.kuleuven.be