Master Thesis

Radboud University Nijmegen

Attribute-based credentials on smartwatches

Author:
Martijn Terpstra
M.A.Terpstra@student.ru.nl
s0814962

Supervisor:
Prof. Lejla Batina
Lejla@cs.ru.nl

Daily supervisor:
dr. Veelasha Moonsamy
v.moonsamy@cs.ru.nl

Second reader:
dr. ir. Joeri de Ruiter
joeri@cs.ru.nl

October 25, 2017

Abstract

Smartwatches are wearable technology that is becoming increasingly powerful and common in use. Improvement in smartwatch technology means faster processing speed and new ways to interact with other devices. These new capabilities come with new risks but also with new possibilities. On the one hand, wearable technology is a large threat to privacy because it can collect personal data very accurately through its sensors. On the other hand, wearable technologies may allow for new use cases, which might have been complicated with other hardware. The current generation of smartwatches is similar in processing speed to early smartphone. Because of this, it is worthwhile to research if smartwatch can do some things as good or better than other devices such smartphones.

In this thesis I will explore the use of attribute-based credentials (ABCs) on smartwatches. Attribute-based credentials are a method of authentication, which provides advantages over the commonly used identity-based credentials. In contrast to identity-based credentials, attribute-based credentials allow for selective disclosure of attributes and avoid the need for an identity, allowing for easier safeguarding of privacy. In addition to exploring the usefulness and practicality of authenticating using attribute-based credentials on a smartwatch, I will implement a proof of concept application based on the existing IRMA project to show that a smartwatch implementation is technologically viable.

Contents

Contents ii

1 Introduction 1
- 1.1 Smartwatches . . . 2
- 1.2 Authentication . . . 4
- 1.3 IRMA Project . . . 6
- 1.4 Related work . . . 7

2 Methodology 8
- 2.1 Relevant Parties . . . 8
- 2.2 Technical requirements of use cases . . . 9
- 2.3 Role of smartwatch during authentication . . . 10
- 2.4 Development of Proof of Concept . . . 11

3 Proof of Concept 15
- 3.1 Initial Android wear apps . . . 15
- 3.2 IRMAWear . . . 16
- 3.3 IRMAWear2 . . . 19
- 3.4 Limitations . . . 25

4 Results & Future work 26
- 4.1 Results . . . 26
- 4.2 Future Work . . . 26

Bibliography 28

Glossary 31

Software sources 32

Specifications Sony Smartwatch 3 33

Specification Samsung J100H 34

Chapter 1

Introduction

When a user wants to use a service, which is provided by service provider, the service provider may require the user to authenticate themselves. By authenticating, a user reveals some information about themselves. For example, a customer wants to buy alcohol at a supermarket. To ensure that the customer is of drinking age, the supermarket requires that the customer authenticates himself. The customer then shows his driver's license, revealing his date of birth and proving that he is of drinking age.

Authentication can be done via use of ID cards and smartphones and now smartwatches. Smartwatches are not similar to smartphones. What immediately separates smartwatches from smartphones is that smartwatches are small and wearable. A smartphone, if needed for authentication has to be taken out a bag or pocket, unlocked etc. A smartwatch can be available instantly and can be more easily carried around. If the authentication itself is done in a matter of second, taking out a smartphone, unlocking it and storing it after authentication could easily take more time than the authentication itself. This simplification of the authentication process could greatly improve the usability and in turn the adoption of any authentication application.

In this thesis I will explore the use of attribute-based credentials (ABCs) on smartwatches. To do so, I will research and document attribute-based credentials and smartwatches and their unique advantages and shortcomings. After that, I will write a smartwatch-specific application to study the use of attribute-based credentials. Finally, I will document the current shortcomings of smartwatches when it comes to their use of attribute-based credential authentication.

I will first discuss the necessary background knowledge, both the terminology and the relevant preexisting work in this chapter. I will then dedicate a chapter to explain my methodology. This consists of formalizing the technical requirements for an attribute-based credentials implementation and the exploration of useful use cases. After that I will dedicate chapter 3 to explain a proof of concept I made of a working implementation of attribute-based credentials on a smartwatch. Finally, I will summarize my findings on the matter and discuss the future work that can be done.

1.1 Smartwatches

Smartwatches are wearable computers worn around a person's wrist. In addition to timekeeping, like traditional watch, smartwatches comes with a mobile operating system and are capable of running applications. Smartwatches usually have embedded sensors that are able to measure things like movement and temperature. Smartwatches can interact directly with a user via voice communication and a touchscreen, and with other devices using WiFi and/or Bluetooth. One feature that is noticeably absent when compared to a smartphone is a video camera. Some, but not all smartwatches also support NFC (Near-field communication). Additionally, the smartwatch can act as interface for the user to the smartphone. Besides the difference in sensors, the most noticeable difference between the various smartwatches is the operating system. Popular smartwatch operating systems are *Tizen*[1] developed by Samsung, *WatchOS*[2] developed by Apple and *Android Wear*[3] developed by Google.

For this thesis I looked at smartwatches running Android Wear specifically. The regular Android operating system runs on smartphones and tablets. Android Wear is a version of the Android operating system specifically designed for smartwatches. The most important reason for choosing Android Wear is that there already exists a working Android implementation of IRMA. This is useful because large parts of the Android application code base can be replicated. Because of this similarity, only the parts specific to Android Wear have to be rewritten. Furthermore, Android wear is also practical because the Android development tools can also be used for Android Wear development. Particularly useful is the Android emulator, which emulates an Android device on non-Android machines.

Compared to other devices such as smartphones, smartwatches are unique in a few ways. Due to its nature it tends to always be active and listening. The wearing of the watch means there is no delay between using and not using the device. Whereas a smartphone would occasionally be locked or put away. One specific area where the smartwatch does better than a smartphone is (physical) gesture recognition[26]. While most smartphone also have motion sensors, the data from a smartwatch is more accurate as it is worn by the user. This introduces new use cases like the gesture-based authentication [15][16]. Another advantage of a smartwatch over a smartphone lies in the flexibility of its hardware. Unlike a smartphone, a smartwatch can be worn and can be used immediately. A smartwatch cannot always be as easily used. A smartphone usually requires a free hand to hold it when using the device. In addition to these features, smartwatches come with sensors like GPS, gyroscope, hearth rate monitor and diving depth sensor.

A smartwatch is frequently used in combination with a smartphone. The smartwatch I used for my experiment would skip its initial setup until it had been paired with a smartphone. When connected with my It acted as an additional screen. The smartwatch would show the smartphone, the smartwatch would act as an extension of my smartphone.

[1] `https://www.tizen.org`
[2] `https://www.apple.com/watchos/`
[3] `https://www.android.com/wear/`

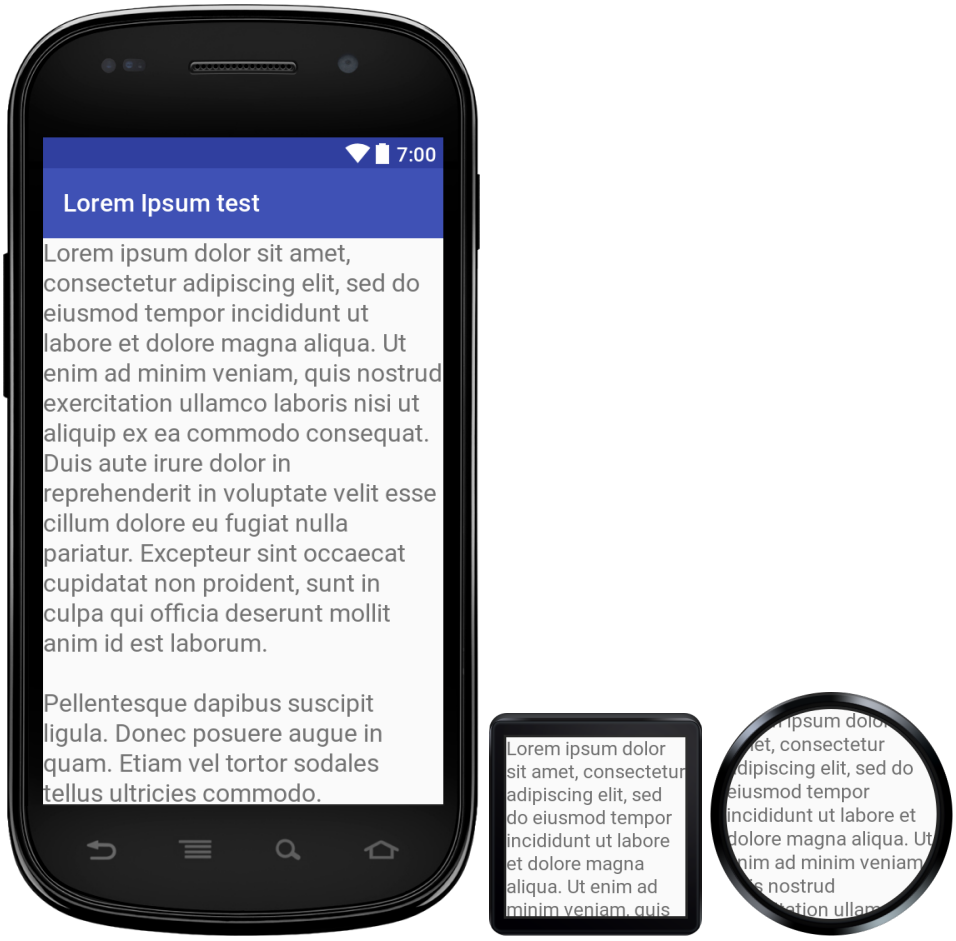

Figure 1.1: Comparison of screen size between smartphone and smartwatch using placeholder text. In practice applications do not devote their entire screen to displaying text, leaving room for white space, images and interactive elements such as onscreen buttons

For example a user can view notifications coming from his smartphone on his smartwatch. It can also act as an addition input when used alongside a smartphone. For example, the smartwatch can set alarms and skip music tracks when playing music on the phone.

From a usability perspective there are multiple problems. The main problem with interacting with the user is the small touchscreen. With a smaller display fewer elements can be displayed at once as seen in Figure1.1. Since screen of a smartwatch is smaller than many on-screen keyboard on smartphones and tablets, this puts a limit on the complexity of user inputs. The small screen limits the amount of information that can be displayed without sacrificing readability.

Compared to a smartphone, a smartwatch also comes with new security risks. For example there is research[25][17] showing that passwords typed can be recovered from motion data captured using wearable technologies.

1.2 Authentication

Identity-Based Credentials

For authentication Identity-based credentials are often used. Compared to attribute-based credentials, identity-based credentials are easier to implement. At its simplest both client and server agree on a shared secret when creating an identity and associating a level of authentication with that identity. When the client wants to authenticate, he does so by showing ownership of an identity with an appropriate level of authentication. To prove possession of that identity the user shows that he knows the secret associated with that identity. For example a user signs up for an online video streaming service where only authorized users can view certain movies. The user would create an account (identity) with a password (secret). During the creation of that identity the server could verify the legitimacy of the user, for example by making the user pay a fee. Finally, by associating each user with an identity, a user's behavior can easily be tracked across sessions. This results in a loss of privacy compared to the situation where individual sessions cannot be linked to the same user. The concept of identity-based credentials does come with drawbacks. Firstly the server has to keep track, and keep confidentiality, of shared secrets of its users. Secondly users have to keep track of many secrets, unique for each service and level of authorization. For example a user may use separate email accounts for both his professional and private life.

Attribute-Based Credentials

Attribute-based credentials or ABCs is a method for authenticating. Authentication is the act of proving to another party that you posses certain attributes. In this context a credential shall be a cryptographically signed attribute. Any party will be able to verify that a party has signed an attribute by using the signer's public key. This can be done either by proving the attributes or by proving an identity for which the other party knows the associated attributes.

An attribute is disclosed to a party when it learns the contents of that attribute. For example, a shop's owner can learn a person's nationality when they use a passport to prove their age.

Attribute-based credentials are a method of authentication that has eliminated several problems around identity-based authentication. The user does not prove an identity. Instead, they prove attributes they hold. Attribute authorities give users the information needed to construct proofs for attributes they hold. Service providers can verify these proofs using information published by the attribute authorities. An example can be given using the supermarket scenario described earlier. Using attribute-based credentials, when a customer is asked to prove they are of drinking age, they provide the supermarket a proof that they have the attribute "Over 18 years old". The supermarket can verify this proof to confirm the customer is actually of drinking age. The given proof contains neither the user's identity nor the exact age of the customer.

Attributes are information about a person such as their nationality, date of birth or

main bank account. An attribute is proven to another party if either the other party has verified this attribute itself, or instance a nation can verify that a person has their nationality by if the other party trusts whomever has verified the attribute. For checking their records, and border control may verify a nationality by checking the validity of a passport issued by that nation.

In contrast to attributes, an identity is a piece of information, like a serial number, uniquely attributed to one party for which a service provider knows the associated attributes. For instance if a supermarket issues a loyalty card and keeps track of the usage of that loyalty card, the supermarket can confirm that a user hold a certain attribute by checking a unique identifier or that loyalty card and verifying that it was used in a certain way.

Attribute-based credentials are based around the notion of zero-knowledge proofs. Zero-knowledge proofs allow one party to prove a statement to another party without revealing anything else about that statement. In the case of attribute-based credentials an attribute holder, using zero-knowledge proofs, can prove that he holds certain attributes without revealing his identity or other attributes the user holds.

One of the advantages of attribute-based credentials is unlinkability. Two authentication sessions to a party are unlinkable when that party cannot prove that the authentications are done by the same person. Linkability does not necessarily mean a party learns the person's identity. However, if a party knows the person's identity in both authentications, they are definitely linkable. Linkability is undesirable because it makes it easy to compromise the privacy of its users by allowing users to be tracked across sessions.

Another advantage is the straightforward use of selective disclosure. Selective Disclose is the ability to disclose only a subset of attributes. More privacy is preserved when an authentication can be done by disclosing only the attributes required for that authentication. Another useful property of this is the ability to re-use the same credentials for different requirements.

To get something resembling selective disclosure with identity-based credentials a user may try to create multiple identities with different sets of attributes associated with them. The problem with this is twofold. Firstly this is impractical or sometimes impossible in practice. Having to manage multiple identities for multiple services is not something users commonly do. Secondly if authentications are not unlinkable, for instance if both identities are linked to the same home address, then the full set of attributes can still be determined by combining knowledge of multiple identities.

There are two major schemes that allow for selective disclosure of credentials, Microsoft's U-Prove and IBM's Idemix. Both schemes are based on the verification of signed commitments. U-Prove [20] depends on the discrete logarithm problem. The holder has a secret key for each attributes and the issuer learns of the signed attributes of the holder. Through an interactive zero-knowledge protocol the holder can prove knowledge of specific attributes. Unlinkability requires new credentials for each authentication.

IBM's Idemix [8] depends on the RSA problem. Unlike U-Prove it requires only a single secret key for all attributes. The holder can use his own master keys for his

credentials. Issuing is done using a blind signature, where the issuer does not learn the resulting form of the credentials. Unlike U-Prove Idemix provides unlinkability without the issuer having to issue multiple credentials. Two projects are worth mentioning, ABC4Trust and the IRMA project, are based on the Idemix protocol.

There are two interesting project based on Idemix: IRMA, explained in section 1.3 and ABC4Trust. ABC4Trust [21] is a EU-funded project that researches and develops ABC systems. It has so far produced two pilots. Firstly a Greek pilot [13] that enables student to evaluate courses anonymously. Secondly a Swedish pilot [4] allowed students to anonymously access online resources.

1.3 IRMA Project

IRMA, short for I Reveal My Attributes, is an ecosystem for attribute-based credentials developed by the *Stichting Privacy by Design*[4] based on the Idemix technology. IRMA has several pilots with the Radboud University, Surfnet[5] and Alliander[6]. Besides these pilots it is possible to test the technology using a demo website[7] or setup your own server using the publicly available source code. This particular project is useful as an ecosystem for a proof of concept on a smartwatch. There already exists a working Android implementation on smartphones. Some current smartwatches run Android Wear, a modified version of Android. Given the similar operating systems, it should be easier to port the application to a smartwatch than if the smartwatch was of a completely different ecosystem. Even if the operating systems were similar, having to implement only an attribute holding application instead of implementing all parties is a useful advantage. All the official IRMA software is publicly available on GitHub at `https://github.com/credentials/`. Attributes can already be obtained and stored on MULTIOS smart cards[8] and Android smartphones[9].

For the purpose of these three pieces of software are relevant. `irma_api_server`[10] is software that allows for the creation and verification of IRMA attributes. One server can be used to issue credentials on behalf of multiple parties.

`irma_js`[11] is a JavaScript library for interaction with an instance of `irma_api_server`. `irma_js` and `irma_api_server` do not need to be controlled by the same entity. A company could use `irma_js` on its website and trust another party with the technical knowledge to run `irma_api_server` for verifying the attributes. `irma_android_cardemu`[12] is the Android version of the IRMA application for holding and managing the users

[4] `https://www.privacybydesign.foundation`
[5] `https://www.surfnet.nl`
[6] `https://www.alliander.com/nl`
[7] `https://demo.irmacard.org/`
[8] `https://github.com/credentials/irma_card`
[9] `https://github.com/credentials/irma_android_cardemu`
[10] `https://github.com/credentials/irma_api_server`
[11] `https://github.com/credentials/irma_js`
[12] `https://github.com/credentials/irma_android_cardemu`

attributes. The proof of concept developed for this thesis is largely based on the code of `irma_cardemu` and work in combination with `irma_js` and `irma_api_server`.

1.4 Related work

For a general overview of attribute-based credentials *Concepts and languages for privacy-preserving attribute-based authentication*[6] standardizes some concepts used in different scheme for attribute-based credentials and *The ABC of ABC*[14], explores the privacy aspects of attribute-based credentials.

For a more detailed look at IRMA, in *IRMA: practical, decentralized and privacy-friendly identity management using smartphones*[3] the use of IRMA on smartphones is explained and in *Towards Practical Attribute-Based Identity Management: The IRMA Trajectory*[2], the trajectory for the IRMA project as a whole is explained. The most prominent implementation of IRMA is on smartphones. Another interesting implementation of IRMA is *the implementation of IRMA on smart cards*[24]. Smart cards are more limited, both in memory capacity and processing speed, than other devices that run IRMA. The result of this is a simplified version of IRMA designed specifically for these low power devices.

A few papers focus on extensions of attribute-based credentials. This includes, delegation of credentials to other parties[10] the revocation of already issued credentials[7][18] and mutual authentication between holder and verifier[1].

Instead of authentication a user for a specific session, it is also possible to use *Attribute-based signatures*[19]. This allows attribute holders to sign a message allowing other parties to verify.

In addition to specific technical expansions there is research into the use of attribute-based credentials in specific use cases, including web services[22], eHealth systems[11], smart homes[23] and web-shopping in a privacy-preserving fashion[12].

Lastly, attribute-based encryption[9][5] is related, but different from attribute-based credentials. Unlike attribute-based credentials which provide authentication, attribute-based encryption allow for the encryption (and decryption) based of attribute. This allows for confidentiality between multiple users with the same attributes.

Chapter 2

Methodology

To evaluate the viability of Attribute-based credentials on smartwatch I look into multiple questions. Firstly, are smartwatches a viable alternative to other hardware for use of attribute-based credentials. Secondly, does the use of smartwatches have large advantage for specific uses of attribute-based credentials compared to other hardware. Thirdly, is the use of attribute based credentials technologically possible on a smartwatch.

To answer these questions I will explore the use cases of a smartwatch and create proof of concept. In the next few sections I will formalize some requirements, advantages and limitations of using attribute-based credentials on smartwatches. Finally, I will describe my setup for developing a proof of concept application, which I will then explore in greater detail in chapter 3.

2.1 Relevant Parties

In this thesis I will be referring to three relevant parties, the issuer, the reader and the holder.

- The Issuer is the party issuing credentials to holder.
- The Reader is the party that will verify the holder's credentials.
- The Holder is the party that provides credential to a reader

Just because attribute-based credentials can be used does not necessarily mean that attribute-based credentials are practical. For a more complete use case of an application using attribute-based credentials we require more. Ideally our application should be able to handle multiple attributes and selective disclosure. In addition to obtaining and disclosing credentials, we would like to be able to manage our credentials, allowing us to view and remove our stored credentials. Lastly when we have multiple credentials that can be used, we would like to select which credentials to use for authentication.

We can show the practicality of using a smartwatch over using any other device in two ways. Firstly we can find a use case where existing implementations have problems and

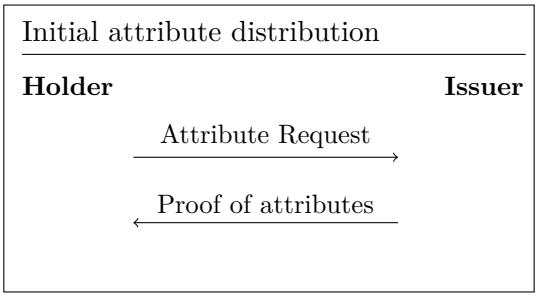

Figure 2.1: Minimal interactions needed for issuing attributes

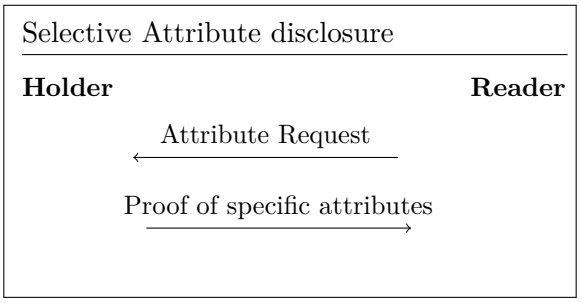

Figure 2.2: Minimal interactions needed of attribute disclosure

show how smartwatches overcome these problems. In section 2.2 I explore smartwatch specific use cases. Secondly we can find a use case where smartwatches can be used in the same way as current devices, showing that smartwatches are a viable alternative. In chapter 3 I explore the use of a smartwatch in the existing infrastructure of IRMA.

2.2 Technical requirements of use cases

To show that an application can authenticate using attribute-based credentials you need it to perform at least two actions. First it needs to be able to receive and store an attribute. Secondly it needs to be able to reveal a stored attribute. This would show that attribute-based credentials are technically possible. For this we need at least three parties. First we need an attribute holder, that will receive and reveal attributes. Secondly we need an attribute reader who want the holder to reveal their attributes. Lastly we need an attribute authority who is trusted by the reader to give authority to the given attributes.

2.3 Role of smartwatch during authentication

There are three roles a party can play; the issuer, the holder and the reader. Because of its portability, a smartwatch can be used in situations where carrying around a larger device, such as a smartphone, would be an issue. This makes a smartwatch practical in situations where a party has to move to a location and where the holder does not want to pull out his smartphone to authenticate.

The role of issuer does not require physical presence or a quick reply to unanticipated requests. Because of this there is no benefit to using a smartwatch as an issuer.

The use of smartwatch as a reader may not be practical. A reader may want physical proximity, e.g. during an age check in a supermarket. If the reader itself does not require mobility, the advantages of a smartwatch over a smartphone are limited.

If the attribute reader reads the attributes of many attribute holders from a single location, portability and initial setup time are not major issues. Furthermore, a change of operators of the attribute reader would be more difficult since the device is worn. Because of these limited advantages, I think smartwatches are a poor fit for reading out attributes.

This leaves holder authentication. The initialization is no better for a smartwatch, since it needs to be neither to be timely nor physically present. There are 2 types of scenarios to be considered when using a smartwatch as a holder. There are the scenarios where the smartwatch is used alongside a smartphone and the scenarios where a smartwatch is used independently.

The smartwatch will be used in the role of attribute holder. Therefore, only the use cases where an attribute holder would be involved are explored. The only required use cases where the attribute holder participates are the case where the holder is initialized with its proof of attributes and the case where it wants to (selectively) prove its attributes to another party.

It is possible to use a smartwatch and smartphone together as a holder party. In practice there would be several setups possible. One method would be to use the smartphone as a proxy for communications to the smartwatch. This would solve the problem of communication of a smartwatch with a third party. However, since the user has to initialize the session using the smartphone, it would likely be simpler to just use the smartphone.

Another setup is where the smartwatch acts like a proxy for the smartphone. Due to its wearable nature, a smartwatch can be used in situations where a smartphone cannot. For instance the user cannot store a device in a pocket or bag when not using it, or the user needs to have both hands free when using the device.

For a use case where smartwatches are more useful than other device for attribute-based authentication, we require two things. First we need a situation where attribute-based credentials are useful. This means a situation where authentication is needed and can be done by revealing only attributes. Secondly we need a situation where physical proximity matters or no other device (like a smartphone can be used). The situation where physical proximity matters useful for smartwatch use cases because it plays to the

strengths of a smartwatch. Situations where a smartphone cannot be used are interesting because often a smartphone is more practical than a smartwatch.

A simple use case would be a fast age check in a supermarket. A more convenient authentication would allow for faster processing of a queue of customers.

A more complex use case would be during a marathon race where racers where a party would like to verify contestants have passed all checkpoints. Carrying around a smartphone and taking it out at every stop would be inconvenient and slow down runners. Furthermore, attribute-based-credentials can be used to track progress.

- Before the race, the issuer issues the credential "is race qualified" to all racers.
- The first checkpoint a reader verifies that the holder is qualified to race and issues a credential to prove that it passed the first checkpoint
- At every following checkpoint the reader verifies that the holder has passed the previous checkpoint and issues a new credential verifying that it has passed the current checkpoint.
- At the end of the race a reader only has to verify that the holder passed the last checkpoint to verify that the holder has run the race fairly.

2.4 Development of Proof of Concept

IDE

Android studio is an IDE (integrated development environment) for Android development for both phone and Android wear. Although it is possible to develop Android applications with a simple text editor and the `android` command line application, the complexity of java and Android makes it difficult for someone not familiar with the system. For reference a minimal hello world application generated by Android-studio already has over 3500 files and folders.

What is particularly useful for Android wear development is the previewing of the UI without running the application. Most UI elements are defined by their activities. Activities are a collection of UI elements with an associated class that handles the user interactions. The definition of the user interface on Android is done using XML file. Manual editing of these files is a lot simpler with most text editors than using Android Studio. The updated activity UI can be previewed as it is being edited with Android studio. If you were to do this manually, you would have to rebuild the application every time to test the new UI.

This is important because screen space on the wear is much more limited when compared to smartphones and tablets.

Much of the development can be done with an emulator, emulating the smartphone. There is an official Android emulator that simulates tablets, phones and smartwatches. It allows users to test applications on the same machine they are developed on. You

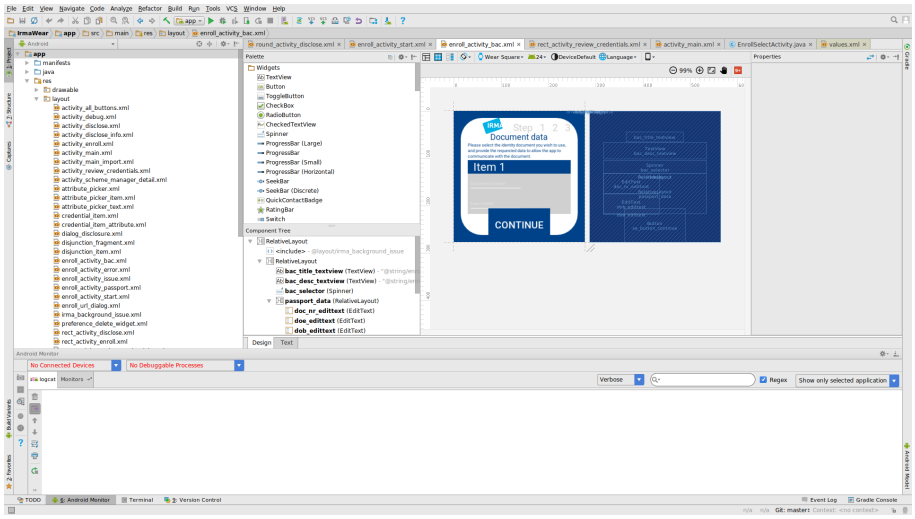

Figure 2.3: Screenshot of development environment showing a preview of the IRMAWear User Interface

can control the emulator with mouse and keyboard, clicking the screen with a mouse simulates a touch on a touchscreen.

The emulator works reasonably well however there are some differences from a physical watch. Firstly it can run on a different speed from actual devices. If an application runs smoothly in an emulator, it does not mean it will run smoothly on a physical watch and vice versa. Secondly the emulator accepts keyboard input, something you cannot do with a smartwatch. This allows the emulator be used in situation where a physical device cannot be used. The emulator does not have a paired phone, shares the internet connection of its host and does not use Bluetooth. This may be an issue when trying to test for setting up connections to other devices. The emulator has no sensor data, which may be a problem for some use cases, but was not an issue in this thesis. Because the emulator could not accurately emulate real use cases of the smartwatch, I later developed using just a physical watch.

Differences phone/wearable development

Android apps developed for phones and tablets don't automatically work on Android smart wear and vice versa. Firstly this is because the screen interface is much smaller, so small that an on-screen keyboard is not practical and only 2 or 3 onscreen buttons would be feasible. Voice commands and swipe gestures (movements across the screen) might be valid input methods but input remains limited. Secondly android wear requires a newer API than some devices provide. Most apps don't require the newest API and can work on older android version. Apps designed for smart wear however require a newer API just to run.

The process of developing of an app for Android Wear is very similar to that of

developing an app for Android. Android phones and tablets can use the exact same applications since they run the same operating system. UI elements will simply be scaled based on the size of the screen. You can compile one `apk` (Android Application Package) that installs on both tablets and phones. The Android wear applications have to be designed and compiled separately. An `apk` compiled for phones/tablets will not install on an Android wear and vice versa. You can import classes from phone/tablet applications if you have the source code but the UI elements have to be made from scratch.

In android studio, you can develop an app for both wearable and phone/tablet at the same time. Each version (wear or mobile) of the app will have some unique files, allowing each to have a unique interface while sharing the same libraries. When compiling, android studio generates two different apps (apks), one for mobile and one for wearable. The app for the wearable device will not necessarily run on mobile and vice versa due to different requirements on the systems.

Device used

After initially developing on an emulator, I continued developing on a physical watch. I used the *Sony Smartwatch 3*[1] as a physical smartwatch. Multiple models were available but while there were large differences in price, the differences relevant to developing applications were minor. I chose this specific watch because it was cheap (compared to the other models) and had all the required features. I could successfully use the IRMA app on a *Samsung Galaxy J1 smartphone*[2]. Beside a larger touchscreen and a camera, the smartphone did not outperform the smartwatch in specifications. Because of this I was confident the smartwatch could run IRMA as well.

[1] Specifications detailed in appendix 4.2
[2] Specifications detailed in appendix 4.2

Figure 2.4: `IRMAWear` running on physical watch

Chapter 3

Proof of Concept

It took several iterations to get attribute-based credentials working on a smartwatch. This was largely due to a lack of experience with android development. I first created small apps to test the build process for Android and Android Wear applications and figure out how setup and analyze basic functionality like UI interaction and communication with other devices. After that, over the course of several iterations I created a working IRMA implementations. The next few sections explain the intermediary attempts with section 3.3 explaining the final iteration of the proof of concept.

3.1 Initial Android wear apps

First I created a simple application to test that

- The Android development environment works.
- I can create a functional application and run it with the emulator
- The application can make an internet connection with another application.
- The application can keep an Internal state.

The application is a simple challenge response system that sends messages back and forth as plain text. In addition to the application I made a helper program outside of the Android application. First I made a simple proxy server that logged incoming and outgoing connections. Secondly I created a python script that interacted with the Android application. To analyze the communication I setup a relatively simple network. In between each communication there is a man in the middle proxy that log all messages being sent back and forth. The sources of these are included in appendix 4.2

I had written these tools for two reasons. First it made it easier to debug any network issues. Secondly I suspected the protocols may have to be adjusted. This setup allowed me to model an attacker being able to view all traffic to see what new information can be learned from this new setup.

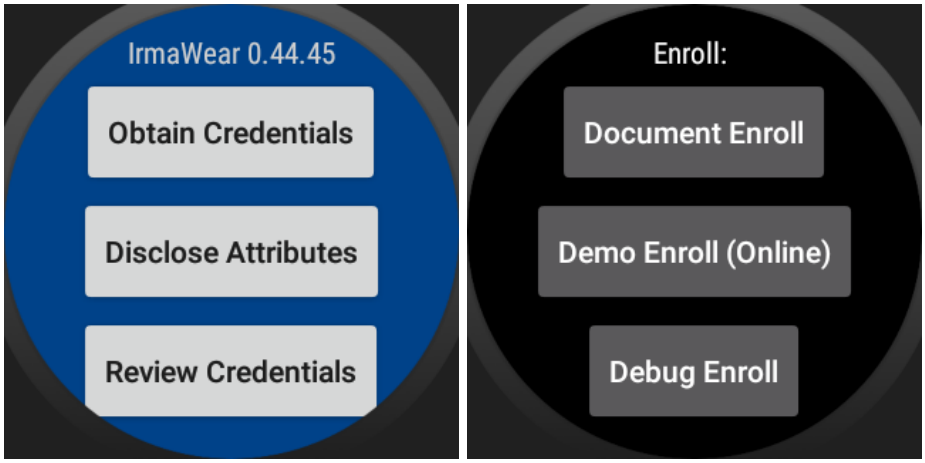

Figure 3.1: Screenshots of menus of `IRMAWear` running on an emulator

3.2 IRMAWear

Next I implemented an IRMA version for Android wear. I named it `IRMAWear` to avoid confusion with the existing IRMA implementation on Android that works with smartphones and tablets but not with smartwatches. The source code of the application is available online at `https://bitbucket.org/martijntje/irmawear/`.

Initially I started by creating a simple smartwatch app extending it with the parts from the existing IRMA application. This process was slow and did not seem viable.

Instead, I started again. I created a new empty Android wear application and added all code from the Android application into `IRMAWear`. This led to a code base which included the full IRMA application but could not compile for Android wear.

I then took the code base and I gradually tried compiling the application. Each time an error returned I found the source of the error and either commented out the problematic parts or replaced them with dummy versions. In the end I did not have to change any of the cryptographic libraries. However, I did change most of the code regarding UI and user interaction.

After that I recreated the functionality of the first simple application so I could easily test the application. I removed most of the original user interface to make `IRMAWear` work. Although I included the code of the original IRMA application there was no way for me to access this functionality without a proper user interface. On smartwatches buttons need to be large compared to the screen size. If the buttons are too small, pressing the current button become error-prone. I found that three buttons stacked vertically works well.

When there is more functionality than a user can use with the buttons on the screen, it is best to make a button to open new sub-menu with new buttons of it own. When you create a new sub menu, the best way to do so is by creating a new activity. This way you waste no screen space on a button because the user can swipe right to left to return to the parent menu.

Obstacles encountered with IRMAWear

I did not get a working IRMA smartwatch app on my first attempt due to several issues.

The smartphone IRMA app can enroll using the demo website[1] without needing a second device. It does so by opening the demo website on the phone where the user can enter his/her details. I initially planned to recreate this but could not do so. The first problem is with inputting the user data. The online enrollment requires the user to input his data. Entering text is possible on an emulator, however a real smartwatch has no keyboard. The smartwatch API provides no method for selecting dates are inputting text. [2]

Secondly the smartwatch app tries to display the demo web page for issuing credentials. Since the Android wear version I tested has no web browser (or method of installing one), the application simply crashes when trying to open a web page.

Another large problem when trying to port the existing Android IRMA application to the smartwatch was using the existing communication channels. The Android application can be easily tested using a phone by visiting the IRMA demo website[3] and scanning a QR code. This QR code contains the information needed to set up a connection between the smartphone and the demo server. Since smartwatches have no cameras, this cannot be done with just a smartwatch. This poses a problem because the smartwatch can not be used as a drop-in replacement for the smartphone. At the very least the initial session info has to be receive by any IRMA client application. The `IRMA` session cannot be setup unless the IRMA client app can communicate with the server.

I looked into several alternative methods of communications. However, the downside of this is that both client and server have to be modified to allow for a new transfer method. Before I solved this problem, as outlined in section 3.3, I looked at a few other options.

One possibility is to communicate via sound. This should technically be possible since Pied piper[4] is an example of software that can transfer data as sound so it is possible. Another android specific program is quietmodem[5]. Downsides would be limited data transfer speeds (due to noise) and easy eavesdropping of communications.

I could also communicate via a USB cable using `adb` (Android Debug Bridge), this is a developer feature not normally enabled and has some security risks. Once an Android device trusts a computer connected via a USB cable, that computer can issue arbitrary commands to the device.

Other option was sending the initial session info over Bluetooth. Ideally only the initial session info needs to be send over a second channel. This was a problem with Bluetooth. On the smartwatch I tested, I could not use WiFi and Bluetooth at the same

[1] `https://demo.irmacard.org/`

[2] Technically it is possible to copy selected text to a clipboard and paste it into a text input field, but this is impractical

[3] `https://demo.irmacard.org/tomcat/irma_api_server/examples/issue-all.html`

[4] `https://github.com/rraval/pied-piper`

[5] `https://github.com/quiet/org.quietmodem.Quiet`

time. This means that, even if I could receive the session info, I could not connect to the IRMA server unless it was also Bluetooth equipped and within range.

Another possibility would be NFC. This would allow the user to receive the session info easily by simply holding the smartwatch in range of the NFC reader. Unfortunately I ran into technical problems here. The specifications of the smartwatch imply that NFC should work. When I tested it, I could not easily get an NFC reader to connect to the device. The NFC reader could read other NFC devices (passport and *OV-chipkaart*), but did not detect the smartwatch. Because I later solved the problem of receiving the session info using another channel, as detailed in section 3.3, I did not continue looking into communication using NFC.

Lastly I could receive the session info using the smartphone and send it through to the smartwatch. This works as a proof of concept but is not practical. If a smartphone can be used to receive the message, the user might as well use IRMA on his smartphone.

When paired with a phone, Android Wear applications cannot directly access the Internet. They must communicate with their corresponding handheld app (either via MessageApi or DataApi) and request that it executes whatever HTTP requests you need.

There are libraries to work around this issue [6] [7] by having the phone act as a proxy to the internet without disabling Bluetooth. Since this would require the use of a paired smartphone for internet connectivity, this would have no obvious advantage over using IRMA on the smartphone.

Lastly it was impossible to display all the same information onscreen as the smartphone app. I could not reuse the existing user interfaces without modifications. This is mainly because the screen size of even the largest Android wear smartwatch is a fraction of the screen size of most smartphones.[8] The existing Android IRMA application sometimes displays multiple paragraphs of text. This is used for explaining IRMA when receiving attributes from within the app and when viewing the stored attributes. Display paragraphs of texts works on a smartphone but is more complicated on a smartwatch. Since the screen is small, the application can display only a limited amount of information. Due to this it may be advisable to reverse as much screen estate when you need to display text. To a certain amount you could use a smaller font, however this is not ideal since this does make text harder to read. Requiring users to have perfect eyesight is probably unreasonable. You could also make text fields scroll-able, allowing users to use swiping gestures to move through text. In a later iteration of the app I used the `<scrollview>` element liberally allow all elements to fit the user interface. The cost here is that the user has to scroll to view all the required information.

[6] `https://developers.google.com/android/reference/com/google/android/gms/wearable/MessageApi`

[7] `https://developers.google.com/android/reference/com/google/android/gms/wearable/DataApi`

[8] see figure 1.1 for an example

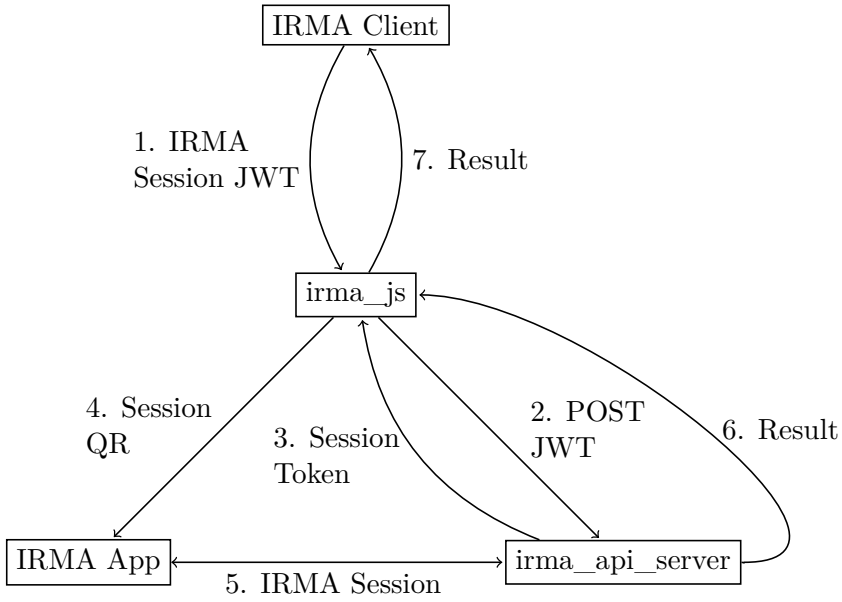

Figure 3.2: Overview of existing IRMA infrastructure replicated from the online IRMA documentation at `https://credentials.github.io/`

3.3 IRMAWear2

After reaching major roadblocks with the previous smartwatch app, I decided to write a new app. I named the latest iteration of my smartwatch App `IRMAWear2`[9]. The approach used here is different from the previous app. In the previous app I modified existing code until I got an application that ran. This did not work well because I spend a lot of time fixing compile time messages due to small differences in the Android and Android Wear ecosystem. This time I started with a fresh smartwatch application and slowly incorporated elements from the existing IRMA infrastructure into it. Because of this iterative process I had a working app from first to final iteration, which sped up development.

Integration with existing IRMA infrastructure

Ideally, in order for `IRMAWear2` to work with the same use cases as the smartphone, the smartwatch would reuse the existing IRMA infrastructure. Figure 3.2 show the data flow in an IRMA session. The arrows represent data flow and the numbers indicate chronological order. JWT is short for *JSON Web Token*, a data format. [10] The IRMA Smartwatch App would take the same role as the existing IRMA App. Because of this we only need to account for steps 4 and 5 in our proof of concept. In step 4 we obtain

[9]Source code is publicly available online at `https://bitbucket.org/martijntje/irmawear2`.
[10]See also `https://tools.ietf.org/rfc/rfc7519.txt`

information from `irma_js` on how to contact `irma_api_server`. Once a connection with`irma_api_server` has been setup, no further contact with `irma_js` is needed for the current session.

Normally the Smartphone App obtains session information by scanning a QR code in step 4 from `irma_js` and in step five communicates with the `irma_api_server` via HTTP.

I cannot treat `IRMAWear2` like a black box and mimic the behavior of the smartphone app exactly. Step 5, the communication with the `irma_api_server` relies on a stable internet connection. The only challenge here is forcing the smartwatch to use WiFi. I could recreate Step 5 without much effect. The smartwatch I tested could not connect to both Bluetooth and WiFi at the same time. I could force the smartwatch to use of WiFi by making the paired smartphone unavailable via Bluetooth, either by disabling Bluetooth on the phone or by placing it out of range.

I could however, not do step 4 (receiving the session info via QR) without a slight modification of the session. Displaying and scanning a QR code work well with a smartphone, however it relies on the use of a camera. Unlike smartphones, smartwatches lack a camera and can therefore not scan QR codes. One method around this is using `adb`, a developer feature that allows for easy interactions with an android device. Using `adb` for communication works in principle but is not practical. Firstly it requires the smartwatch user to know of this hidden feature. Secondly it requires the smartwatch user to explicitly confirm trusting a computer each time it connects to a new computer. Thirdly it requires a connection via a USB cable, which may not be possible while wearing the watch. Lastly and perhaps most importantly, `adb` gives the connected computer full shell access to a device, allowing it to easily compromise the devices' security without the user noticing.

To get around the problems with sending the session info I tested two modifications of this information flow. Figure 3.3 shows the first modification. I add one extra step where the smartwatch app sends its own contact info to `irma_js`.

`irma_js` is written using JavaScript and can be run, outside a web browser, from the command line for testing. Normally these scripts display the session code using a QR displayed with Unicode block characters. I modified this script[11] such that it outputs the session info is outputted to stderr. This way the message can be piped to a shell script which handles setting up the connection with the smartwatch. This method worked but required the manual writing of JavaScript for individual uses. Compared to using the demo website this process was not user-friendly.

The method I used here was to display this information as a qr code on the smartwatch's screen and read it using my laptops webcam. This qr code would contain the smartwatch local ip address and port number it is listening on. This work as long as `irma_js` and `IRMAWear2` can communicate each other via IP. I accomplished this by running my own slightly modified instance of `irma_js` (and `irma_api_server`). I did this by running `irma_js` outside a web browser using the node command line version.

[11] the nodejs modifications are available online at `https://bitbucket.org/martijntje/irma_makefile/src`

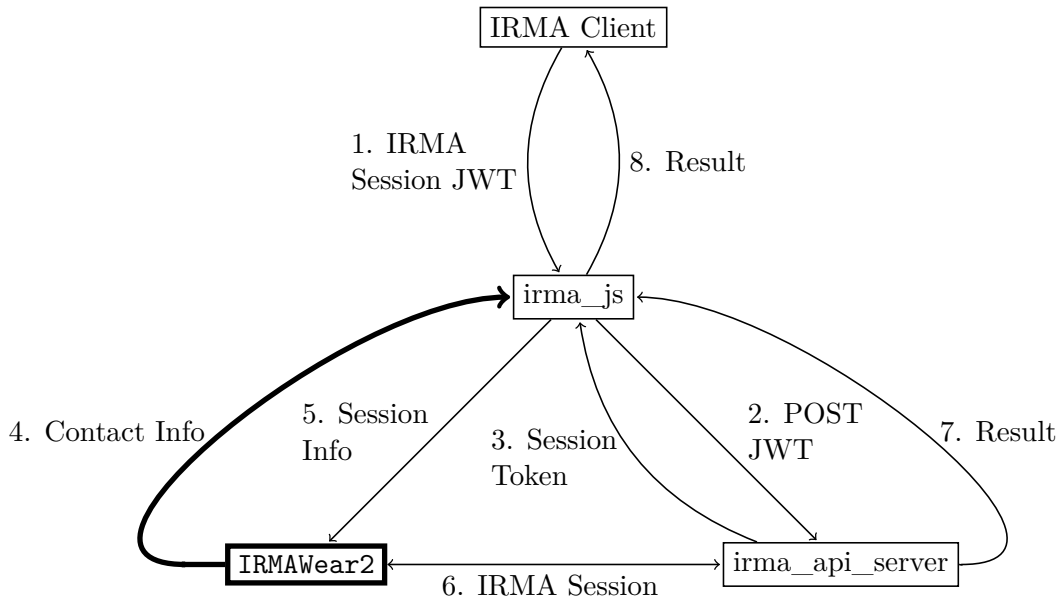

Figure 3.3: Modification 1: Modifying `irma_js` to listen for an incoming connection first

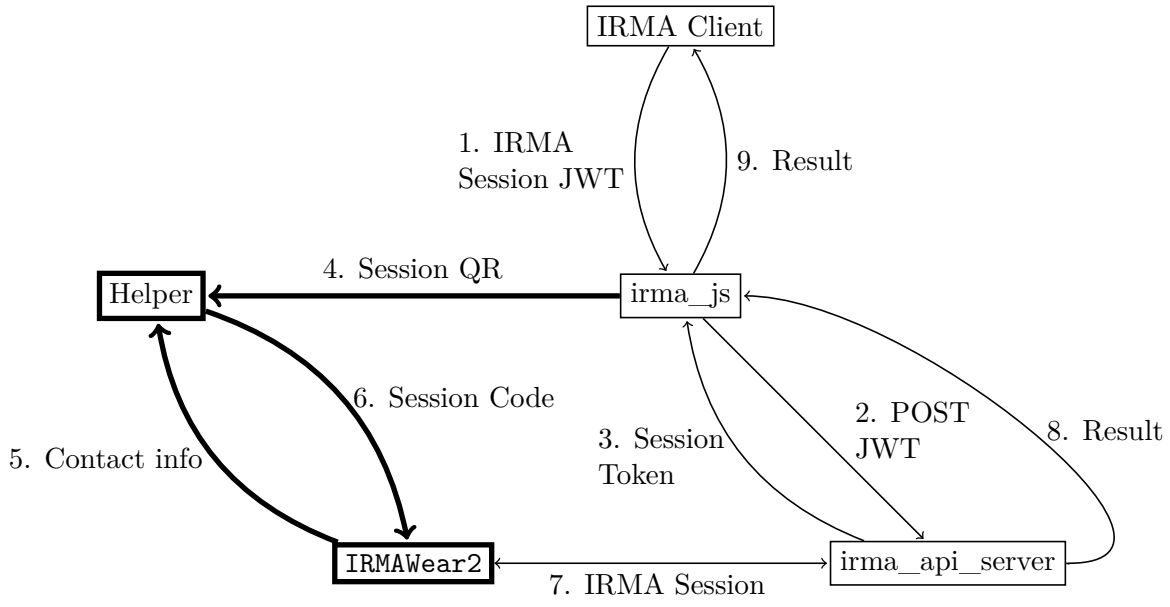

Figure 3.4: Modification 2: Adding a helper agent to help communication between `irma_js` and the app

Figure 3.5: Screenshot of initial view of `IRMAWear2`.

Figure 3.6: Screenshots of the QR code displayed to set up a connection and active communication.

I tested a second workflow, showing in Figure 3.4, which solves the communication problem by added an extra agent. This is explained in more detail in section 3.3. Although this setup has more agents and steps, this setup works well and works with an unmodified `irma_js`.

User Interface

I created a custom User Interface (UI) for `IRMAWear2`. This was necessary for two reasons. Firstly because, as discussed in section 3.3, the app requires a modified workflow because it can't read QR codes. Secondly because of its small display fewer UI elements fit on the screen at once. To avoid navigating through menus or using small text I have opted to nest the main user interface in a `scrollview` element. The result of this is that you can access most elements by scrolling using the touch screen.

The main view, without scrolling looks as displayed in figure 3.5. This view the title of the app, and either a *connect* button when idle or a status message when busy.

When clicking the *connect* button the UI displays a QR code, as shown in figure 3.6. The text above the qr text is the same as the content of the qr code and is only used for human feedback. After a connection has been setup and the session with `irma_api_server` has started, the request will be displayed onscreen like in figure 3.7. Some graphical elements do not fit the small screen space, however all interactions work

Figure 3.7: Screenshots of dialog for disclose or attributes.

Figure 3.8: Two screenshots showing the interface for viewing attributes in IRMAWear2.

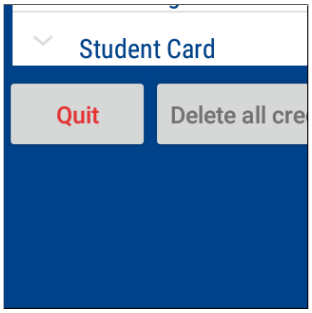

Figure 3.9: Screenshot of Interface below attribute view

properly.

Once the user has attributes stored using the app, the user can scroll down select the attribute from the list and look at its values as seen in figure 3.8.

Below the list of attributes there are two more buttons as seen in figure 3.9. The quit button it there for convenience, and the “delete all credentials” allow a user to reset the stored information of the application.

Communication setup

For the smartwatch to use IRMA properly it is important that the `irma_api_server` or `helper` program can easily contact the smartwatch. The easiest way is to use WiFi with both parties connected on the same network. To create a stable environment, I use `create_ap`[12], a script that easily sets up a quick WiFi access point on my laptop for the smartwatch to connect to. This is advantageous to simply connecting both devices to the same network directly for two reasons. Firstly my laptop can connect to certain networks the smartwatch cannot, most noticeably `WPA Enterprise` networks. Secondly my laptop has a better range for WiFi than the smartwatch.

To make the modification described in figure 3.3 work I compiled my own version if `irma_api_server` and `irma_js`. This build included script to communicate to the smartwatch over `adb` and custom JavaScript to allow the outputting via a different source than displaying a QR code. To automate the process of setting up a custom IRMA server, using `make`, I created a makefile[13] to automatically setup a local server. To test the modification described in figure 3.3 I wrote a small script, `qr_bridge`[15] that first, reads a qr code using a webcam, and then captures a qr code from the computer screen. This allows the smartwatch to be used with unmodified servers. The script relies on `zbar`[16] for quickly parsing qr codes.

App dependencies

In order to display a QR code on the smartwatch I make use of the `zxing`[17] library. I did not investigate any other libraries because I did not find any fault with this one. Besides that `IRMAWear2` relies on the IRMA libraries `credentials_api`[18], `credentials_idemix`[19], `irma_api_common`[20], and `irma_configuration`[21], which are also used in the smartphone app.

I also imported the code for the smartphone implementation, `irma_android_cardemu`[22], with a few modifications to make it work as a library instead of a standalone application. I had to add the following line to its `build.gradle` file to allow it to compile as a library

```
apply plugin: 'com.android.library'
```

In addition to treading the smartphone app as a library, I changed the names of the `MainActivity` class and `activity_main` activity to avoid name conflicts with the

[12] `https://github.com/oblique/create_ap`
[13] files are available online[14]
[15] `https://bitbucket.org/martijntje/qr_bridge`
[16] `http://zbar.sourceforge.net/`
[17] `https://github.com/zxing/zxing`
[18] `https://github.com/credentials/credentials_api`
[19] `https://github.com/credentials/credentials_idemix`
[20] `https://github.com/credentials/irma_api_common`
[21] `https://github.com/credentials/irma_configuration`
[22] `https://github.com/credentials/irma_android_cardemu`

smartwatch versions of these files. I could also have deleted these files for the same result.

New Code

Besides reusing the existing IRMA software, `IRMAWear2` has 3 original classes. To avoid littering the code base with classes, I have written most code in a single class, `MainActivity`. In addition to the main class, I had to create two more classes for specific tasks.

Android will automatically kill any application that takes longer than 5 second to respond to user input. To prevent this you have to create a new thread. Since multi threading requires new threads to be their own class, I had to introduce 2 additional classes. The `BackgroundClient` class listens on a socket and handles the initial session info message. The class `AsyncQRUpdate` generates the QR images used in figure 3.6. While the creation of the qr code could run on the main thread, it occasionally lags enough that if ran on the main thread would crash the entire application.

3.4 Limitations

While the produced app show that IRMA can run on a smartwatch nicely, it does have some limitations. Most obviously it was only tested on a single device. I suspect the app will behave similarly on different Android Wear watches, but could well have some display issues as different watch may have a smaller or differently shaped display (e.g. there are smartwatches with a round instead of rectangular display). I had to simplify some interactions, to make them work on a smartwatch. The dialog for disclosing and issuing attributes do not have a “more information” option, as is present in the smartphone version, and the option to delete credentials does so without displaying a confirmation dialog. The Dialog used during the issuing and verifying of credentials is functional if you know what is going on but cannot display the full information. I wrapped the UI elements wrapped in a purposefully over sized `scrollview`. This means that user can scroll down multiple screens beyond the lowest element. This is because of a quirk in the UI element. The `ExpandableList` element cannot be of variable size inside a variable size view. Either the list of credentials large enough to accommodate all possible attributes fully expanded or the scroll view is large enough to accommodate for an expanding `ExpandableList`. A fixed sized attribute list is less desirable because this would mean that any UI elements below the list would be impractical to use.

Chapter 4

Results & Future work

4.1 Results

In this thesis I have shown attribute-based credentials to work on a smartwatch. I have done so by creating a proof of concept that can interact with the existing IRMA infrastructure to obtain, manage and reveal attributes with other parties. Due to the nature of the smartwatch I had to change the communication channels. Rather than the app reading the session info from the server to start the IRMA session, the smartwatch now first has to send its own information to the server or party acting as a proxy to the server.

The only thing exposed by intercepting this new initial message is how to contact the smartwatch. Although this did slightly change the interaction a user has with an app, since no underlying cryptographic protocols were changed, this did not impact the security. At best this makes it slightly easier to set up a man-in-the-middle scenario. As long as the original protocol is resistant against man-in-the-middle attacks, this setup should also be resistant.

4.2 Future Work

A major remaining issue is usability. There is the issue of finding a practical use case. I have shown that a smartwatch is a viable alternative in use cases where a smartphone was already viable. The use of a smartwatch for attribute-based credentials is easy to justify with a use case where the use of a smartphone is difficult but the use of a smartwatch is not.

While a smartwatch can be used, a smartphone still has advantages. The smartphone is more ubiquitous, has a larger screen and allows for text input.

Besides finding an appropriate use case, there were technical issues that may be different on other devices. There were some technical issues I encountered that may not be present on other smartwatch models. The smartwatch did not allow for text input, limiting the complexity of user interaction. The watch I tested ran Android Wear 1.5. Recently Google released Android Wear 2.0. This new operating system may solve some

technical problems encountered. For instance this new operating system allows for the use of an on-screen keyboard controlled by gestures. Another roadblock I encountered was the difficulty using the smartwatches communication interfaces independently. The smartwatch communicates with the smartphone using either WiFi or Bluetooth. It can however not use WiFi and Bluetooth at the same time and will not connect with a third party unless the user explicitly disables communications with the phone. Watches based of other operating systems like `WatchOS` or `Tizen` might not have the same technical problems. If a new smartwatch also includes a fully featured web browser, it could be reasonable create an app that interacts with the IRMA demo website.

I designed the smartwatch app as a proof of concept without prior knowledge of android development. As a result of this, the user interface is lacking. The user interface was not optimized for display on a smartwatch, requiring scrolling to view parts of user interface elements.

The programming was done to produce a working app in a short time, rather than ensuring long term quality. Because of this, the app was not designed to run on any smartwatch than the one I used. Furthermore the app relies on the current IRMA infrastructure, so any future changes to the IRMA protocols would cause the app to stop working with existing infrastructure.

Currently there is no clear use case for attribute-based credentials where the usage of a smartphone is difficult but the usage of a smartwatch is not.

I have not fully explored all possibilities for setting up a IRMA session using a smartwatch. It may be possible to set up a connection without the smartwatch having to announce its contact info first, like is the case with a smartphone. This should be possible with NFC, but I did not explore this. It may be useful to allow for authentication without confirmation.

e.g. A user may want to automatically confirm age verification if he knows the session can only be initiated when the server is physically present.

Bibliography

[1] Gergely Alpár and Jaap-Henk Hoepman. A secure channel for attribute-based credentials:[short paper]. In *Proceedings of the 2013 ACM workshop on Digital identity management*, pages 13–18. ACM, 2013.

[2] Gergely Alpár and Bart Jacobs. Towards practical attribute-based identity management: The irma trajectory. In *IFIP Working Conference on Policies and Research in Identity Management*, pages 1–3. Springer, 2013.

[3] Gergely Alpár, Fabian van den Broek, Brinda Hampiholi, Bart Jacobs, Wouter Lueks, and Sietse Ringers. Irma: practical, decentralized and privacy-friendly identity management using smartphones.

[4] Souheil Bcheri, Norbert Goetze, Monika Orski, and Harald Zwingelberg. D6. 1 application description for the school deployment. Technical report, Technical report, ABC4Trust, 2012.

[5] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 321–334. IEEE, 2007.

[6] Jan Camenisch, Maria Dubovitskaya, Anja Lehmann, Gregory Neven, Christian Paquin, and Franz-Stefan Preiss. Concepts and languages for privacy-preserving attribute-based authentication. In *Policies and research in identity management*, pages 34–52. Springer, 2013.

[7] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. *Advances in CryptologyEUROCRYPT 2001*, pages 93–118, 2001.

[8] Jan Camenisch and Els Van Herreweghen. Design and implementation of the idemix anonymous credential system. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 21–30. ACM, 2002.

[9] Melissa Chase and Sherman SM Chow. Improving privacy and security in multi-authority attribute-based encryption. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 121–130. ACM, 2009.

[10] Manu DRIJVERS, Jaap-Henk HOEPMAN, and Bart JACOBS. Efficient delegation of idemix credentials. 2014.

[11] Linke Guo, Chi Zhang, Jinyuan Sun, and Yuguang Fang. Paas: A privacy-preserving attribute-based authentication system for ehealth networks. In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, pages 224–233. IEEE, 2012.

[12] Brinda Hampiholi and Gergely Alpár. Privacy-preserving webshopping with attributes.

[13] Abendroth Joerg, Vasiliki Liagkou, Apostolis Pyrgelis, Christoforos Raptopoulos, Ahmad Sabouri, Eva Schlehahn, Yannis Stamatiou, and Harald Zwingelberg. D7. 1 application description for students. 2012.

[14] Merel Koning, Paulan Korenhof, and Gergely Alpár. The abc of abc-an analysis of attribute-based credentials in the light of data protection, privacy and identity-. 2014.

[15] Wei-Han Lee and Ruby Lee. Implicit sensor-based authentication of smartphone users with smartwatch. In *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*, page 9. ACM, 2016.

[16] Wei-Han Lee, Xiaochen Liu, Yilin Shen, Hongxia Jin, and Ruby B Lee. Secure pick up: Implicit authentication when you start using the smartphone. In *Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies*, pages 67–78. ACM, 2017.

[17] Xiangyu Liu, Zhe Zhou, Wenrui Diao, Zhou Li, and Kehuan Zhang. When good becomes evil: Keystroke inference with smartwatch1. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1273–1285. ACM, 2015.

[18] Wouter Lueks, Gergely Alpár, Jaap-Henk Hoepman, and Pim Vullers. Fast revocation of attribute-based credentials for both users and verifiers. *Computers & Security*, 67:308–323, 2017.

[19] Hemanta K Maji, Manoj Prabhakaran, and Mike Rosulek. Attribute-based signatures. In *Topics in Cryptology–CT-RSA 2011*, pages 376–392. Springer, 2011.

[20] Christian Paquin and Greg Zaverucha. U-prove cryptographic specification v1. 1. *Technical Report, Microsoft Corporation*, 2011.

[21] Ahmad Sabouri, Ioannis Krontiris, and Kai Rannenberg. Attribute-based credentials for trust (abc4trust). In *International Conference on Trust, Privacy and Security in Digital Business*, pages 218–219. Springer, 2012.

[22] Hai-bo Shen and Fan Hong. An attribute-based access control model for web services. In *Parallel and Distributed Computing, Applications and Technologies, 2006. PDCAT'06. Seventh International Conference on*, pages 74–79. IEEE, 2006.

[23] Koen van Ingen, E van Gelderen, and BPF Jacobs. Attribute-based authentication & signatures for regulating home access. 2016.

[24] Pim Vullers and Gergely Alpár. Efficient selective disclosure on smart cards using idemix. In *IFIP Working Conference on Policies and Research in Identity Management*, pages 53–67. Springer, 2013.

[25] He Wang, Ted Tsung-Te Lai, and Romit Roy Choudhury. Mole: Motion leaks through smartwatch sensors. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pages 155–166. ACM, 2015.

[26] Chao Xu, Parth H Pathak, and Prasant Mohapatra. Finger-writing with smartwatch: A case for finger and hand gesture recognition using smartwatch. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, pages 9–14. ACM, 2015.

Glossary

	Credentials	cryptographically signed attributes
	Idemix	Attribute-based credential scheme (Used in IRMA)
	Selective disclosure	Ability to disclose only a subset of all attributes during authentication
	U-Prove	Attribute-based credential scheme
	Unlinkability	Inability by other parties to know two sessions have the same participants
	Zero-knowledge proof	Protocol for proving knowledge of a fact without reveal that fact
IBC	Identity-based credentials	Authentication based on Identity management
IRMA	I Reveal My Attributes	Ecosystem for attribute-based credentials
ABC	Attribute-based credentials	Authentication using attributes
`JWT`	Java Web Token	Data format used by `irma_js` and `irma_api_server`
`adb`	Android Debug Bridge	Developer tool for interacting with Android and Android wear devices
`IDE`	Integrated development environment	Set of integrated software for developing software
`UI`	User Interface	Part of software that directly interacts with user
`apk`	Android Application Package	format for distributed Android software
`nodejs`	Node.Js	Popular JavaScript Library

Software sources

IRMA	
Demo website	https://demo.irmacard.org/
Main Repository	https://github.com/credentials/
Android App	https://github.com/credentials/irma_android_cardemu
irma_api_server	https://github.com/credentials/irma_api_server
irma_js	https://github.com/credentials/irma_js
Newly written software	
Helper scripts	https://bitbucket.org/martijntje/abc-wear/
IRMAWear	https://bitbucket.org/martijntje/irmawear/.
IRMAWear2	https://bitbucket.org/martijntje/irmawear2
Server setup Makefile	https://bitbucket.org/martijntje/irma_makefile
QRProxy	https://bitbucket.org/martijntje/qrproxy
qr_bridge	https://bitbucket.org/martijntje/qr_bridge
Watch test	https://bitbucket.org/martijntje/watchtest
Other Resources	
Pied Piper	https://github.com/rraval/pied-piper
org.quietmodem.Quiet	https://github.com/quiet/org.quietmodem.Quiet
create_ap	https://github.com/oblique/create_ap
zxing	https://github.com/zxing/zxing
zbar	http://zbar.sourceforge.net/

Specifications Sony Smartwatch 3

Specification as available from the manufacturer's website[1]

Requirements	Android 4.3 and onwards / AndroidTMWear[2]
Water protected	[3]
IP68 rated	
Performance	Quad ARM A7, 1.2 GHz/ 512 MB RAM, 4 GB eMMC
Sensors	Ambient light sensors / Accelerometer / Compass / Gyro / GPS
Battery	420mA (up to 2 days' normal use)
Optional accessories	Wrist straps in various colors (sold separately)
Color choices (wrist strap)	Black / Lime / Stainless steel / Leather Brown / Leather Black
Controls	Voice, touch, and gesture input / Microphone / On/off/wake-up key
Connectors	Bluetooth®4.0 / NFC / Micro USB / Wi-Fi

[1] `https://www.sonymobile.com/us/products/smart-products/smartwatch-3-swr50/specifications/`

[2] SmartWatch 3 SWR50 supports up to AndroidTMWear 1.5. Android Wear 2.0 and onwards are not supported.

[3] The SmartWatch 3 is water and dust protected as long as you follow a few simple instructions: all ports and attached covers are firmly closed; you can't take the smartwatch deeper than 1.5m of water and for longer than 30 minutes; and the water should be fresh water. Casual use in chlorinated pools is permitted provided it's rinsed in fresh water afterwards. No seawater and no salt water pools. Abuse and improper use of device will invalidate warranty. The smartwatch has an Ingress Protection rating of IP68.

Specification Samsung J100H

The smartphone I used alongside the smartwatch is a Samsung J100H. This is a model I personally own. The specifications were taken from Samsungs website[4] and manually verified on the phone using the Device explorer app [5].

Processor	
CPU Speed	1.2GHz
CPU Type	Dual-Core
Display	
Size (Main Display)	4.3 inch (109.2mm)
Resolution (Main Display)	480 x 800 (WVGA)
Technology (Main Display)	TFT
Color Depth (Main Display)	16M
S Pen Support	No
Camera	
Video Recording Resolution	HD (1280 x 720)@30fps
Main Camera - Resolution	CMOS 5.0 MP
Front Camera - Resolution	CMOS 2.0 MP
Main Camera - Flash	Yes
Main Camera - Auto Focus	Yes
Memory	
RAM Size (GB)	0.5 GB
ROM Size (GB)	4 GB
Available Memory (GB)	2.04 GB
External Memory Support	MicroSD (Up to 128GB)

[4] `http://www.samsung.com/my/support/model/SM-J100HZBDXME`
[5] `https://github.com/iamtrk/Device-Explorer`

Network/Bearer	
Multi-SIM	Dual-SIM
SIM size	Micro-SIM (3FF)
Infra	2G GSM, 3G WCDMA
2G GSM	GSM850, GSM900, DCS1800, PCS1900
3G UMTS	B1(2100), B8(900)
Connectivity	
USB Version	USB 2.0
Location Technology	GPS, Glonass
Earjack	3.5mm Stereo
MHL	No
Wi-Fi	802.11 b/g/n 2.4GHz
Wi-Fi Direct	Yes
DLNA Support	No
Bluetooth Version	Bluetooth v4.0
NFC	No
Bluetooth Profiles	A2DP, AVRCP, DI, HFP, HID, HOGP, HSP, MAP, OPP, PAN, PBAP
PC Sync.	Kies
General Information	
Color	Blue, Black, White
Form Factor	Touchscreen Bar
Physical specification	
Dimension (HxWxD, mm)	129 x 68.2 x 8.9
Weight (g)	122
Battery	
Internet Usage Time(3G) (Hours)	Up to 9
Internet Usage Time(Wi-Fi) (Hours)	Up to 12
Video Playback Time (Hours)	Up to 9
Standard Battery Capacity (mAh)	1850
Removable	Yes
Audio Playback Time (Hours)	Up to 40
Talk Time (3G WCDMA) (Hours)	Up to 10

Audio and Video	
Video Playing Format	MP4, M4V, 3GP, 3G2, MKV, WEBM
Video Playing Resolution	HD (1280 x 720)@30fps
Audio Playing Format	MP3, M4A, 3GA, AAC, OGG, OGA, WAV, AMR, AWB, FLAC, MID, MIDI, XMF, MXMF, IMY, RTTTL, RTX, OTA
Other	
S-Voice	No
Mobile TV	No
OS	Android
Sensors	Accelerometer, Proximity Sensor