

BACHELOR THESIS  
COMPUTER SCIENCE



RADBOD UNIVERSITY

---

# IRMA over Bluetooth

---

*Author:*  
Dion Scheper  
s4437578

*First supervisor/assessor:*  
dr. ir., B. Jacobs  
bart@cs.ru.nl

*Second assessor:*  
dr., S. Ringers  
sringers@cs.u.nl

June 25, 2018

*[This page intentionally left blank]*

### **Abstract**

The communication channel that IRMA currently supports is the internet. We will show that Bluetooth is a viable alternative in certain use cases that partially solves the problem of identifiable IP addresses, can bring peer to peer communication to the smartphone app, and encourages users to come up with their own use cases.

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Research question . . . . .	4
<b>2</b>	<b>Preliminaries</b>	<b>5</b>
2.1	Identification and Authentication . . . . .	5
2.2	The Proving Game . . . . .	6
2.2.1	Actors . . . . .	6
2.2.2	The Attacker . . . . .	6
2.3	Zero-Knowledge Proof . . . . .	6
2.4	IRMA . . . . .	7
2.4.1	The Players . . . . .	7
2.4.2	The protocol . . . . .	7
2.4.3	The architecture . . . . .	8
2.4.4	QR . . . . .	9
2.5	Bluetooth . . . . .	9
2.5.1	SDP . . . . .	9
2.5.2	Quality of Service . . . . .	10
2.5.3	Pairing and Bonding . . . . .	10
2.5.4	Association models . . . . .	10
<b>3</b>	<b>Research</b>	<b>11</b>
3.1	Applications of Bluetooth . . . . .	11
3.1.1	One-One . . . . .	11
3.1.2	Many-One . . . . .	12
3.2	Common transport . . . . .	13
3.2.1	Seralization . . . . .	13
3.2.2	Confidentiality . . . . .	14
3.2.3	Bluetooth bonding . . . . .	14
3.3	Security and Privacy . . . . .	14
3.3.1	Integrity . . . . .	14
3.3.2	Confidentiality . . . . .	15
3.3.3	Authentication . . . . .	15
3.3.4	Denial of Service Attack . . . . .	15

3.3.5 Privacy . . . . .	15
<b>4 Related Work</b>	<b>16</b>
<b>5 Conclusions</b>	<b>17</b>
5.1 Research Answer . . . . .	17
<b>A Appendix</b>	<b>21</b>
A.1 IrmaBluetoothTransportServer.java . . . . .	21
A.2 IrmaBluetoothTransportClient.java . . . . .	24
A.3 IrmaBluetoothTransportCommon.java . . . . .	27

# Chapter 1

## Introduction

The subject of privacy has become more prevalent [2][3][4]. Every once in a while privacy violations become news because of a misbehaving company [1]. European politicians have introduced a new law governing the gathering and processing of personal data [13]. This is called the General Data Protection Regulation and is further referenced as GDPR. This law forces organizations to rethink the way they handle personal data.

It remains to be seen how companies incorporate the new regulation. Organizations will search for technical and organizational measures that can help them. This is where IRMA comes in; an attribute based authentication technology that can reside on your smartphone as a possible solution.

IRMA has evolved over the years. It has migrated from a smart card to the smartphone. The documentation and IRMA software have matured. And the smartphone app is now cross platform.

A feature that IRMA does not support is a peer to peer session. This is because it uses the internet for all communication. One of the problems with the internet is that the number of IP addresses do not scale. A short-term solution to this problem is Network Address Translation [5]. This is a widely accepted solution and is implemented in consumer routers. IRMA users can not be reached over the internet without some configuration in their routers due to this NAT.

There are some technologies that could provide the peer-to-peer functionality. We will concentrate on Bluetooth. Bluetooth is included on every smartphone made today. It is a device-to-device communication standard. Let us illustrate the potential of this feature with an example:

The IRMA group supports infrastructure to load attributes on to your smartphone, including your age limits. If the owner of a bar wants to check your age, he could use IRMA over Bluetooth. With only minimal effort; downloading the app from the app store. This contrasts the current situation where he would be required to have a web server, and install IRMA functionality on that server.

## 1.1 Research question

How can Bluetooth support IRMA as a transport channel? We consider the following subquestions in the analysis:

- What advantages or disadvantages does Bluetooth have over the internet as an IRMA transport channel?
- What are the IRMA scenarios for Bluetooth instead of internet?
- How can Bluetooth features be technically realised in the IRMA app?

The objective of this paper is to show that Bluetooth can provide a transport channel for IRMA. And argue that this feature could potentially help with the roll out. The main contribution of this paper is a design to add Bluetooth functionality to IRMA in Java code.

## Chapter 2

# Preliminaries

We will first start with some preliminaries on the inner workings of IRMA and Bluetooth. The design choices made in section three will be more clear when backed up with the background presented here.

### 2.1 Identification and Authentication

Identification is the process of presenting some pseudonym that can be traced back to you as a person by some party. This could be your face, physically, but also an identifying number like your student number.

There is also non identifying information. This can not be traced back to you as a person when observed. For example eye color, age, city you live in. It is interesting that when multiple non identifying sources of information are combined they could be identifying.

The set of pseudonyms and other (non) identifying information can be classified into attributes. Proving the fact to you really are owner of an attribute is called authentication. There are typically three ways to authenticate:

- something you have  
For example your IRMA smartphone app or smart card.
- something you are  
For example your fingerprint or eyes.
- something you know  
For example a password or pin.

All of these ways of authentication have strengths and flaws. Passwords are hard to remember, you could lose your smartphone, and your fingerprint is irreplaceable for example. We will not discuss these problems or advantages here as this is out of scope for this thesis.



IRMA uses a combination of something you have and something you know. There are the attributes on your phone and there is the pin code in your head that you need in order to participate in an IRMA session.

## 2.2 The Proving Game

Throughout this thesis we will speak of the proving game. We can model the process of authentication as a role playing game. After the game is played it is evaluated if the authentication succeeded or not. We can also change the rules of the game to accommodate other requirements like privacy.

### 2.2.1 Actors

The actors in our game are the 'prover', Patrick, and 'verifier', Veronica. Patrick needs to prove some part of his digital identity. And Veronica needs to be convinced of some part of Patrick's identity. If the prover and verifier are playing according to the rules of the game, then the game is won when Patrick can convince Veronica of a claim. And he must not be able to convince Veronica of some invalid claim.

There are some relations between the verifier and prover. These only become apparent when one looks at physical context.

1. Veronica *wants* to be convinced by Patrick and wants the protocol to be such that a cheating Patrick can not convince her of something that is not true.
2. Patrick has contextual information that he should *trust* Veronica. He agrees with the disclosure of some part of his digital identity.
3. Patrick receives *something of value* and might search for ways to cheat the authentication to receive that value.

### 2.2.2 The Attacker

The attacker is the infamous Eve. Eve has been known to listen in on the communication between Alice and Bob in security analysis, also from Rivest et al [18]. Now she also listens in on the communication between Patrick and Veronica who are participating in a proving game. She tries to undermine the game. And can work with or against any of the two participants. She is the woman in the middle.

## 2.3 Zero-Knowledge Proof

The goal of the game is for Veronica to be convinced of some part of Patrick's identity. The way this is implemented in IRMA is using Zero-Knowledge

proofs and digital signatures. You prove that you own the signature of an attribute 'age>18' by giving a proof with which Veronica can verify that you indeed have a valid signature for the given attributes but she can learn nothing else from the proof.

## 2.4 IRMA

IRMA, I Reveal My Attributes, provides authentication without the explicit need to identify the prover. It has been based on idemix, previous work done by researchers at IBM Zurich [9][10].

An IRMA user can receive digital attributes like student number, eye colour, or age. Digital IRMA attributes may represent a physical trait like eye colour or an artificial one like student number. Attributes are bundled in credentials. And a credential can be disclosed, partially, using a verification protocol.

### 2.4.1 The Players

- Issuer  
This party can provide credentials based on an authentication scheme which is domain dependant. E.g. this can be that you verify your name by providing your passport.
- Verifier  
The party which verifies some attributes for some specific purpose. E.g. Veronica wants to know for sure the name of Patrick. Then Veronica is the verifier.
- Client  
The user that discloses a partial identity. E.g. Patrick.

### 2.4.2 The protocol

This is a general description of the IRMA protocol. This applies regardless of the communication channel used. First Patrick gets his credentials from Ivy, the issuer:

1. Patrick wants some credential from Ivy.
2. Ivy provides a way for Patrick to receive these credentials but she will need an other form of authentication from Patrick. That form could be his passport for example. This is necessary because Ivy is going to vouch for the credentials she is going to give to Patrick, and Patrick can reuse those after he has gotten them.
3. Patrick receives the credentials from Ivy.

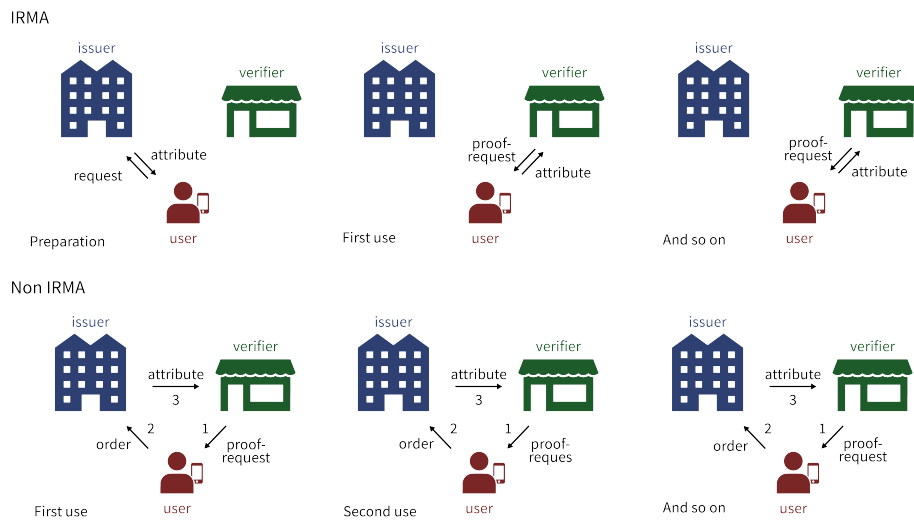


Figure 2.1: IRMA decentralised architecture as displayed on <https://privacybydesign.foundation/irma-explanation/>

Now that Patrick has his credentials ready he can prove any of these to any verifier. In our example we will generalize this to Veronica and look at a high level description of the protocol.

1. Veronica lets Patrick know where to connect to.
2. Patrick connects to Veronica
3. Patrick asks for the session, this includes a set of attributes Patrick has to reveal.
4. Patrick examines this set of attributes. If he agrees with the disclosure he accepts.
5. Patrick now calculates the proof based on the credentials he has and the parameters provided by Veronica.
6. He sends the proof to Veronica
7. Veronica verifies the proof and accepts if it is valid.

### 2.4.3 The architecture

The architecture of IRMA is decentralized. As can be seen in Figure 2.1. Important here is that after Patrick receives the credentials, they only exist on the device where Patrick receives them.

The products of the IRMA project at this moment include an API server with issuer and verification possibilities. And recently a mobile app was released, iOS and Android, that can function as a wallet of credentials.

The project has separated the concerns over multiple projects which are all available on github<sup>1</sup>.

- *irma\_mobile* this is the multi platform mobile application for Android/iOS.
- *gabi* Idemix in Go.
- *irmago* client implementation in Go.
- *irma\_api\_common* client implementation in Java.
- *irma\_api\_server* the IRMA API server implementation.
- *irma\_js* JavaScript client for in the browser.

#### 2.4.4 QR

IRMA uses Quick Response codes, QR-codes, to communicate connection information. This is an optical encoding method that encodes some text, often an URL, into a two dimensional image. This image can be decoded by the IRMA app using the devices' built in camera.

### 2.5 Bluetooth

Bluetooth specifies a device-to-device communications standard. The standard allows supported devices to connect to each other and share data. It is supported by billions of devices<sup>2</sup>.

There are a few concepts in Bluetooth extracted from the specification that need to be explained here. These are SDP, pairing and bonding, and quality of service.

#### 2.5.1 SDP

Bluetooth can use a process called Service Discovery Protocol to establish a transport channel. Veronica will start by broadcasting an identifier of the service that she is using, together with a user friendly name. Patrick is looking for Veronica's Bluetooth broadcast announcement. He will, after he has found her signal, communicate with her and establish a transport channel that is free at that time. It is not necessary to use SDP to establish a channel if the MAC address of the other party is known in advance.

---

<sup>1</sup><https://github.com/credentials/>

<sup>2</sup><https://www.bluetooth.com/bluetooth-technology>

### 2.5.2 Quality of Service

Bluetooth guarantees that the packets it sends are received in order and reliably [7]. If used with the 'secure channel' mode of operation, it also provides mutual authentication, message integrity and confidentiality of the connection. It also makes those claims for subsequent Bluetooth sessions between two bonded devices. This has been attacked in the past [14]. The Bluetooth channel has an effective range of 20-100 meters. This depends on the environment in which the two devices are located.

### 2.5.3 Pairing and Bonding

Bonding is the exchange of two long term cryptographic keys, providing confidentiality and mutual authentication. Two devices are authenticated before bonding by means of a pincode. Pairing is the process of establishing a session key.

One of the features of Bluetooth is that it is session-less. It is designed to bond with a device for repeated use. A Bluetooth device automatically connects to devices to which it was previously bonded. The assumption is that the device can be trusted because there was trust earlier by bonding. This contrasts with IRMA where the trust is not in the other device but in the IRMA protocol.

### 2.5.4 Association models

Devices with bluetooth can associate in multiple ways [7]. These are named Just Works, Numeric Comparison, Out of Band, and Passkey Entry. For this thesis we will look at Numeric Comparison and Just Works.

- Just Works is used without visual confirmation. There is no pincode or QR. This protects against passive eavedroppers but not against active MitM attacks.
- Numeric Comparison shows on both devices a numeric code and the users are asked to compare those. If they are equal then the connection succeeds. This protects against MitM attacks as well.

## Chapter 3

# Research

Data in this article has been gathered by performing a design and create cycle. The following software and corresponding documentation has been used to implement Bluetooth functionality in the app:

- Android Studio 3.0 (Android API  $\geq 23$ )
- Java 8
- Bluetooth v4.2

### 3.1 Applications of Bluetooth

There are two different applications in which Bluetooth can play a role and enhance IRMA. I will technically describe both in the following subsections and go into more detail about the consequences in the conclusions.

#### 3.1.1 One-One

The one-one approach described in this subsection is the one implemented in the appendix. Patrick and Veronica interact with each other in a peer to peer fashion. Veronica will let the prover know where to connect to. The URL to connect to is embedded in a QR code. This QR is also used in IRMA sessions over the internet. This means that the experience is familiar to Patrick. The steps are as follows:

1. Veronica prepares the attributes she wants Patrick to reveal.
2. Veronica's IRMA app prepares a QR code that corresponds to a standardized structure.
3. Patrick selects the option to scan a QR code in the IRMA app.

4. Patrick's device will now process the QR, connect to Veronica, and request the IRMA session information.
5. Veronica's device will send the prepared IRMA session request.
6. Patrick's device will process this according to the IRMA protocol specifications.
7. Veronica can read the information disclosed by Patrick and use that for the goal she had specified.

The string that is being represented by the initial QR should have the following structure. It should start with an IRMA bluetooth identifier so that the client can assume it is an IRMA bluetooth connection and not an internet connection. After which it should provide the connection parameters. These should at least include where to connect to and how to connect to it secure. The following structure is implemented in the PoC:

```
irma-bluetooth://MAC_ADDRESS/KEY_BASE64_ENCODED
```

The MAC address is the bluetooth MAC address and is enough for two devices to establish a connection. A symmetric key is appended to provide confidentiality. Message integrity can be provided by means of a HMAC but was not included in this PoC. A better solution is to let Veronica authenticate as is discussed in the section on security.

Some notes on the strengths and weaknesses of this approach are as follows. Patrick gets a visual hint of the device he is connecting to. This is due to the QR code being physically visible on the device. And the QR code disappears on the screen of Veronica when the devices are connected. We can provide perfect forward secrecy since the key lives for a short period of time and is forgotten by both devices after the session is complete.

The user interface is in accordance to the 7th law of identity as specified by Cameron [11]. The experience for Patrick is consistent regardless of connecting by Bluetooth or internet.

### 3.1.2 Many-One

Veronica will setup a server and Patrick will interact with it as a client. The native Bluetooth interface is used to connect the two devices. In this situation the server of Veronica is running and accepting any request for a specified duration. The steps that parties have to take are as follows:

1. Veronica prepares the attributes she wants to be revealed by the 'Patrick's' that are going to connect.
2. Patrick opens the IRMA app and selects Veronica her access point based on a familiar name.

3. Patrick's device will now use the native key exchange of Bluetooth to establish a connection with Veronica. This can use either the Just Works or the Numeric Comparison association method.
4. The devices will now handle the request in accordance to the IRMA protocol.
5. Veronica her server application will have a way to disseminate the results and act upon them.

This approach requires extra infrastructure from Veronica in the form of a IRMA supporting server application. Veronica might also want to save or process the responses she got on her server. For example when registering presence of a group of students. Patrick also has to adjust and select an IRMA access point from a list based on a human friendly identifier. This identifier may or may not be authenticated. If using 'Just Works' to associate the other device is not authenticated. It is necessary to implement Veronica's authentication in some other way. This contrasts the previous subsection where Patrick got visual confirmation of the device he was connecting to. This time he does not have that same guarantee. A solution to this problem is discussed in the conclusion section.

An advantage of the 'Just Works' method is that you can do it asynchronously without any action of Veronica after setup. In the 'Numeric Comparison' method you still need to compare the two numbers on both devices. This may or may not be preferable.

## 3.2 Common transport

Now we have a high level understanding of how Bluetooth might be used in the interaction between Veronica and Patrick. We continue by digging a little deeper into the Android Java code and look at how we provide serialization and confidentiality on that layer. This poses a few challenges for the programmer working on assembling packets and maintaining confidentiality. Another thing the programmer has to work around is the 'bonding' of Bluetooth devices by default.

### 3.2.1 Seralization

The Android Bluetooth API provides access to two byte streams; the receiving and sending stream.

```
1 InputStream getInputStream ()
2 OutputStream getOutputStream ()
```

Patrick will initiate the IRMA session and request the IRMA session information from Veronica. This session is represented by the Java class



`JwtSessionRequest`. The handling of these Java objects is done by a Bluetooth abstraction layer. This allows IRMA programmers to use intuitive code, e.g.: `public boolean write(DisclosureProofRequest dpr)`. Which sends *dpr* over the connection. This also applies to packets received. Patrick might expect something of type `DisclosureProofRequest.class` based on the request he made. When receiving the raw data the abstraction layer returns an `Object.class` and it is to the transport receiver to cast this object to the right class. When the casting fails, the connection fails and assumes the integrity of either the message or the protocol has been breached. It is possible to add integrity protection on top of this so that an invalid proof and integrity breach can be distinguished.

### 3.2.2 Confidentiality

We have implemented AES-CBC to provide confidentiality on this transport layer. That means that all packets going in or out are encrypted by this key. This key lives as long as the session is running and is reset to zero immediately after the connection is broken. This provides perfect forward secrecy.

### 3.2.3 Bluetooth bonding

Bluetooth bonds devices by default. This allows Patrick and Veronica to connect to each other more easily or even automatically. This is useful where Bluetooth is used for car kits are wireless keyboards. Not in the case of IRMA. The notion of a session is missing here. Therefore it is recommended for usability to remove the bond using Java Reflection.

## 3.3 Security and Privacy

This is a security assessment based on the security principles of: integrity, confidentiality, and authentication. The DoS attack is considered afterwards. And we conclude with an observation on privacy.

### 3.3.1 Integrity

The packets exchanged between Patrick and Veronica are not checksummed and neither do they have a message authentication code. These constructions are normally used to provide the guarantee of integrity. The shared key between Patrick and Veronica is secret and has a short lifetime of five to twenty seconds. The integrity therefore has to be attacked by an active attacker that does not have the encryption key. Such an attacker can try to adjust the packet to something malicious. The proof of concept in the appendix assumes that an active attacker cannot efficiently generate a stream

of bytes that would lead to a chosen Java object after decryption and deserialisation. In which case any attacks on integrity would be noticed and the result would be DoS. It is recommended to add an additional method to protect integrity for example by including an HMAC in production environment.

### **3.3.2 Confidentiality**

Confidentiality is provided on the assumption that Eve can not obtain the key. This is practically enforced by Veronica by shielding the visual representation of the key, QR, against Eve in the one-one scenario. The key exchange in the many-one is based on Bluetooth security guarantees.

### **3.3.3 Authentication**

The devices are not authenticated to favor usability over security. To authenticate both parties we use the following steps in the one-one scenario: the QR presented by Veronica contains a key that is used in one session. The only one being able to commit in to the session is Patrick. This can however be done by letting Veronica disclose attributes that allow identification in that specific context before the authentication of Patrick. If the device of Veronica supports a screen it is also possible to allow device authentication by means of a pincode that is verified on both devices. Thus Patrick is authenticated by the IRMA protocol to Veronica but the devices that they use are not authenticated to each other.

### **3.3.4 Denial of Service Attack**

Eve can undermine the bluetooth channel by probing Veronica. That way Patrick can not use the channel. She has to have a device in close proximity for this to succeed. Notice that Eve is not able to Hijack the communications with Patrick in the one-one scenario since Patrick will work with a cryptographic key communicated through the QR code. The same is not true for many-one scenario's. If used with the 'Just Works' association model then the connection can be hijacked during setup of the connection with an active MitM attack.

### **3.3.5 Privacy**

The privacy of Patrick is the main concern in IRMA. And one of the problems acknowledged is that IP addresses can be seen as pseudonyms [6]. You do not necessarily have this problem with a Bluetooth connection. The bluetooth group is working on randomization of the MAC addresses so that you no longer have the problem of abusing it as a pseudonym.

## Chapter 4

# Related Work

This idea of modeling security requirements as a role-play was coined by Rivest et al and is used here [18]. Other researchers quickly took over this tradition instead of the more mathematical notation.

For an elaborate example on zero-knowledge we would like to refer to the 'How to explain Zero-Knowledge to your children' paper [17]. The original scientific paper on zero-knowledge theory is there for further reading on the underlying math [15]. These cryptographic ideas are the ones on which IRMA bases its security.

Current authentication methods with web servers mostly use password based authentication. A possible reason is nicely illustrated in a survey from 2012[8]. Widespread use of the password has not been eradicated yet because of good reasons. Though researchers all over the world have tried to set up alternative authentication mechanisms. An example of that is Idemix[10] which uses attribute based authentication on which IRMA is based.

IRMA also considers the physical context in which it plays be it a web server, a garbage dumpster, or a supermarket. The initial work of IRMA focused on smart cards [19]. This work though has halted and the focus has shifted to smartphones [6].

The evolution of Bluetooth shows a strong focus on security. A good survey on possible Bluetooth threats is from 2010 [12]. Since then it has improved by incorporating cryptographic tools. Interestingly the technique developed here using QR codes as session establishment has been done before [16].

## Chapter 5

# Conclusions

Bluetooth is a viable communications channel for IRMA to use besides the internet. We will answer the research questions in the first section that support this claim.

### 5.1 Research Answer

- What advantages does Bluetooth have over the internet?
  1. Bluetooth allows peer to peer sessions, this makes it easier for end users to start using IRMA.
  2. Bluetooth encourages decentralization of IRMA verifier responsibilities. Where in the future commercial parties may try to reveal information about identities and exploit the 'verification as a service', Bluetooth brings the IRMA attribute verification alive on every end users smartphone and potentially end points in contextual authentication situations.
  3. Bluetooth solves the problem of large scale tracking because the session has a small perceivable physical range. This opposed to the internet where a web server gets requests from all over the world. It is still technically possible to track but as IRMA uses multi-show unlinkability it 'discourages' such an act.
- What disadvantages does Bluetooth have?
  1. For now it comes with identifiable MAC address. This problem is being addressed by Bluetooth group with random MAC addresses. This is a serious drawback as this MAC address is device identifying, which is more personal than an IP address. The latter is identifying a network.
  2. Internet is still necessary due to the split-key solution. It would be great to use IRMA without internet, but because part of the

unlock key in the app is on the IRMA server; you will need to connect to it.

- What are the IRMA scenario's for Bluetooth instead of internet?
  - It would be situations where there is a physical place to go to. For example; public transport, restaurants, bars, cars, theater, etc.
  - On the other hand does it encourage end users to replace normal authentication methods by IRMA. For example if you have a camping, then you can ask people to provide them with some attributes instead of a copy of their passport.

# Bibliography

- [1] <https://nos.nl/artikel/2197753-autoriteit-persoonsgegevens-windows-10-schendt-privacy.html>.
- [2] <https://nos.nl/artikel/2198712-meer-privacy-op-internet-stap-dichterbij.html>.
- [3] <https://nos.nl/artikel/2199500-datahandelaren-schenden-privacy-van-miljoenen-nederlanders.html>.
- [4] <https://nos.nl/artikel/2207112-prijs-grootste-privacyschending-naar-kabinet-wegens-aftapwet.html>.
- [5] <https://tools.ietf.org/html/rfc1631>.
- [6] 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.
- [7] SIG Bluetooth. Bluetooth 4.2 core specification. *Bluetooth SIG*, 2009.
- [8] Joseph Bonneau, Cormac Herley, Paul C Van Oorschot, and Frank Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 553–567. IEEE, 2012.
- [9] Jan Camenisch and A Lysyanskaya. Efficient non-transferable anonymous multi-show credential system with optional anonymity revocation. 01 2001.
- [10] 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.
- [11] K Cameron. The laws of identity. 2005. *Microsoft Corporation*, 2009.
- [12] J. Dunning. Taming the blue beast: A survey of bluetooth based threats. *IEEE Security Privacy*, 8(2):20–27, March 2010.

- [13] Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *Official Journal of the European Union*, L119:1–88, May 2016.
- [14] Chia-Ming Fan, Shiuhpyng Shieh, and Bing-Han Li. On the security of password-based pairing protocol in bluetooth. In *Network Operations and Management Symposium (APNOMS), 2011 13th Asia-Pacific*, pages 1–4. IEEE, 2011.
- [15] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208, 1989.
- [16] Artur Hłobaż, Krzysztof Podlaski, and Piotr Milczarski. Applications of qr codes in secure mobile data exchange. In *International Conference on Computer Networks*, pages 277–286. Springer, 2014.
- [17] Jean-Jacques Quisquater, Myriam Quisquater, Muriel Quisquater, Michaël Quisquater, Louis Guillou, Marie Annick Guillou, Gaïd Guillou, Anna Guillou, Gwenolé Guillou, and Soazig Guillou. How to explain zero-knowledge protocols to your children. In *Conference on the Theory and Application of Cryptology*, pages 628–631. Springer, 1989.
- [18] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
- [19] 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.

## Appendix A

# Appendix

The proof of concept on which this thesis is based is provided here for reference and to be scrutinized. It is in the open source domain.

### A.1 IrmaBluetoothTransportServer.java

```
1 package org.irmacard.cardemu.bluetooth;
2
3 import android.bluetooth.BluetoothAdapter;
4 import android.bluetooth.BluetoothServerSocket;
5 import android.bluetooth.BluetoothSocket;
6 import android.os.Handler;
7 import android.os.Message;
8 import android.support.annotation.NonNull;
9 import android.util.Log;
10
11 import com.google.gson.Gson;
12 import com.google.gson.reflect.TypeToken;
13
14 import org.irmacard.api.common.JwtSessionRequest;
15 import org.irmacard.api.common.disclosure.DisclosureProofRequest;
16 import org.irmacard.api.common.disclosure.DisclosureProofResult;
17 import org.irmacard.api.common.util.GsonUtil;
18 import org.irmacard.credentials.idemix.proofs.ProofD;
19 import org.irmacard.credentials.idemix.proofs.ProofList;
20 import org.irmacard.credentials.info.AttributeIdentifier;
21 import org.irmacard.credentials.info.InfoException;
22 import org.irmacard.credentials.info.KeyException;
23
24 import java.io.IOException;
25 import java.io.InputStream;
26 import java.io.OutputStream;
27 import java.net.URL;
28 import java.util.ArrayList;
```



```

29 import java.util.Date;
30 import java.util.List;
31
32 import javax.crypto.SecretKey;
33
34 import io.jsonwebtoken.JwtBuilder;
35 import io.jsonwebtoken.Jwts;
36
37 /**
38  * Created by neonlight on 24-11-17.
39  */
40
41 public class IrmaBluetoothTransportServer extends Handler
    implements Runnable{
42     private static final int CONNECTION_WAIT = 20000;        //
43         Milliseconds to wait for the client to connect
44     private static final int CONNECTION_TIMEOUT = 20000;    //
45         Milliseconds after being connected; to drop the connection.
46     private SecretKey key;
47     private IrmaBluetoothHandler handler;
48     private static IrmaBluetoothTransportServer instance;
49     private IrmaBluetoothTransportCommon common;
50
51     @Override
52     public void handleMessage(Message msg) {
53         super.handleMessage(msg);
54         IrmaBluetoothHandler.State[] values =
55             IrmaBluetoothHandler.State.values();
56         handler.publish(values[msg.what]);
57     }
58
59     private IrmaBluetoothTransportServer(SecretKey key,
60         IrmaBluetoothHandler handler) {
61         this.key = key;
62         this.handler = handler;
63     }
64
65     public static void start(@NonNull SecretKey key, @NonNull
66         IrmaBluetoothHandler handler) {
67         if(instance == null) {
68             instance = new IrmaBluetoothTransportServer(key,
69                 handler);
70             new Thread(instance).start();
71         }
72     }
73
74     @Override
75     public void run() {
76         android.os.Process.setThreadPriority(android.os.Process.THREAD_PRIORITY_BACKGROUND);
77     }
78 }

```

```

71     receiveConnection();
72 }
73
74 private void receiveConnection() {
75     // Create the listening server socket
76     BluetoothServerSocket serverSocket;
77     BluetoothSocket socket;
78     try {
79         serverSocket = BluetoothAdapter.getDefaultAdapter()
80             .listenUsingInsecureRfcommWithServiceRecord("Irma",
81                 IrmaBluetoothTransportCommon.IRMA_UUID);
82     } catch (IOException e) {
83         Log.e("TAG", "ServerSocket failed", e);
84         return;
85     }
86
87     // Serve any client connecting.
88     try {
89         Log.d("TAG", "Accepting connection");
90         socket = serverSocket.accept(CONNECTION_WAIT);
91         common = new IrmaBluetoothTransportCommon(key, socket);
92
93         sendEmptyMessage(IrmaBluetoothHandler.State.CONNECTED.ordinal());
94
95         long timeout = System.currentTimeMillis() +
96             CONNECTION_TIMEOUT;
97
98         DisclosureProofRequest disclosureProofRequest =
99             handler.getDisclosureProofRequest();
100         // TODO: Refactor the transportserver/handler? Does it
101         // need to verify?
102
103         boolean done = false;
104         while(common.connected() && System.currentTimeMillis() <
105             timeout && !done) {
106             Object obj = common.read();
107
108             if(obj instanceof ProofList) {
109                 ProofList proofList = (ProofList) obj; done =
110                     true;
111                 try {
112                     proofList.populatePublicKeyArray();
113                     DisclosureProofResult result =
114                         disclosureProofRequest.verify(proofList);
115                     if(result.getStatus() ==
116                         DisclosureProofResult.Status.VALID) {
117                         sendEmptyMessage(IrmaBluetoothHandler.State.SUCCESS.ordinal());
118                     } else {
119                         sendEmptyMessage(IrmaBluetoothHandler.State.FAIL.ordinal());
120                     }
121                 }

```

```

112         }
113         common.write(result.getStatus());
114     } catch (InfoException | KeyException |
115             RuntimeException e) {
116         Log.e("TAG", "Proof exception", e);
117         sendEmptyMessage(IrmaBluetoothHandler.State.FAIL.ordinal());
118         common.write(DisclosureProofResult.Status.INVALID);
119     }
120     } else if(obj instanceof RequestJwtSession) {
121         common.write(disclosureProofRequest);
122     } else {
123         Log.d("TAG", "Unrecognized object: " + obj);
124     }
125 } catch (IOException e) {
126     Log.e("TAG", "Connection failed.", e);
127     sendEmptyMessage(2);
128 }
129
130 instance = null; // This server is a Singleton.
131 common.close();
132 try {
133     serverSocket.close();
134 } catch (IOException e) {
135     Log.e("TAG", "Server close failed", e);
136 }
137 }
138 }

```

## A.2 IrmaBluetoothTransportClient.java

```

1 package org.irmacard.cardemu.irmaclient;
2
3 import android.bluetooth.BluetoothAdapter;
4 import android.bluetooth.BluetoothDevice;
5 import android.bluetooth.BluetoothSocket;
6 import android.util.Log;
7
8 import com.google.gson.Gson;
9 import com.google.gson.internal.Primitives;
10
11 import org.irmacard.api.common.JwtSessionRequest;
12 import org.irmacard.api.common.disclosure.DisclosureProofRequest;
13 import org.irmacard.api.common.disclosure.DisclosureProofResult;
14 import org.irmacard.api.common.util.GsonUtil;
15 import org.irmacard.cardemu.bluetooth.IrmaBluetoothTransportCommon;
16 import org.irmacard.cardemu.httpclient.HttpClientException;
17 import org.irmacard.cardemu.httpclient.HttpResultHandler;

```

```

18
19 import java.io.IOException;
20 import java.io.InputStream;
21 import java.io.OutputStream;
22 import java.lang.reflect.Type;
23 import java.math.BigInteger;
24
25 import javax.crypto.SecretKey;
26
27 /**
28  * Created by neonlight on 20-11-17.
29  */
30
31 public class IrmaBluetoothTransportClient implements IrmaTransport {
32     private BluetoothDevice device;
33     private BluetoothSocket socket;
34     private IrmaBluetoothTransportCommon common;
35     private Gson gson;
36
37     public IrmaBluetoothTransportClient(SecretKey key, String mac) {
38         Log.d("TAG", "IrmaBluetoothTransportClient - Prover");
39         this.gson = GsonUtil.getGson();
40         this.device =
41             BluetoothAdapter.getDefaultAdapter().getRemoteDevice(mac);
42         if(connect()) {
43             this.common = new IrmaBluetoothTransportCommon(key,
44                 socket);
45         }
46     }
47
48     private boolean connect() {
49         // Create socket to device
50         Log.d("TAG", "Creating socket");
51         try {
52             socket =
53                 device.createInsecureRfcommSocketToServiceRecord(IrmaBluetoothTransportCommon
54                     Log.d("TAG", "Socket Created");
55             } catch (IOException e) {
56                 Log.e("TAG", "Socket creation failed", e);
57                 return false;
58             }
59
60         // Connect to the device
61         try {
62             socket.connect();
63             Log.d("TAG", "Socket Connected");
64             return true;
65         } catch (Exception e) {
66             Log.e("TAG", "Socket could not connect", e);

```

```

64         return false;
65     }
66 }
67
68 @Override
69 public <T> void post(Type type, String url, Object object,
70     HttpResultHandler<T> handler) {
71     Log.d("TAG", "POST::" + type + ":" + url
72         + ":" + object + ":" + handler);
73     try {
74         common.write(gson.toJson(object),
75             IrmaBluetoothTransportCommon.Type.POST_PROOFLIST);
76         T result = Primitives.wrap((Class<T>)
77             type).cast(common.read());
78         if(result != null) {
79             handler.onSuccess(result );
80         } else {
81             throw new IOException("Object is not correctly
82                 received.");
83         }
84     } catch (IOException e) {
85         handler.onError(new HttpClientException(0, "Bluetooth
86             Error"));
87         common.close();
88     }
89 }
90
91 @Override
92 public <T> void get(Type type, String url, HttpResultHandler<T>
93     handler) {
94     Log.d("TAG", "GET::" + type + ":" + url + ":" + handler);
95     try {
96         common.write("",
97             IrmaBluetoothTransportCommon.Type.GET_JWT);
98         T result = Primitives.wrap((Class<T>)
99             type).cast(common.read());
100         if(result != null) {
101             handler.onSuccess(result );
102         } else {
103             throw new IOException("Object is not correctly
104                 received.");
105         }
106     } catch (IOException e) {
107         handler.onError(new HttpClientException(0, "Bluetooth
108             Error"));
109         common.close();
110     }
111 }

```

```

102     @Override
103     public void delete() {
104         Log.d("TAG", "DELETE");
105     }
106 }

```

### A.3 IrmaBluetoothTransportCommon.java

```

1 package org.irmacard.cardemu.bluetooth;
2
3 import android.bluetooth.BluetoothDevice;
4 import android.bluetooth.BluetoothSocket;
5 import android.support.annotation.NonNull;
6 import android.util.Log;
7
8 import com.google.gson.Gson;
9 import com.google.gson.reflect.TypeToken;
10
11 import org.irmacard.api.common.JwtSessionRequest;
12 import org.irmacard.api.common.disclosure.DisclosureProofRequest;
13 import org.irmacard.api.common.disclosure.DisclosureProofResult;
14 import org.irmacard.api.common.util.GsonUtil;
15 import org.irmacard.credentials.idemix.proofs.ProofD;
16 import org.irmacard.credentials.idemix.proofs.ProofList;
17
18 import java.io.IOException;
19 import java.io.InputStream;
20 import java.io.OutputStream;
21 import java.lang.reflect.Method;
22 import java.nio.BufferOverflowException;
23 import java.security.NoSuchAlgorithmException;
24 import java.util.ArrayList;
25 import java.util.Arrays;
26 import java.util.Date;
27 import java.util.List;
28 import java.util.UUID;
29 import java.util.concurrent.TimeoutException;
30
31 import javax.crypto.Cipher;
32 import javax.crypto.KeyGenerator;
33 import javax.crypto.SecretKey;
34 import javax.crypto.spec.IvParameterSpec;
35 import javax.crypto.spec.SecretKeySpec;
36
37 import io.jsonwebtoken.JwtBuilder;
38 import io.jsonwebtoken.Jwts;
39
40 /**

```

```

41  * Created by neonlight on 24-11-17.
42  */
43
44  public class IrmaBluetoothTransportCommon {
45      public static final UUID IRMA_UUID =
          UUID.fromString("c7986f0a-3154-4dc9-b19c-a5e713bb1737");
          //TODO: choose UUID (this is random generated)
46      private static final long TIMEOUT = 3000;           //
          network I/O timeout in milliseconds.
47      private static final int BUFFER_SIZE = 1024;        // nr
          bytes the buffer should hold
48      private static final int PACKET_SIZE = 4096;        //
          maximum size of the received object
49      private BluetoothSocket socket;
50      private InputStream is;
51      private OutputStream os;
52      private SecretKey key;
53      private Gson gson;
54
55      public enum Type {
56          POST_PROOFLIST,
57          PROOF_RESULT_STATUS,
58          GET_JWT,
59          PROOFREQUEST
60      }
61
62      public IrmaBluetoothTransportCommon(@NonNull SecretKey key,
          @NonNull BluetoothSocket socket) {
63          this.key = key;
64          this.socket = socket;
65          this.is = null;
66          this.os = null;
67          try {
68              this.is = socket.getInputStream();
69              this.os = socket.getOutputStream();
70          } catch (IOException | NullPointerException e) {
71              Log.e("TAG", "I/O could not be opened", e);
72          }
73          this.gson = GsonUtil.getGson();
74      }
75
76      public boolean connected() {
77          return this.socket.isConnected();
78      }
79
80      static public SecretKey generateSessionKey() {
81          SecretKey secretKey = null;
82          try {
83              KeyGenerator keyGen = KeyGenerator.getInstance("AES");

```

```

84         keyGen.init(128); // for example
85         secretKey = keyGen.generateKey();
86     } catch (NoSuchAlgorithmException e) {
87         Log.e("TAG", "NoSuchAlgorithm", e);
88     }
89     return secretKey;
90 }
91
92 public void setInputStreams(@NonNull InputStream is,
93                             @NonNull OutputStream os) {
94     this.is = is;
95     this.os = os;
96 }
97
98 public byte[] encrypt(byte[] bytes) {
99     try {
100         SecretKeySpec secretKey = new
101             SecretKeySpec(key.getEncoded(), "AES");
102         Cipher cipher =
103             Cipher.getInstance("AES/CBC/PKCS5Padding");
104         IvParameterSpec ivParams = new IvParameterSpec(new
105             byte[cipher.getBlockSize()]);
106         cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivParams);
107         return cipher.doFinal(bytes);
108     } catch (Exception e) {
109         // Reset the byte array to zero on failure
110         Log.e("TAG", "Encryption failed", e);
111         for(int i = 0; i < bytes.length; i++) {
112             bytes[i] = 0;
113         }
114         return bytes;
115     }
116 }
117
118 public byte[] decrypt(byte[] bytes) {
119     try {
120         SecretKeySpec secretKey = new
121             SecretKeySpec(key.getEncoded(), "AES");
122         Cipher cipher =
123             Cipher.getInstance("AES/CBC/PKCS5Padding");
124         IvParameterSpec ivParams = new IvParameterSpec(new
125             byte[cipher.getBlockSize()]);
126         cipher.init(Cipher.DECRYPT_MODE, secretKey, ivParams);
127         return cipher.doFinal(bytes);
128     } catch (Exception e) {
129         Log.e("TAG", "Decryption failed", e);
130         for(int i = 0; i < bytes.length; i++) {
131             bytes[i] = 0;
132         }
133     }
134 }

```



```

126         return bytes;
127     }
128 }
129
130 private boolean unbond(BluetoothDevice device) {
131     try {
132         Method m = device.getClass().getMethod("removeBond",
133             (Class[]) null);
134         m.invoke(device, (Object[]) null);
135     } catch (Exception e) {
136         return false;
137     }
138     return true;
139 }
140
141 /**
142  * Close the socket, and UNBOND it!
143  */
144 public void close() {
145     BluetoothDevice tmp = null;
146     try {
147         tmp = socket.getRemoteDevice();
148         socket.close();
149     } catch (IOException | NullPointerException e) {
150         Log.d("TAG", "Connection closed failed, socket already
151             closed?");
152     }
153     Log.d("TAG", "Closed");
154     if(unbond(tmp)) Log.d("TAG", "closing and unbonding
155         succesfull");
156 }
157
158 /**
159  * Keep Busy-Waiting on the stream for it to show up with data,
160  * then take that data.
161  * Return it to be processed.
162  * @param buffer the buffer in which to place the data.
163  * @return the number of bytes that have been read.
164  * @throws IOException in case the InputStream is unavailable,
165  *         or broken.
166  * @throws TimeoutException in case it took too long.
167  */
168 private int readPacket(byte[] buffer) throws IOException,
169     TimeoutException{
170     long time = System.currentTimeMillis() + TIMEOUT;
171     while(System.currentTimeMillis() < time && is.available() <
172         1) ;
173     if(is.available() > 0) {
174         return is.read(buffer);
175     }
176 }

```

```

168     }
169     throw new TimeoutException("The reading failed");
170 }
171
172 /**
173  * Keep reading packets until you can reconstruct it to an
174  * object.
175  * The first two bytes received contain the length of the data.
176  * @return the byte array containing the decrypted object
177  * @throws TimeoutException in case it took too long to read the
178  * object
179  * @throws IOException in case the channel is broken.
180 */
181 private byte[] readObject() throws TimeoutException,
182     IOException{
183     byte[] buffer = new byte[BUFFER_SIZE]; byte[] result = new
184         byte[PACKET_SIZE];
185     int nr_bytes = readPacket(buffer);
186     if (nr_bytes < 4) { return null; }
187
188     // Parse packet size field
189     final int payload_length = ((buffer[0] & 0xff) << 8) |
190         (buffer[1] & 0xff);
191     int payload_size = nr_bytes - 2;
192
193     // Start processing.
194     System.arraycopy(buffer, 2, result, 0, payload_size);
195     boolean done = payload_size == payload_length;
196
197     // Continue reading if packet is not complete.
198     while (!done) {
199         nr_bytes = readPacket(buffer);
200         if (payload_size + nr_bytes > PACKET_SIZE) {
201             throw new BufferOverflowException();
202         }
203
204         System.arraycopy(buffer, 0, result, payload_size + 2,
205             nr_bytes);
206         payload_size += nr_bytes;
207         done = payload_size == payload_length;
208     }
209
210     return decrypt(Arrays.copyOfRange(result, 0,
211         payload_length));
212 }
213
214 /**
215  * The IRMA object is to be returned:
216  * - DisclosureProofRequest

```

```

210     * - JwtSessionRequest
211     * - ProofList
212     * - DisclosureProofResult.Status
213     * etc. etc. etc.
214     * @return an object that could be any of the above class.
215     */
216     public Object read() {
217         try {
218             byte[] bytes = null;
219             while(bytes == null) {
220                 bytes = readObject();
221             }
222
223             Type irma_type = Type.values()[((bytes[0] & 0xff) << 8)
                | (bytes[1] & 0xff)];
224             byte[] result = Arrays.copyOfRange(bytes, 2,
                bytes.length);
225
226             switch(irma_type) {
227                 case POST_PROOFLIST:
228                     String jwt = new String(result);
229                     Log.d("TAG", "RECV: POST_PROOFLIST: " + jwt);
230                     List<ProofD> proofDS = gson.fromJson(jwt, new
                        TypeToken<ArrayList<ProofD>>(){}.getType());
231                     ProofList proofList = new ProofList();
232                     proofList.addAll(proofDS);
233                     return proofList;
234                 case PROOF_RESULT_STATUS:
235                     Log.d("TAG", "RECV: PROOF_RESULT_STATUS: " + new
                        String(result));
236                     return DisclosureProofResult.Status.valueOf(new
                        String(result));
237                 case GET_JWT:
238                     Log.d("TAG", "RECV: GET_JWT ");
239                     return new RequestJwtSession();
240                 case PROOFREQUEST:
241                     Log.d("TAG", "RECV: PROOFREQUEST: " + new
                        String(result));
242                     //TODO: build the JwtSessionRequest at server
                        side instead of replicating at client side
243                     DisclosureProofRequest request =
                        gson.fromJson(new String(result), new
                            TypeToken<DisclosureProofRequest>(){}.getType());
244                     JwtSessionRequest req = new JwtSessionRequest(
                        getDisclosureJwt(request),
245                         request.getNonce(),
246                         request.getContext()
247                     );
248                 );
249                 return req;

```

```

250         default:
251             Log.d("TAG", "RECV: UNKNOWN IRMA TYPE");
252             return null;
253     }
254 } catch (TimeoutException | IOException e) {
255     Log.e("TAG", "Timeout readObject socket", e);
256     return null;
257 } catch (Exception e) {
258     Log.e("TAG", "Unknown object mismatch?", e);
259     return null;
260 }
261 }
262
263 private String getDisclosureJwt(DisclosureProofRequest dpr) {
264     // Translate the object to bytes
265     String result = gson.toJson(dpr);
266
267     //TODO: fix hacky string manipulation
268     result = result.substring(1, result.length() - 1);
269     JwtBuilder builder = Jwts.builder();
270     String jwt = builder.setPayload(
271         "{ \"sub\": \"verification_request\", \" +
272           \"iss\": \"bluetooth\", \" +
273           \"iat\": \" + new Date().getTime() + \", \" +
274           \"sprequest\": { \" +
275             \"validity\": 60, \" +
276             \"request\": { \" +
277               result +
278             \"} \" +
279           \"} \" +
280           \"}\" ).compact();
281
282     return jwt;
283 }
284
285 /**
286  * Public functions to write the objects to the other end.
287  * As a developer you may want to transfer the object 'MyObject'
288  * 1. implement 'public boolean write(MyObject object)'
289  * 2. implement 'case ??' in the 'read' function above.
290  *
291  * @param dpr the DisclosureProofRequest
292  * @return it succeeded.
293  */
294 public boolean write(DisclosureProofRequest dpr) {
295     return write(gson.toJson(dpr), Type.PROOFREQUEST);
296 }
297
298 public boolean write(ProofList proofList) {

```

```

299         return write(gson.toJson(proofList), Type.POST_PROOFLIST);
300     }
301
302     public boolean write(DisclosureProofResult.Status status) {
303         return write(status.toString().getBytes(),
304             Type.PROOF_RESULT_STATUS);
305     }
306
307     public boolean write(String url, Type type) {
308         return write(url.getBytes(), type);
309     }
310
311     /**
312      * The bare level write, it writes the bytes, adds a 'size'
313      * header field, and serialises the irma type.
314      * @param object the bytes representing the object to send.
315      * @param irma_type the 'type' of the IRMA packet, indicates to
316      * the receiver what it is receiving.
317      * look into the enum Type at the top of this
318      * class declaration.
319      * @return it succeeded.
320      */
321     private boolean write(byte[] object, Type irma_type) {
322         // Build Enchilada to be encrypted
323         int irma_int = irma_type.ordinal();
324         byte[] enchilada = new byte[2 + object.length];
325         enchilada[0] = (byte) (irma_int >>> 8);
326         enchilada[1] = (byte) (irma_int);
327
328         // Set Payload
329         System.arraycopy(object, 0, enchilada, 2, object.length);
330
331         // Encrypt the enchilada
332         byte[] encrypted = encrypt(enchilada);
333
334         // Attach header
335         int length = encrypted.length;
336         byte[] packet = new byte[2 + length];
337         packet[0] = (byte) (length >>> 8);
338         packet[1] = (byte) (length);
339         System.arraycopy(encrypted, 0, packet, 2, length);
340
341         // Send it
342         Log.d("TAG", "SEND: " + new String(packet) + "(" +
343             packet.length + ")");
344         try {
345             this.os.write(packet);
346             return true;
347         }

```

```
343         } catch(IOException e) {
344             Log.e("TAG", "Write failed", e);
345             return false;
346         }
347     }
348 }
```