Bachelor's Thesis Computing Science

Signatures in IRMAseal

Dirk Doesburg
s1040211

January 15, 2023

Daily supervisor:
D. Ostkamp, MSc.

First assessor:
dr. B.E. van Gastel

Second assessor:
dr. H.K. Schraffenberger

Radboud University

Abstract

IRMAseal is an identity-based encryption service that uses a Trusted Third Party (TTP) to encrypt and decrypt messages, in which recipients can be defined by any combination of IRMA attributes. It does not yet provide authenticity, integrity, and non-repudiation. This thesis presents several approaches to provide these properties in IRMAseal. The security and privacy guarantees, ease of implementation, and some usability features of these approaches are determined and compared. The schemes are based on IRMA's attribute-based signatures, identity-based signatures, or a third party that creates standard signatures.

Our main findings are that (1) the schemes that are based on IRMA signatures do not require additional trusted parties outside of the IRMA ecosystem, whereas the others need third parties for signing or distributing signing keys, with some mechanism to distribute trust, and (2) schemes that give the IRMAseal client access to key material—allowing it to create multiple signatures later, without a separate IRMA session—come with security limitations (mainly for non-repudiation) directly because of the access to key material.

Contents

Chapter 1

Introduction

A fundamental problem of end-to-end encryption is that, while the technique has been available for decades, it is in many cases not adopted by the general public. Specifically, even though two well-known encryption standards—OpenPGP and S/MIME—have been available since the 1990s [8, 28], end-to-end encryption is rarely used for emails. For instance, a study on 81 million emails at a university [33] found that only 0.06% were encrypted.

While some modern messaging systems—such as WhatsApp and Signal—do use end-to-end encryption by default, they often still rely on users to verify the identity of their contacts, e.g. by comparing a security code in real life. In practice, no user does this, severely limiting the security offered by this end-to-end encryption.

The poor adoption of end-to-end encryption can be blamed on the lack of usability of the existing solutions. Several studies have shown that PGP is too hard to use for a majority of email users [33, 25], and even in the case of Signal, the vast majority of users fail to verify security codes [30]. In both cases, the underlying problem is that manual key management[1] is cumbersome and difficult.

A project that aims to tackle the problem of manual key management is IRMAseal[2], a system for identity-based or attribute-based encryption that eliminates the burden of manual key management for users. IRMAseal is an encryption service that uses a Trusted Third Party (TTP) to distribute keys for the decryption of messages. Instead of a public key of the recipient, the sender only requires the (static) master public key of the TTP, and an identity of the recipient. To decrypt, the recipient proves against a TTP that he/she owns this identity. The TTP then generates a 'user secret key' that the recipient can use to decrypt the message.

IRMAseal uses the privacy-friendly identity platform IRMA[3] to authenticate the recipient. This is an attribute-based credential system, that allows users to show that they have certain attributes: small pieces of information, such as `age > 18` or `first name`. IRMAseal uses a set of attributes as an identity, allowing users to address messages to for example people with `age > 18: true AND first name: "John"` or `is doctor at: "Radboud`

[1] Even in WhatsApp and Signal, which automatically handle keys, users still need to intervene by verifying codes manually to prevent man-in-the-middle attacks.

[2] `https://github.com/encryption4all/irmaseal`

[3] `https://irma.app/`

UMC", rather than just to a single person identified by email address.

IRMAseal can be used to encrypt emails, but also for any other kind of message: it is used in PostGuard[4], an email client add-on that encrypts and decrypts emails, as well as in Cryptify[5], a file-transfer service. However, IRMAseal currently only offers confidentiality of the messages, not authenticity or non-repudiation: IRMAseal does not offer signatures. Authenticity is a feature that is offered by other cryptography for which IRMAseal seeks to be an alternative (e.g. OpenPGP).

In this thesis, we aim to investigate how IRMAseal can be expanded to provide authenticity without the burden of key management for users. Offering authenticity is an important feature, which can for example be seen from the relative prevalence of signed emails compared to encryption: whereas Stransky et al. [33] found that 0.06% of emails were encrypted, 2.8% of them were signed. In the context of email, the importance of signatures also follows from the fact that spoofing (sending an email on behalf of someone else) is trivial, while eavesdropping is not [1].

In our investigation, we assume that there may be multiple TTPs: the system needn't be centralized (see subsection 2.3.5). We also do not assume the context of email; different kinds of messages should also be supported, as they currently are in IRMAseal.

Contribution

We explore how IRMAseal can be expanded with a signature scheme, by considering three options proposed by the creators of IRMAseal [24]:

1. Using IRMA signatures[6] directly. An IRMA signature is made directly on the message.
2. Using IRMA signatures on a public key. An IRMA signature is made on a public key, which is in turn used to sign messages.
3. Using identity-based signatures. Users make a signature on messages, using a key they get from the TTP.

These options are formalized and analyzed in chapters 5, 6 and 7 respectively. Additionally, we present a fourth solution (in chapter 8) in which signatures themselves (not signing keys) can be requested from a TTP.

We first define the properties desired from the solutions in chapter 3. Next, a common technique used in all four schemes is presented in chapter 4, and each proposed solution is discussed in chapters 5 through 8. After comparing the properties actually offered by each option in chapter 9, we give an overview of the related work in chapter 10 and reflect on the findings in chapter 11.

[4] https://postguard.eu/

[5] https://cryptify.nl/

[6] Apart from disclosing attributes, IRMA also offers attribute-based signatures. These are discussed in section 5.1.

Chapter 2

Preliminaries

In this chapter, we present the necessary background information on the workings of IRMA, identity-based encryption in general, and how they are used by IRMAseal. Finally, we discuss the limitations of the trust model behind IRMAseal and introduce a planned improvement: federation.

2.1 IRMA

IRMA is a privacy-friendly identity-management platform that allows users to prove certain properties—known as *attributes*—about themselves, without disclosing any other information [6]. For instance, a user can prove that they are over 18 years old, without revealing their name or date of birth. Within IRMA, the following parties exist:

Users are natural persons who use the IRMA smartphone app to store and reveal attributes to verifiers.

Issuers are parties that give out (issue) attributes to users. For instance, a university can issue a student card, with attributes such as `is student` and `studentnumber`. It is the responsibility of an issuer to only give out attributes to users that are supposed to have them. The rigor of the authentication done by issuers determines the reliability of the attributes that the issuer provides.

Verifiers are parties that wish to know something about users. They ask a user to disclose attributes. For instance, a bar can ask a user to prove that they are over 18 years old.

IRMA uses zero-knowledge proofs (see section 5.1), which allows users to prove that they have attributes, without revealing any other information, and without making it possible for anyone else to reuse these proofs. The process of showing and proving ownership of attributes is called *disclosure*.

One additional important party that exists in the IRMA ecosystem is a *scheme manager*. This is an entity that distributes information about which issuers exist, what kinds of attributes they issue, and their public keys. Normally, all parties in IRMA trust and periodically contact a scheme manager, but all of the information they provide can also be set manually.

2.2 Identity-Based Encryption

Identity-Based Encryption (IBE) is a form of public key cryptography in which any string is a valid public key [4]. That is, there is a trusted third party, from now on known as a Private Key Generator (PKG), that can generate a private key corresponding to any given public key. In IBE, a string used as a public key is also referred to as an *identity*, hence the term identity-based encryption.

The main advantage of IBE over traditional public key cryptography is that only a single (master) public key needs to be distributed to allow for encryption between any two parties. Additionally, messages can be addressed to a user not only without first obtaining their public key, but even before that user uses the system for the first time.

Formally, the following artifacts are defined in IBE:

Master Public Key (MPK): A key known to all parties, generated by the PKG. This key is used together with an identity to encrypt messages.

Master Secret Key (MSK): A key known to the PKG only, that can be used to generate user secret keys for any identity.

Identity: A string for which data can be encrypted. In general IBE, any string is a valid identity, but it is common to use some structured format. See subsection 2.3.1 for the identities used in IRMAseal.

User Secret Key (USK): A key *extracted* from the MSK, and an identity. The PKG transmits the USK to users once they have presented proof of their identity. We write usk_{id} for the USK corresponding to identity id.

Once the PKG has generated its master keypair (mpk, msk), the following operations can be performed:

$\mathsf{Extract}_{msk}(id) \to usk_{id}$**:** Extract a user secret key usk_{id} from the master secret key msk and an identity id.

$\mathsf{Encrypt}_{mpk,id}(M) \to C$**:** Encrypt a message m with an identity id, for the PKG with public key mpk.

$\mathsf{Decrypt}_{usk_{id}}(C) \to M$**:** Decrypt a ciphertext C with a user secret key usk_{id} to obtain a message M.

Combined, the following holds:

$$\mathsf{Decrypt}_{\mathsf{Extract}_{msk}(id)}\left(\mathsf{Encrypt}_{mpk,id}(M)\right) = M \tag{2.1}$$

2.3 IRMAseal

IRMAseal uses IBE with authentication through IRMA, to provide a privacy-friendly and simple way of encrypting messages. By using IRMA attributes as an identity, many different kinds of information can be part of an identity. For example, an IRMAseal identity could be `is student: true` or `email: "john.doe@example.org"`. An identity need not be uniquely identifying, and can even consist of combinations of IRMA attributes. We discuss IRMAseal identities in more detail in subsection 2.3.1.

Within IRMAseal, the involved parties are, broadly speaking:

Sender: The party that wishes to send an encrypted message.

Recipient(s): The party that wants to read the encrypted message. Each recipient needs to have the IRMA app to prove that they are allowed to read a message.

PKG: The trusted third party that has a master keypair, and can extract USKs from the master secret key.

However, this is a simplified view. There is a distinction between the recipients' IRMA apps (that they use to prove their identity), and their IRMAseal client: the software doing the actual decryption. Furthermore, the IRMA infrastructure and issuers are ignored. The entire IRMA ecosystem is involved in authenticating a user against the PKG.

The PKG is an IRMA Verifier. When a recipient requests a USK, the PKG asks the recipient to disclose attributes to satisfy the identity. If the recipient is indeed allowed to get the secret key for that identity, the PKG proceeds to generate the key and sends it to the recipient.

Figure 2.1 shows how IRMAseal works.

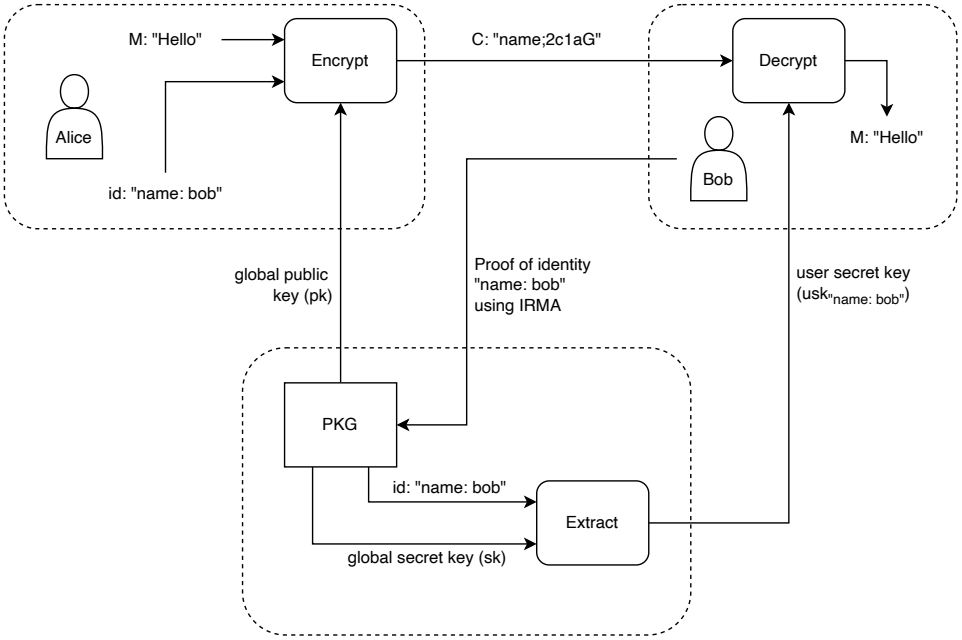

Figure 2.1: A message from Alice to Bob in IRMAseal. `id` is simplified, see subsection 2.3.1.

2.3.1 Identities

As described earlier, identities in IRMAseal and other IBE do not need to be uniquely identifying. Attribute-Based Encryption (ABE), in which messages can be addressed to parties who possess certain attributes, can be achieved by using attributes as an identity. The simplest example of ABE via IBE is using an email address as identity. Messages encrypted for `bob@example.org` can be decrypted by anyone who can prove ownership of the address. This corresponds to attribute-based encryption where an email address attribute is used. In case multiple people own the address (e.g. a mailing list), it is also not uniquely identifying. In IRMAseal, identities consist of a conjunction of IRMA attributes and their values, and a timestamp. IRMA attributes are named as `scheme.issuer.credentialtype.attribute`, so an IRMAseal identity could be for example (formatting is simplified):

```
{
  "conjunction": [
    {
      "attribute": "pbdf.pbdf.email.email",
      "value": "bob@example.org",
    },
    {
```

```
      "attribute": "pbdf.pbdf.mobilenumber.mobilenumber",
      "value": "+31612345678",
    },
  ],
  "timestamp": "2022-09-01 13:37",
}
```

To get the corresponding user secret key, a user needs to prove to the PKG that they possess the required `email` and `mobilenumber` attributes.

The timestamp indicates a moment after which a user must possess the required attributes to get the USK from the PKG. This serves to limit the validity of identities. Messages encrypted for the same attributes but with a different timestamp cannot be decrypted with the same USK. In the same way, it prevents someone from getting USKs with his attributes when he has them (e.g. while he is a doctor), and using them later to decrypt messages when he is no longer a doctor.

Hidden Policies

To protect the privacy of users, IRMAseal uses *hidden policies*. A hidden policy is a version of an identity, from which all of the values have been left out (within IRMAseal, the identity is sometimes referred to as "policy", as it is a policy to determine who can get a USK). While messages are encrypted for an identity that includes the values of all attributes, we assume that recipients can infer the values themselves, even if the values are not included (unencrypted) in the ciphertext. This way, a message can be encrypted for example to a social security number: any recipient can simply try filling in their social security number in a hidden policy, so it's not necessary to expose the value in the ciphertext.

As an example, consider the following identity:

```
{
  "conjunction": [{
    "attribute": "pbdf.nijmegen.bsn.bsn",
    "value": "123456789",
  }],
  "timestamp": "2022-09-01 13:37",
}
```

When encrypting a message for this identity, only the hidden policy

```
{
  "conjunction": ["pbdf.nijmegen.bsn.bsn"],
  "timestamp": "2022-09-01 13:37",
}
```

needs to be included in the ciphertext. A recipient with BSN 123456789 can simply try to decrypt with their BSN filled in, and it will succeed. Hence, IRMAseal ciphertexts include hidden policies, and not the actual values. We will refer to identities (which include the values) as *ids*, and to hidden policies as *pols*

2.3.2 IBKEM

To achieve higher efficiency and better security guarantees than what is offered by 'classic' IBE schemes, IRMAseal uses an Identity-Based Key Encapsulation Mechanism (IBKEM). This is a modified IBE, which, instead of Encrypt and Decrypt, offers algorithms $\mathsf{Encaps}(mpk, id) \to C_{SS}, SS$ which generates and encapsulates a shared secret SS to a ciphertext C_{SS}, and $\mathsf{Decaps}_{usk_{id}}(C_{SS}) \to SS$ which decapsulates the shared secret from C_{SS}. This shared secret can then be used as a key in a more efficient symmetric encryption scheme. IRMAseal uses AES-128-GCM [12] for this.

The IBKEM IRMAseal currently uses is *CGWKV*: an IBKEM based on *CGW* (an IBE scheme by Chen, Gay, and Wee [10]) and a transformation approach from Kiltz and Vahlis [21]. Previously, IRMAseal has used *CGWFO* (*CGW* with the Fujisaki-Okamoto transformation [18]) and *Kiltz-Vahlis IBE*$_1$ [21].

It is also possible to encrypt a message with one shared secret, that can be derived by people with different identities. The process to do so is called multi-recipient encapsulation, with algorithms $\mathsf{MEncaps}(mpk, ids) \to C_{SS}, SS$ and $\mathsf{MDecaps}_{usk_{id_x}}(C_{SS}) \to SS$. Without digging into the details, IRMAseal needs this to be able to encrypt a message for a disjunction of identities in combination with the hidden policy approach. This is done by including in the ciphertext C_{SS} a separate IBE-encrypted number for each identity. In chapter 4 we discuss the details of this technique, and propose a small modification.

The ciphertext of an IRMAseal message consists of a set of hidden policies *pols*, some IBE encrypted numbers that contain the shared secret C_{SS}, and the symmetrically encrypted message C_M. With $\|$ denoting some injective way to concatenate strings, we have $C = pols \| C_{SS} \| C_M$.

2.3.3 Protocol

We can now present the IRMAseal protocol accurately. Diagrams 2.2 and 2.3 show how encryption and decryption work. $\mathsf{MEncaps}(mpk, ids)$ uses the multi-recipient IBKEM to generate and encrypt a shared secret to all of the identities in *ids*, $\mathsf{MDecaps}_{usk_{id}}(C_{SS})$ then returns the shared secret encrypted in C_{SS}, using the USK for one of the identities for which the secret was encapsulated. $\mathsf{AESEncrypt}_{SS}(M)$ uses a shared secret to encrypt M with AES.

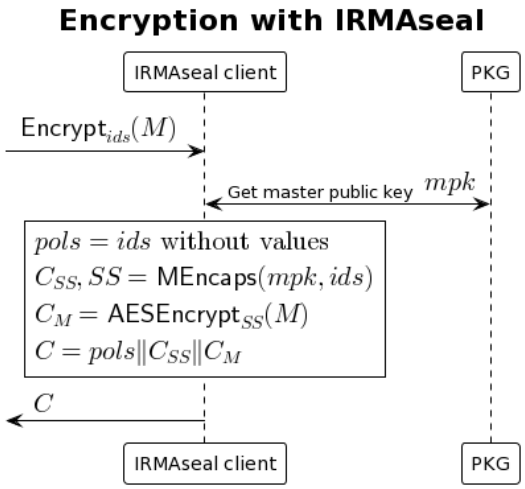

Figure 2.2: Encryption with IRMAseal.

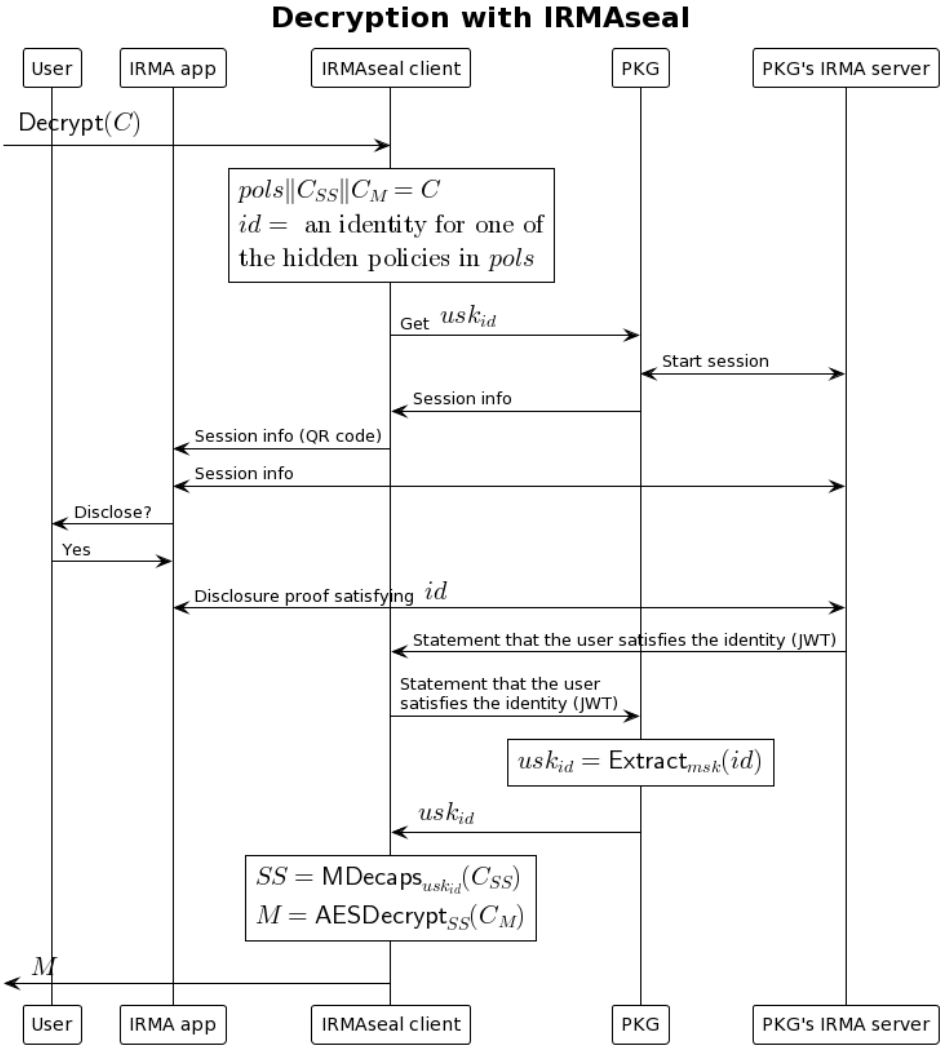

Figure 2.3: Decryption with IRMAseal.

2.3.4 Trust model

Since the PKG decides whether a user gets a USK, the security of IRMAseal is heavily dependent on the PKG. The PKG must not be malicious, and the IRMA authentication also needs to be secure. The security of IRMAseal is based on the following assumptions:

1. The PKG's master secret key is not leaked.
2. The PKG is honest: it does not generate USKs for users that do not satisfy the identity.
3. IRMA authentication is secure: only users that have the required attributes can convince the PKG of that.
4. The private keys of IRMA issuers are not leaked.
5. The issuers of the used IRMA attributes are honest: they don't give out attributes to people that shouldn't have them.

If any of these assumptions do not hold, an attacker (possibly one of the parties) can get the USK for an identity. Specifically, the PKG can always decrypt any message, and IRMA issuers can decrypt messages encrypted for attributes that they can issue. For example, the municipality of Nijmegen could falsely issue a social security number (BSN) attribute `pbdf.nijmegen.bsn.bsn: "123456789"`, and use it to get the USK for identity:

```
{
  "conjunction": [{
    "attribute": "pbdf.nijmegen.bsn.bsn",
    "value": "123456789",
  }],
  "timestamp": "2022-09-01 13:37",
}
```

Furthermore, the PKG is a single point of failure. If it is offline, or its master secret key is lost, users cannot decrypt messages. Finally, the PKG can also keep track of when messages are read, because it gets to see the identities (i.e. including the values, which can even include special categories of personal data such as social security numbers) for which it generates USKs. This limits the privacy of users, even if the confidentiality of the content of messages is not violated.

2.3.5 Federation

The power that lies with the PKG and IRMA issuers is undeniably a concern for the confidentiality and availability of messages encrypted with IRMAseal. The creators of IRMAseal are aware of these concerns, and are working on a federation model, in which multiple PKGs can exist.

In such a model, any organization can set up a PKG, and the consuming software can choose a PKG when encrypting. The PKG that is used to encrypt a message is also the one that can be used to decrypt it, since each PKG has a separate master secret key. Technically, IRMAseal already supports this: federated PKGs can be completely independent, so users can already choose between them.

The way to discover PKGs and determine trust in them is up to consumers of IRMAseal, such as PostGuard for email encryption. This distribution of trust is out-of-scope for this thesis. However, the concept that there can be multiple independent PKGs fundamentally changes the trust model of IRMAseal. Hence, it is important to consider its implications. Throughout this thesis, we assume that there can be multiple PKGs when reasoning about trust, without going into the details of how PKGs are chosen.

Chapter 3

Desired Properties

We start by defining the algorithms that a signature scheme for IRMAseal should provide.

- $\mathsf{SignAndEncrypt}_{attrs,ids}(M) \to C$ adds an attribute-based signature using attributes $attrs$ to message M, and encrypts it for recipients with identities ids. $attrs$ should at least be allowed to be any conjunction of attributes.

  A more flexible signing identity can be nice to have. For instance, it may be more privacy-friendly if $attrs$ supports hiding attribute values from the recipients.

- $\mathsf{DecryptAndVerify}(C) \to M, attrs$ decrypts ciphertext C and verifies the signature on it, returning the message and the attributes used to sign it, if the signature is valid.

To compare different signature schemes for IRMAseal, we determine the security, privacy, and technical properties that are required or desired.

3.1 Security

Any signature scheme should offer some basic security guarantees. We expect them to give at least the same level of confidence as the encryption done by IRMAseal, both in terms of their trust model and in their cryptographic security strength. These basic security properties are:

Authenticity A signature should provide a high level of confidence in the identity of the signer. To achieve this confidence, it should be hard to forge a signature, and the number of parties that need to be trusted in order to trust a signature should be as small as possible. By definition of an identity, the trusted parties will need to include the IRMA issuers that can issue the attributes used in the identity of the signer. However, it is a significant advantage if there is no need to trust for example an IRMAseal PKG for the authenticity of messages.

Integrity A signature should provide a high level of confidence that the message has not been tampered with. This also means that a valid signature on one message should not be valid on any different message.

A related requirement is that a signature provides confidence that the signer intended to sign the message. This property—sometimes referred to as "What You See Is What You Sign" or WYSIWYS, described by Landrock and Pedersen in [22]—should prevent an attacker from tricking users into unknowingly signing something they don't intend to sign.

Finally, signature schemes are often expected to provide non-repudiation. This means that the signer cannot afterwards deny having signed a message. Although Adida et al. and Specter et al. note in [1, 31] that non-repudiation is not always required or privacy-friendly, many applications do require or assume it, so we focus on schemes that provide non-repudiation. A repudiable solution, that still provides authenticity, but only to the intended recipients, is an interesting topic, but out of scope for this thesis.

3.2 Privacy

Aside from security, privacy is a major concern for IRMA and IRMAseal. We expect the signature scheme to offer as much privacy as possible. Of course, the meaning of 'privacy' is subjective and can depend on the context of a message that is being sent. Within this thesis, we understand 'privacy' to encompass the following properties:

Unlinkability If a user signs multiple messages with (different or identical) non-identifying IRMA attributes, it should be impossible to link the signatures to the same user.

Confidentiality of sender identity Only authorized recipients of a message can determine the attributes of the sender used in the signature. Parties who cannot decrypt the message cannot extract the attributes of the signer. Furthermore, nothing other than the used attributes should be revealed, even to authorized recipients.

Confidentiality of message content In order to sign and verify, only the signer and recipient should need to know the message content. The plaintext should not be revealed to any other party, including for example a PKG.

3.3 Technical

Finally, there are some relevant technical properties.

Minimize dependencies Any dependencies that need to be added to IRMAseal are detrimental to both the security and maintainability of IRMAseal. Therefore, for instance, it is preferable to use the same cryptography that is used in IRMAseal, instead of adding other cryptographic primitives.

Keep PKG stateless Except for a master keypair, the PKG does not need to store any state, which makes it potentially scalable. Hence, we prefer a signature scheme that does not require the PKG to store any additional state, and specifically, a PKG should not need to keep records of all produced signatures (both for simplicity and scalability, and for privacy).

Minimize signature size Signatures should be as small as possible to limit communication overhead.

Minimize signing communication Signing a message should require as little communication as possible. Ideally, it could be done offline.

Minimize verification communication Verifying a signature should require as little communication as possible. Ideally, it could be done offline.

Chapter 4

Naive sign-and-encrypt

There is one challenge in the design of a signature scheme that needs to be faced in all four proposed approaches. Before presenting and analyzing our schemes in the next chapters, we first discuss this challenge and a solution to it.

A simple, generic approach to creating a signature scheme for encrypted messages is as follows, where Sign produces some kind of signature on a message:

$$\mathsf{SignAndEncrypt}_{attrs,ids}(M) = \mathsf{Encrypt}_{ids}\Big(M \| \mathsf{Sign}_{attrs}(M)\Big)$$

Davis noted in [13] that there is a flaw in this naive sign-and-encrypt approach, known as the *naive sign-and-encrypt problem*, or *surreptitious forwarding*. This flaw appears in numerous protocols, including OpenPGP and S/MIME.

The signature in this scheme is on the plaintext only, without including the intended recipients. At first glance, that doesn't appear problematic. After all, the recipients know for certain who wrote the message. However, in this approach, a recipient does not know whether the signer meant to send the message to them. Consider the following scenario:

1. Alice writes to Bob:

$$\mathsf{Encrypt}_{\{Bob\}}\Big(\text{“}I\ love\ you\text{”} \| \mathsf{Sign}_{Alice}(\text{“}I\ love\ you\text{”})\Big)$$

2. Bob decrypts this, giving him the plaintext *“I love you”*, and the valid signature from Alice $\mathsf{Sign}_{Alice}(\text{“}I\ love\ you\text{”})$.
3. Bob can then send a message to Charlie, posing as Alice:

$$\mathsf{Encrypt}_{\{Charlie\}}\Big(\text{“}I\ love\ you\text{”} \| \mathsf{Sign}_{Alice}(\text{“}I\ love\ you\text{”})\Big)$$

4. When Charlie decrypts this, he gets the plaintext *“I love you”*, and the valid signature from Alice, falsely convincing him that Alice loves him.

To solve the problem, we just need to include the identities of the intended recipients in the signature, such that the signature is only valid for

messages encrypted for the original recipients. This way, a signed message cannot be secretly replayed to different recipients.

Unfortunately, including the *identities* in a hash is not an option in IRMAseal, because ciphertexts contain only hidden policies, not full identities[1] (see section 2.3.1). This means that it would not always be feasible for recipients to recreate the identities, and thus the hash needed to verify the signature. We need an alternative.

The first alternative to consider is to include the *hidden policies*, instead of identities. However, this approach fails to prevent the replaying example we discussed: the hidden policy for a message to Bob is the same as that to Charlie, specifying only that a `name` attribute is required.

Next, we could include the hash of each separate identity in the ciphertext. Then, a signature can be made over the hashes of all identities. Anyone can verify that the signature corresponds to the hashes of identities appended to the ciphertext. A specific recipient can verify that the signer intended to send the message to at least their identity (and possibly others), by checking that the identity that the specific recipient used for decryption, matches one of the hashes added to the ciphertext, on which the signature was made.

Mathematically, we would get:

$$\begin{aligned}\mathsf{SignAndEncrypt}_{attrs,ids} = pols \| C_{SS} \| h(id_1) \| ... \| h(id_n) \| \\ \mathsf{AESEncrypt}_{SS}\Big(M \| \mathsf{IRMASig}_{attrs}\big(M \| h(id_1) \| ... \| h(id_n)\big)\Big) \quad (4.1)\end{aligned}$$

To verify, a recipient who decrypts the message using usk_{id_x} computes $h(id_x)$, and checks that it is one of the hashes in the ciphertext, and that the signature is indeed $\mathsf{IRMASig}_{attrs}\big(M \| h(id_1) \| ... \| h(id_n)\big)$.

One downside to this solution is that it makes it easier to derive values of attributes from the hidden policies: given a hash of an identity, say $h(id_x)$, it becomes possible to try out all possible values of that identity. For example, if from the hidden policy it is clear that id_x contains a phone number, it is easy to try filling in every possible phone number until the resulting identity hashes to $h(id_x)$. Without such a hash being given, an attacker would need to request USKs for (and hence prove that they own) any possible phone number to find one that decrypts successfully.

Even this issue can be avoided, however. The final solution is to not use a hash directly on an identity, but on an identity together with a random number that is only known to people who own the identity. In fact, with a small change to the way multi-recipient key encapsulation (subsection 2.3.2) is done in IRMAseal encryption, we can get such a number by reusing some numbers included in IRMAseal ciphertexts. We propose a small modification to the MKEM implementation in IRMAseal.

[1] Including the full identities in the encrypted part of a message would also not be privacy-friendly, as all recipients would see each other's attributes

Modifying multi-recipient key encapsulation

The existing multi-recipient key encapsulation mechanism used in IRMAseal works as follows: first, one shared secret SS is generated. Then, for each recipient identity id_i, a separate shared secret and corresponding IBE ciphertext is created: $C_{SS_i}, SS_i = \mathsf{Encaps}(mpk, id_i)$. Finally, $SSS_i = SS \oplus SS_i$ is added to ciphertext per recipient, such that a recipient with id_x can get the single shared secret using $SS = SSS_i \oplus \mathsf{Decaps}_{usk_{id_x}}(C_{SS_i})$.

Intuitively, we would like to reuse SS_i as the random value to hash id_i with. However, in the current construction, any recipient x can compute SS_i for any other recipient i: $SS_i = SSS_i \oplus SS = SSS_i \oplus \mathsf{Decaps}_{usk_{id_x}}(C_{SS_x})$. Then, recipient x can use the hashed identities to derive the attributes that we were trying to keep hidden.

By modifying the multi-recipient key encapsulation mechanism, we can keep SS_i hidden from any other recipient x. We do this by hashing (with a preimage-resistant h) SS_i before XORing it with SS: $SSS_i = h(SS_i) \oplus SS$. Decapsulation is then done with $SS = SSS_i \oplus h\big(\mathsf{Decaps}_{usk_{id_x}}(C_{SS_i})\big)$. Different recipients can derive $h(SS_i)$, but not SS_i. As such, SS_i is now a suitable value to reuse for hashing identities. We refer to this modified multi-recipient key encapsulation mechanism later on with $\mathsf{MEncaps}'(mpk, ids)$, which returns $C_{SS}, SS, SS_1, ..., SS_n$, and $\mathsf{MDecaps}'_{usk_{id_x}}(C_{SS})$ returning SS and SS_x.

We claim without proof that this construction does not influence the security strength of encryption if a collision-resistant hash is used. The security of encryption when h is a random oracle is clear: $h(SS_i)$ is then just as 'random' as SS_i, so the change does not lead to an advantage that can be used to guess SS from SSS_i without knowing SS_i.

Using this modified multi-recipient key encapsulation mechanism, we can reuse each SS_i, putting a signature on $hs = h(id_1 \| SS_1) \| ... \| h(id_n \| SS_n)$, and include hs in the ciphertext. It does not matter whether these hashes are in the encrypted part or not.

Now, only someone with usk_{id_x} can get SS_x, so no others can use $h(id_x \| SS_x)$ to derive an attribute value in id_x. The recipient can verify that the message was intended for id_x by computing $h(id_x \| SS_x)$, checking that it matches one of the hashes included in the ciphertext, and checking that the signature is indeed on all of these hashes.

4.1 Alternatives

While we believe this modification of IRMAseal to be secure and helpful to ensure a binding of signatures to the intended recipients, it is not always required to have such a binding, for example if it is already clear from the plaintext who the intended recipients are. We also have not given a formal proof for the security of this modification. There are several alternatives to

implementing this proposed change:

- Instead, adding separate secret numbers generated specifically to diversify the hashes of identities, that are encrypted with IBE separately. This would work just as well, without changing the existing encryption, but would increase the message size with hundreds of bytes per identity, instead of only a few bytes per identity.

- Not binding signatures to the recipients. Each proposed scheme can simply be used without the proposed modification, at the cost of not preventing the naive sign-and-encrypt problem described above. While this seems like a bad idea, we have also seen that this is a common choice in practice (including in OpenPGP and S/MIME). If plaintexts include an indication of the intended recipients, it doesn't matter if the signature doesn't specify that itself.

- Implementing another solution. Although we have not worked out other solutions in detail, a simpler solution may exist. Specifically, Davis (in section 5.2 of [13]) presents a general solution called *sign-encrypt-sign* in which not only the plaintext but also the ciphertext (the encryption of the *signed* plaintext) is signed. Perhaps this can be used directly in our *IRMA signature on public key* and *IBS* approaches, and by including an additional public key (for a single use on the encrypted message) in the message for our *IRMA signature on message* and *Signature form TTP* approaches. Because this solution cannot be used in the same way for each scheme, we have not worked it out in detail.

Chapter 5

IRMA signature on message

One way to add attribute-based signatures to IRMAseal is by using IRMA's signatures directly on a message. Let us first dive into the details of IRMA's signature scheme. Then, we work towards a functional and secure scheme, and analyze its properties.

5.1 IRMA signatures

When an IRMA user discloses attributes, they do a zero-knowledge proof based on the Schnorr identification protocol and the Fiat-Shamir transformation [14, 9]. Broadly speaking, the following happens [7]:

1. The verifier sends a random number—the *nonce*—to the user.
2. The user does a computation on the nonce that only owners of the required attributes can do, and sends the result back to the verifier.
3. The verifier verifies that the computation was correct.

The use of a nonce gives the verifier certainty that the user is actually present, and not an attacker replaying an old disclosure session they recorded somewhere.

IRMA signatures [6, 17] are very similar to regular disclosure sessions. The difference is that rather than the verifier sending a nonce, a hash of the signed message is used as the nonce. Additionally, the nonce includes another random number, and a timestamp signature[1] which allows the verifier to verify that the attributes had not expired at the time the signature was created. The user then does a computation on the nonce that only owners of the required attributes can do, and the result is the signature. A verifier can then verify that the computation is correct, assuming the hash of the presented message is the used nonce. Figure 5.1 shows how IRMA signatures are created.

Hampiholi et al. [17] show that IRMA signatures are secure. That is, it is not possible to create a forged signature, even given a large number of other messages and signatures by the attacker's choice. Hence, a valid

[1] A signature by a trusted Timestamp Authority, that cannot be requested before the actual time. This provides certainty that the message existed at the specified time, and was not created later.

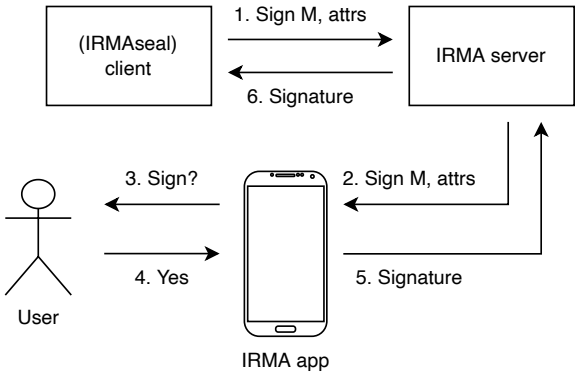

Figure 5.1: Creation of an IRMA signature.

signature on a message convinces a verifier that the message was signed by someone who possesses the attributes used in the signature and that those attributes had not expired at the time of signing.

5.1.1 Creation and verification

Since IRMA signatures are basically disclosure proofs, they are made by the IRMA app of a user. Like disclosures, signatures are currently made by the app only after the app gets a request from an IRMA server (either by opening a URL or by scanning a QR code). That means that in order to create an IRMA signature, we need to first contact an IRMA server, present a QR code to the user, wait for the user to allow creating the signature, and finally get the signature back via the IRMA server.

Verification is normally also done by an IRMA server. However, although this is not supported by the creators of IRMA, Leon Botros, one of the creators of IRMAseal, has implemented a proof-of-concept WebAssembly module[2] that can perform verification locally. We assume IRMA signature verification to be available locally to an IRMAseal client, but this may require more maintenance effort than using an IRMA server.

5.2 Scheme

Based on IRMA signatures, we can easily create a signature scheme for IRMAseal. It seems logical to just create a signature on the message content, append that to the plaintext, and then encrypt it. That is, with $attrs$ representing the attributes that the signer uses to sign the message:

$$\mathsf{SignAndEncrypt}_{attrs,ids}(M) = \mathsf{Encrypt}_{ids}\Big(M \| \mathsf{IRMASig}_{attrs}(M)\Big)$$

[2] `https://gitlab.science.ru.nl/ilab/irma-signature-verify/`

This suffers from two flaws. One of them is precisely the problem discussed in chapter 4.

5.2.1 Plaintext confidentiality

The second issue is that—during the creation of the IRMA signature—the plaintext is exposed to a third party. That is, in order to create an IRMA signature, currently, a request containing the plaintext needs to be sent to an IRMA server, which in turn communicates it with an IRMA app that creates the signature.[3]

This means that the plaintext is exposed to two parties. First, the IRMA app gets to see the plaintext, so it can make sure that the user really intends to sign that specific message. Second, there is the IRMA server (run for example by a PKG) that communicates between the IRMAseal client and the IRMA app.

The latter *only* needs to get the plaintext so that it can forward it to the IRMA app, but as a consequence, the IRMA server needs to be trusted completely. That is, the trust in this IRMA server needs to be even greater than that in a PKG. A PKG can decrypt any message, but only when it has access to a ciphertext, which it typically should not have. The IRMA server, on the other hand, gets access directly to any plaintext that is signed!

While this approach of signing a hash of the plaintext does not expose the plaintext directly, it would still make it possible for the IRMA server to verify that a signature was made, given the plaintext. For example, the IRMA server could find out whenever the specific message *"I love you"* is signed, by looking for requests to sign $h(\text{"}I\ love\ you\text{"})$. This is less of a problem, but still not ideal, and can be avoided: the final solution here is to include a random number (nonce) generated by the signer in the hash that we sign, and in the encrypted part of the message we send. This way, even an IRMA server that is looking out for specific hashes cannot predict what hash would correspond to the message *"I love you"*, but a recipient who can decrypt the message can use the included nonce to perform verification.

Combining solutions to the two flaws, we get a scheme in which we sign not the message, but a hash that includes the hashes of identities, the message, and a nonce. Figure 5.2 shows encryption and signing with this scheme. As you can see, three additional parts are included in the ciphertext when compared to the encryption-only currently offered by IRMAseal: a nonce, the hashes of each identity (and corresponding secret), and the signature itself. Decryption, shown in Figure 5.3, is mostly the same but followed with steps to verify the signature.

[3] In section 11.4 we suggest exploring changes to IRMA that allow signature creation without an IRMA server being able to read and modify traffic.

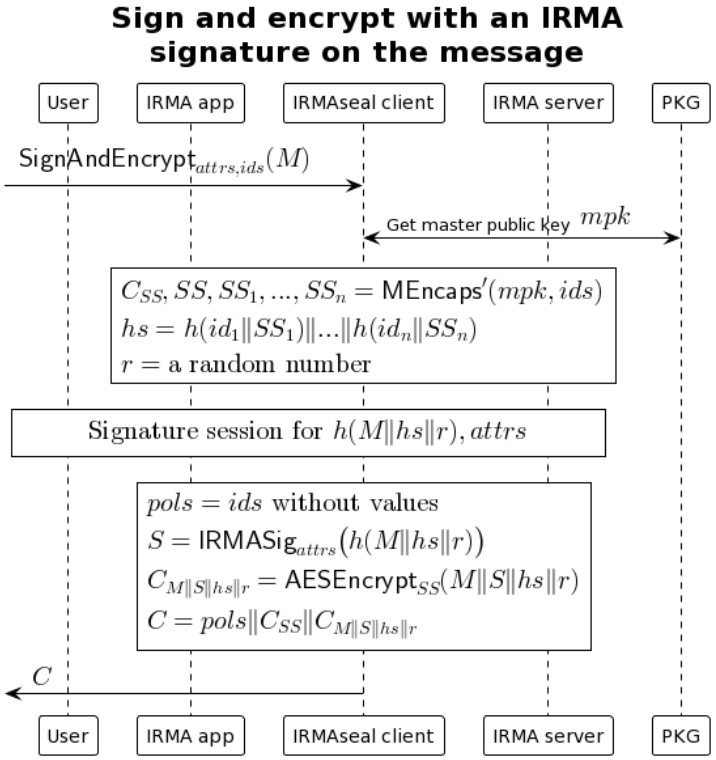

Figure 5.2: Signing and encrypting a message with an IRMA signature on the message.

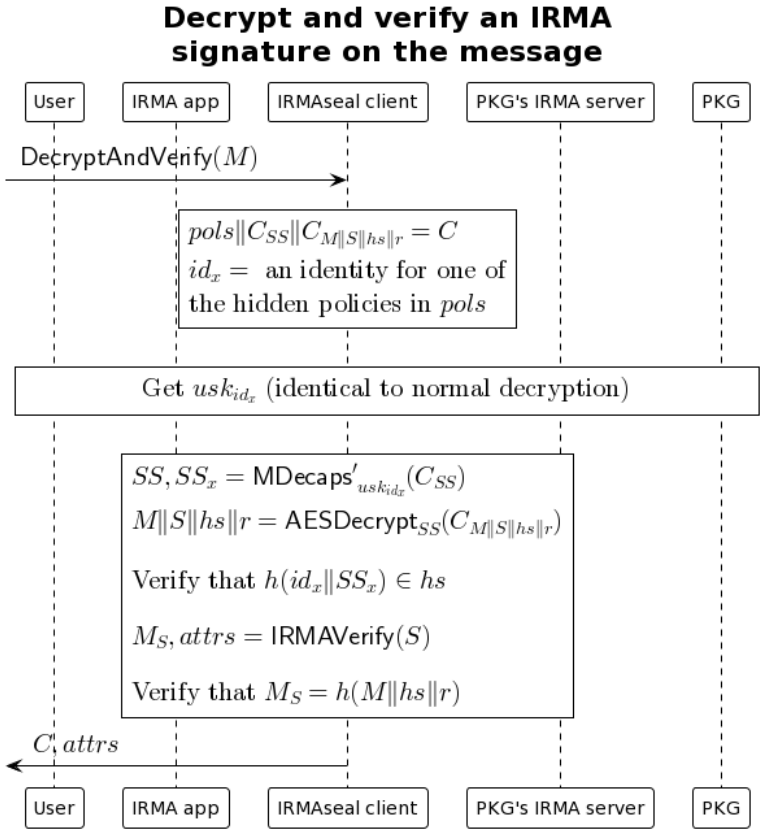

Figure 5.3: Decrypting and verifying a message signed with an IRMA signature on the message.

Signature creation

While the above scheme solves the naive sign-and-encrypt problem, and preserves plaintext confidentiality, the solutions did introduce a new problem. Before signing, the IRMA app displays the message that it will sign, so that users can confirm that they want to do so. This is done to prevent attackers from tricking users into signing something they don't want to. However, by not signing the plaintext but a hash, the "message" shown to the user for confirmation becomes illegible to the user. The best we can do is to make the message slightly more readable and make hashes used for IRMAseal signatures distinguishable from any other hash. For example, we can sign `IRMAseal-signature-hash:9be31c8150ac(...)` and inform the user of what "message" they should expect to sign.

5.3 Analysis

5.3.1 Security

Since IRMA signatures have been shown to be secure, our scheme trivially offers authenticity and integrity. The cryptographic security strength behind this is equal to that behind IRMA disclosures, and hence that of authentication against an IRMAseal PKG.

Next, let us examine the trust model behind the provided authenticity and integrity. Assuming the security of IRMA signatures, attack scenarios are:

1. A malicious or compromised issuer, wrongfully giving out attributes that can be used to create a signature.
2. A malicious or compromised Timestamp Authority (chosen by the IRMA developers) could be used to create an IRMA signature with attributes that had expired before the time of signing.
3. Tricking a user into signing an attacker's message.

Of these scenarios, the first is inherent in the IRMA system as a whole. When an issuer wrongly gives out valid attributes, there is no way to distinguish between valid and invalid attributes, regardless of whether IRMA signatures or disclosures are used. Hence, no signature scheme based on IRMA attributes is possible that does not have this vulnerability.

The second scenario—a malicious or compromised Timestamp Authority (or TA)—is intrinsic to IRMA signatures specifically: IRMA disclosures do not involve a timestamp as they are done in real-time with the verifying party. Within IRMA, the TA that is used is determined by each issuer (signatures using attributes from different issuers with different a TA are not supported), so just as the trustworthiness of IRMA attributes depends on how careful the issuer is, it also depends on the issuer's choice of TA. In any case, the risk posed by this attack scenario is not very high: it can only be used to forge signatures *using expired attributes*, not by attackers who never had the required attributes in the first place.

The final scenario—tricking a user into signing an attacker's message—is the most serious. It can be mitigated only by the IRMA app showing the exact message that is about to be signed. However, since our scheme requires signing a hash instead of the plaintext of the message, it is hard for users to prevent signing something they don't want to. Note that IRMA users can be tricked into signing in a context totally unrelated to IRMAseal. At the very least, clearly distinguishing hashes used in IRMAseal—such as by adding a `IRMAseal-signature-hash` prefix—should help to prevent attackers from tricking users into signing just any hash (possibly in a context that has nothing to do with IRMAseal) without realizing it will be abused as a signature in IRMAseal.

Still, the IRMA server used to create a signature for IRMAseal (which communicates with the IRMA app during SignAndEncrypt) could secretly pass a different hash through to the app, making the user sign the wrong message. Hence, this IRMA server either needs to be trusted completely (for example if it is under the control of the signing user) or the IRMAseal client should instruct users to verify that the hash shown by the IRMA app is equal to the hash shown by the IRMAseal client. In the latter case, the IRMA server is still in a powerful position it could try to abuse. In section 11.4 we propose modifying IRMA to remove the intermediary IRMA server for signature creation.

In conclusion, the parties trusted with authenticity and integrity are the IRMA scheme as a whole (including the issuers of the used attributes, and the Timestamp Authority), and of course the IRMAseal client itself. However, while not directly involved in verification, the IRMA server used to create signatures seems to be the greatest risk, as it could trick users into unintentionally signing wrong messages. The only mitigations for this risk are to rely on the user to check that the IRMA app displays the right message (hash), and to use trustworthy IRMA servers, perhaps in a federated fashion.

Non-repudiation Within IRMA, it is usually assumed that credentials do not get leaked: there is usually no revocation possible. Thus, a user cannot convincingly repudiate a signature by stating that their credentials have been stolen. Even if a user leaks their credentials intentionally in an attempt to repudiate a signature, the timestamp included in it proves that the signature was made before the credentials were leaked.

Some IRMA attributes do support revocation[4]. In that case, an IRMA signature proves that the attributes had not been revoked before the time of signing. Hence, in the case of revokable attributes, a signer who wants to repudiate a signature would need to make it plausible that their credentials had been stolen but not yet revoked at the time of signing.

We can conclude that IRMA signatures do provide non-repudiation in this scheme, at least to the extent that non-repudiation is possible in the IRMA ecosystem.

5.3.2 Privacy

Unlinkability is inherently offered by IRMA signatures if non-identifying attributes are used. By signing a specific hash instead of the plaintext, we managed to fully protect the confidentiality of the message content, even if the IRMA server used for signature creation is compromised. The identity of the sender is of course revealed to the IRMA server used for signing, and

[4]Issuers of revokable attributes maintain a kind of list of revoked credentials, called an *accumulator*, such that a user can prove that their credentials are not included in this accumulator [5].

to all authorized recipients, but not to anyone else. Again, the IRMA server used for signing is a weak point.

5.3.3 Technical

The proposed scheme is technically simple. No new cryptographic primitives need to be implemented in Rust to be used by the IRMAseal client, because only the existing IRMA signatures are used. A challenging aspect is that local verification of IRMA signatures, while already implemented as a proof-of-concept, would need to be maintained. This is a large dependency and takes significant effort to maintain. Perhaps one day, the functionality may be offered by IRMA or the IRMA team directly. Without local verification—using an IRMA server (perhaps in a federated fashion) for verification—the trust model behind the authenticity offered by the scheme is much less favorable.

This scheme does not require any changes to the encryption PKGs.

The overhead in message size of the proposed scheme is more or less the size of an IRMA signature, which is around 3000 bytes, depending on the used attributes. On top of that, we need to include one hash (presumably 32 bytes, using SHA256) for each identity of a recipient. This is likely negligible compared to the size of the IRMA signature unless very many identities are used.

The communication required for signing is one IRMA signature session, involving an IRMA server (such as the one used by the PKG) and the IRMA app. This is very similar to a disclosure session, except that the user can pick an IRMA server different from that used by the PKG. The use of a signature session does add the risk of the user getting tricked into signing a wrong hash, which would not be possible in an approach that uses disclosure sessions. Since the IRMA server for signing can be chosen by the signer, a federated approach is possible. For example, students and staff of a university can sign through the university's IRMA server. This limits the risks associated with using an IRMA server that is not directly controlled by the users (they can pick a somewhat trustworthy server, i.e. hosted by a party they trust), as well as possible availability issues resulting from a single point of failure.

Verification can theoretically be done offline, given that the IRMA scheme information (that is normally retrieved from a scheme manager, but can be specified manually) is available.

Chapter 6

IRMA Signature on public key

A second approach based on IRMA signatures is using an IRMA signature on a public key, and in turn, using the corresponding private key to sign a message. This solution makes it possible to sign multiple messages, using only a single IRMA signature, so with a single use of the IRMA app. This may make this method usable in more scenarios, but also has some drawbacks in security, privacy, and simplicity.

6.1 Scheme

This scheme can be instantiated with any choice of public key signature scheme, such as RSA or EdDSA [26, 19]. Using some public key signature scheme defined by algorithms $\mathsf{PKSign}_{sk}(M) \to S$ and $\mathsf{PKVerify}_{pk}(S, M)$, where (pk, sk) is a keypair generated by $\mathsf{PKKeyGen}() \to sk, pk$, we can define the proposed scheme in Figure 6.1 and Figure 6.2. Simply put, we create a keypair, sign it with IRMA, and then use the signed keypair to sign a message. Consequently, multiple messages can be signed with the same public key if the same attributes are needed to sign all of them.

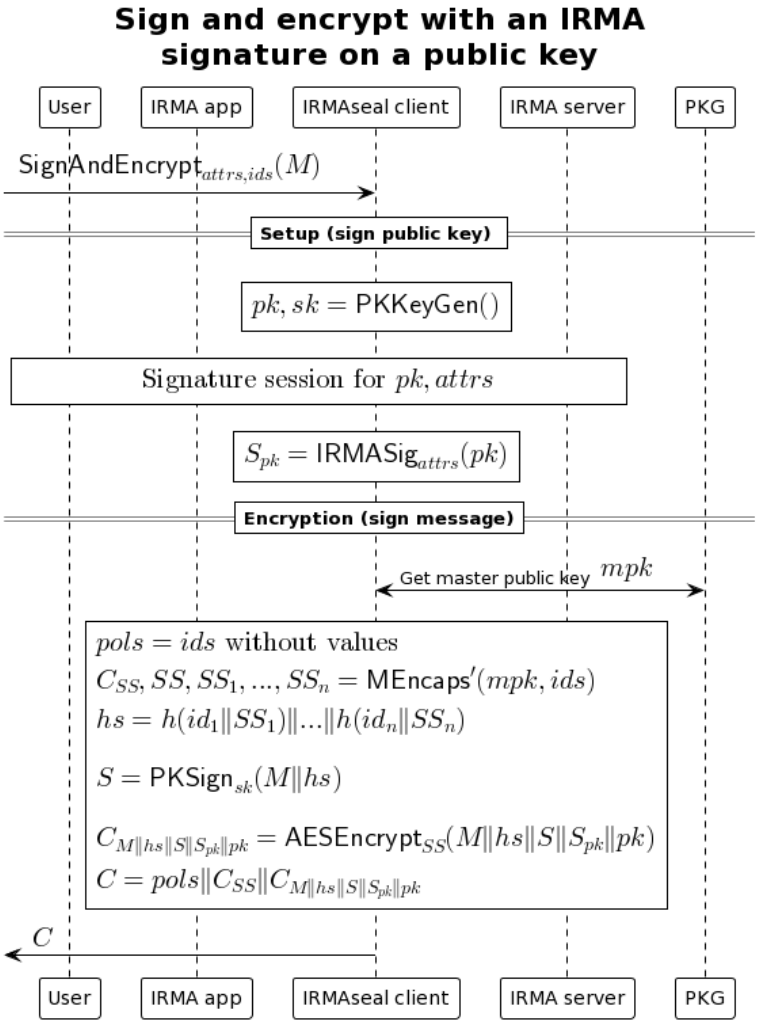

Figure 6.1: Signing and encrypting a message with an IRMA signature on a public key. Multiple messages can be signed with the same public key by repeating the "Encryption" part, as long as all messages are to be signed with the same IRMA attributes.

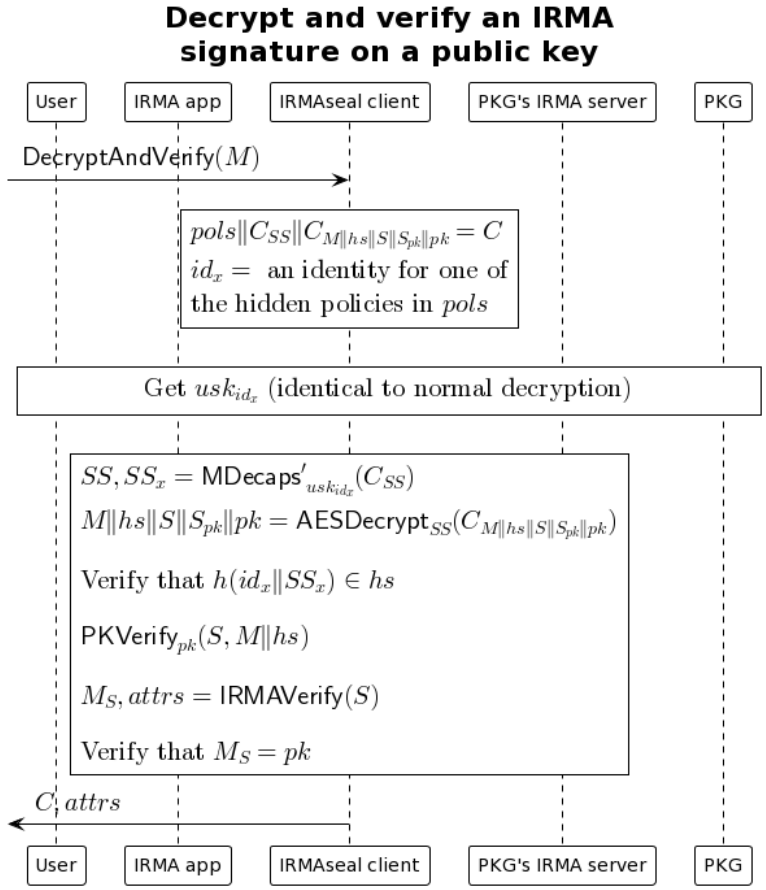

Figure 6.2: Decrypting and verifying a message with an IRMA signature on a public key.

A result of not using IRMA signatures on the message itself is that the trusted timestamp included in IRMA signatures applies only to the public key: the keypair was generated before the moment indicated by the timestamp, when the used IRMA attributes had not yet expired. The timestamp in the IRMA signature says nothing about the moment the actual message was signed, which could be much later than the moment the keypair was generated and signed. To provide certainty that the message was created before the expiration of IRMA attributes, even for messages *received* after the expiry of the used attributes, the proposed scheme can be extended with a separate trusted timestamp on (a hash of) the message.

6.2 Analysis

The main advantage that the current approach can offer over the previous one is that it allows signing multiple messages with the same public key,

i.e. without the sender having to use the IRMA app multiple times for signatures with the same attributes. This advantage directly leads to some limitations in security and privacy.

6.2.1 Security

While the IRMA signature on the public key provides the very same security guarantees as before, the intermediate private key opens up a new attack scenario. The ability to reuse this key makes it possible to sign multiple messages, but it also introduces the risk of leaking the private key. As long as private keys are destroyed immediately after use, there is no added risk (assuming the public key signature scheme is secure). However, to benefit from the reuse of a keypair, the keypair would need to be kept for some time after its creation. Doing this securely is difficult, and having a private key exist within the IRMAseal client at all (this is not the case when using IRMA signatures on the message itself, as those signatures are created in the IRMA app instead) is a risk, for example by exposing the private key to side-channel attacks.

Consequently, even if a signer keeps the private key secure (e.g. by destroying it immediately), recipients can never know with certainty that a sender really did keep the key secret, which not only undermines the recipient's trust in the sender's authenticity, but also means that this scheme does not provide non-repudiation. A signer can always claim that their private key was stolen, or even intentionally leak it to repudiate a message.

Although adding a trusted timestamp to each message, and perhaps adding a revocation mechanism (although the latter would probably require users to keep revocation certificates for all private keys they used, which is impractical), could help mitigate some of the security issues mentioned above, it remains clear that the security of this scheme is far from ideal.

6.2.2 Privacy

A limitation of privacy that directly follows from reusing a private key is the loss of unlinkability. Each message signed with the same private key is clearly linked by the corresponding public keys being equal, regardless of whether the attributes used to sign the public key are identifying. While this is an obvious limitation, that can easily be avoided by not reusing the private key whenever unlinkability is desired, it might be hard for users to understand this.

The confidentiality of the message content is not affected by this scheme, as the public key signature is created and verified locally. The sender identity is again only revealed to the IRMA server used to create the signature on the public key, as well as the authorized recipients, which is not a problem.

6.2.3 Technical

As well as creating and verifying IRMA signatures, an implementation of this scheme needs to be able to generate keypairs, create and verify public key signatures, and optionally (to benefit from sending multiple messages with a single IRMA signature session) store a keypair securely. This introduces a number of new dependencies not needed in the previous approach, increasing the surface area for potential security issues to some degree. Implementations of many public key cryptographic schemes are often carefully audited, mitigating the security implications of this, but the additional dependencies do increase the complexity and size of the implementation.

Similar to when using an IRMA signature on a message directly, the PKG does not need to be changed, but we do need to maintain local IRMA signature verification.

The message size is increased by one IRMA signature, a hash per recipient identity, as well as a public key signature, and the public key that was used. The total overhead for signatures thus depends on the public key signature scheme. For example, using EdDSA instantiated with the Ed25519 curve, the public key takes 32 bytes, and the signature takes 64 bytes. On the other hand, using RSA with 2048-bit keys, both signature and public key take up 256 bytes each [19].

Signing a public key with IRMA requires communication through an IRMA server with the IRMA app. After that, multiple messages can be signed with public key signatures without any more communication. Verification of the public key signatures can also be done offline. The IRMA signature can be verified offline if the IRMA scheme information is available.

Chapter 7

Identity-Based Signatures

Instead of relying on IRMA signatures, it may be more flexible to use an Identity-Based Signature (IBS) scheme instead. An IBS is a signature created with a USK that can be generated by a PKG, on behalf of any string as identity.

Whereas IRMA signatures contain the disclosed attribute values, an IBS can be on behalf of any string as identity, giving us full control of the information included in an identity. It is possible to use a conjunction of attributes with values, but also to use an *IRMA ConDisCon* (a Conjunction of Disjunctions of Conjunctions), enabling signatures on behalf of for example `name: "Alice" OR name: "Bob"` without disclosing which name was actually shown to the PKG, or even to specify that a certain attribute is *not null*, without putting any requirement on the value itself. This makes an IBS approach more flexible privacy-wise. As a downside, a PKG needs to be trusted for authenticity, as it can generate forgeries at will.

Various IBS schemes have been proposed in the literature (many of which are mentioned by [20]), some using a transformation on existing primitives, and others introducing entirely new primitives. One straightforward construction of IBS schemes is using *certification*. This technique is mentioned in a multitude of papers [3, 15, 20]. In this approach, a PKG creates a signature on a user's public key and identity (a *certificate*), and the user uses the corresponding secret key to create signatures. Like our approach in chapter 6, any standard signature scheme can be used for this. Essentially, in this certification-based approach, the USK consists of the user's private key from a standard signature scheme and the PKG's signature on the corresponding public key and the user's identity. A signature consists of a standard signature on the message, the corresponding public key, and the certificate.

The simplicity of this construction shows that a secure IBS is trivial to construct. [3] and [20] indicate that, since such a simple secure construction exists, all other constructions are not aimed at higher security, but efficiency in terms of signature and key size, and signing and verification performance. Since the certification approach is already quite efficient in terms of signature size and signing/verifying cost[1], efficiency is likely not an issue.

[1]For example, using Ed25519 (which targets a 128-bit security level, just like the IBE used by IRMAseal), a certification-based IBS would take only 160 bytes (64 for a signature on a message, 32 for the public key, and 64 for the certificate signature) and some encoding

There is, however, a property that is not offered by all IBS schemes: unlinkability. When multiple signatures are made with the same signing key, they can be linked together, and distinguished from signatures from a different singing key with the same identity. Hence, a certification-based IBS is typically linkable[2].

7.1 Scheme

Without fixating on a specific IBS scheme, we propose a scheme using the algorithms $\mathsf{Sign}_{ussk_{id}}(M) \to S$, $\mathsf{Verify}_{mpsk,id}(S, M)$ and $\mathsf{SExtract}_{mssk}(id) \to ussk_{id}$, where $\mathsf{SExtract}$, $ussk$, $mpsk$ and $mssk$ have an added s compared to their IBE counterparts for Signing-Extract, User Secret Signing Key, Master Public Signing Key, and Master Secret Signing Key, to avoid naming conflicts with the corresponding IBE algorithms and artifacts. Contrary to the previous schemes, this one takes an id as input, rather than $attrs$. This is because in this scheme the signing identity is not constrained to being a conjunction of attributes. Still, id can be replaced with just $attrs$ without any problems. The signing and verification procedures of this scheme are shown in Figure 7.1 and 7.2.

---

of the identity. The verification consists of just two Ed25519 verifications. This is much smaller than both an IRMA signature, and a USK for the IBE used in IRMAseal.

[2]Certification-based IBSs are linkable unless the PKG always generates the same public key for a given identity, for example by storing it after generating it for the first time. Such storage for unlinkability is not practical for IRMAseal because it requires the PKG to store a large and continuously increasing number of keypairs.

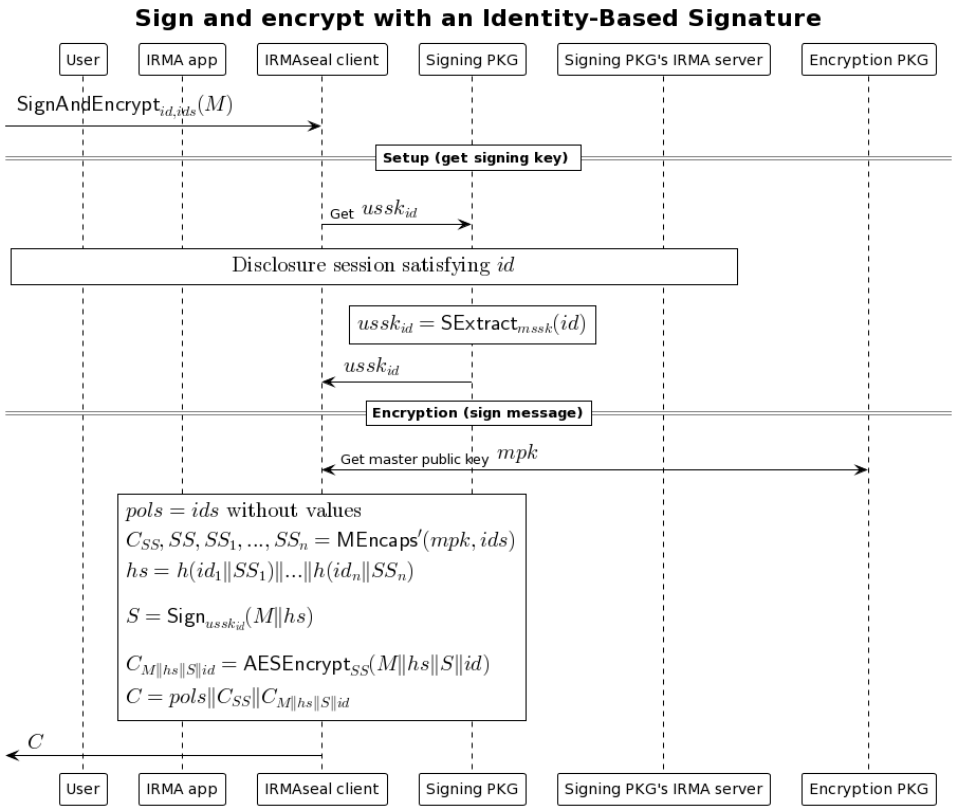

Figure 7.1: Signing and encrypting a message with an identity-based signature. Multiple messages can be signed with the signing key by repeating the "Encryption" part, as long as all messages are to be signed with the same IRMA attributes.

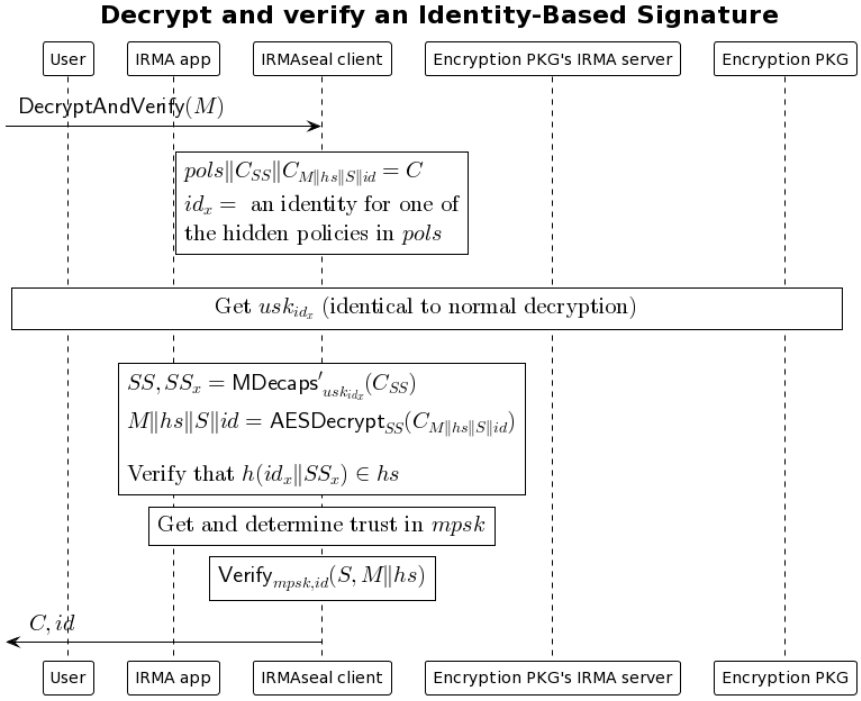

Figure 7.2: Decrypting and verifying a message with an identity-based signature.

For verification, one important step is that of getting *mpsk*. What this means exactly depends on the context. If only a single central signing PKG should exist, this *mpsk* could be known in advance or requested from that PKG. In a federated system, some mechanism needs to be in place to get *mpsk* and determine whether the corresponding PKG is trustworthy.

7.2 Analysis

In this scheme, there can be two distinct PKGs: one used for encryption, and one used for signing. These can be the very same entity (although the cryptographic functionality that needs to be offered for signing differs from the existing functionality for encryption), two different servers hosted by the same party, or entirely distinct. How the choice for both of these servers is made is out-of-scope for us, and can be left to other software that consumes IRMAseal. However, these choices are crucial for the trust a recipient can have in a signature, leading to some interesting security properties:

7.2.1 Security

Recipients need to be careful in which signing PKGs they trust: an adversary can create their own signing PKG, and sign for any identity they want, so

trusting *every* PKG is fatal. Instead, it could be useful to trust specific PKGs in a federated fashion, as is possible when selecting a PKG for encryption.

As an example, a university could host a signing PKG. It then makes sense for its employees to trust this PKG. A signer who wants to create a trustworthy signature for employees could use the university's PKG to make signatures that are likely to be trusted by university staff.

Apart from the complicated trust model, a scheme using IBS inherently suffers from some of the same problems as the approach using IRMA signatures on a public key, or in fact as any public key signature scheme: secret keys (in this case $ussk_{id}$) might leak, and users can intentionally leak them in an attempt to repudiate signatures they made earlier.

One more detail needs to be addressed: the signature as described above only guarantees that *someone with a certain identity* signed a message. It does not guarantee that the required attributes were valid (not expired) at the time of signing. The scheme can be easily extended to make guarantees about timing by (1) adding a timestamp into the identities indicating the time at which the disclosed attributes expire, and (2) adding a timestamp from a trusted timestamping authority on the message or signature. Such timestamps may however introduce additional linkability.

7.2.2 Privacy

By entrusting a third party with authentication, this scheme makes it possible to use any kind of identity. Hence, an IRMA ConDisCon can be used, hiding the actual attributes used to satisfy the identity, or even requiring only that an attribute is *not null*, without putting any requirement on the value itself. This makes the IBS approach more flexible privacy-wise.

The IBS-based approach does not expose any information about the message content but does require disclosing the sender's attributes to a PKG.

Important to note is that it depends on the IBS that is used whether the resulting signatures are linkable or not. Certification-based IBSs from the same signing key are linkable in the same way that signatures from public keys signed with an IRMA signature are.

7.2.3 Technical

The IBS-based approach requires implementing both an IBS scheme, as well as PKG functionality for that IBS, either as a separate server or an addition to the existing PKGs for encryption. A mechanism for the distribution of trust (to determine which signing PKGs are trusted) also needs to be implemented.

The overhead of the IBS-based approach in the message size depends entirely on the IBS scheme that is used. Using a simple certification-based

IBS, the signature could take 160 bytes, and like in the other approaches, another 32 bytes per recipient for the hashed identities (chapter 4).

Signing communication consists of a disclosure session with a signing PKG and its IRMA server. The resulting *ussk* can be reused afterward, although that does raise some concerns related to the validity of the disclosed attributes, and non-repudiation. If those concerns are mitigated with timestamps, perhaps additional communication with a TA is needed.

Verification only requires getting *mpsk* from the PKG used for signing. This public key can also potentially be kept offline.

Chapter 8

Signature from TTP

Any scheme for IRMAseal using Identity-Based Signatures requires getting a secret from a PKG in order to sign, which also implies that a PKG can forge signatures. Furthermore, the possibility of keeping and reusing this key for some time leads to some security issues, weakening the non-repudiation that is offered.

In this chapter, we present an alternative, in which signatures cannot be created by users themselves, but are instead requested from a TTP. This has the advantage that users can never claim to have lost their signing keys, providing good guarantees of non-repudiation.

Simply put, the scheme works as follows: the user requests a signature from the TTP on some message (hash), discloses some attributes to the TTP, and gets a signature. The TTP can use any underlying public key signature scheme for this. Verification is simply verification of the underlying signature using a public key of the TTP.

While this scheme can be instantiated with any public key signature scheme, we also show in Appendix A that it can use a signature scheme based on the IBKEM already used by IRMAseal's encryption PKGs. This means that a PKG could use the same master keypair for both encryption and to serve as a TTP for signatures.

8.1 Scheme

Using some public key signature scheme, again defined by the algorithms $\mathsf{PKSign}_{sk}(M)$ and $\mathsf{PKVerify}_{pk}(S, M)$, where (pk, sk) is a keypair generated by $\mathsf{PKKeyGen}()$, we define a scheme with signatures from a TTP as follows:

A user can request a signature on a hash $h(M\|hs\|r)$ (with hs and r as in chapter 4 and subsection 5.2.1) of a message M on behalf of an identity id (which can be, but needn't be limited to, a conjunction of IRMA attributes).[1] The TTP then asks the user to disclose attributes satisfying id and generates the signature $S = \mathsf{PKSign}_{sk}\big(h(M\|hs\|r)\|id\big)$. Here sk is a secret key of the TTP, which could be equal to the MSK of a PKG used for encryption if the signature scheme from Appendix A is used,

[1] hs could also be sent to the TTP separately instead of hashed again in $h(M\|hs\|r)$ but there is no advantage in doing so, so we opt for giving a TTP as little information as possible.

or a separate secret key for signatures. The TTP then sends the signature S to the user. Verification consists of retrieving pk (the MPK or a separate public key for signatures) from the TTP, and verifying S using $\mathsf{PKVerify}_{pk}\big(S, h(M\|hs\|r)\|id\big)$.

Figures 8.1 and 8.2 show the signing and verification procedures respectively. This scheme can easily be extended to include a timestamp of the time of signing: such a timestamp can be included in id, with the signing TTP acting as a witness. Again, the step for getting the signing TTP's pk uses some mechanism to get a public key and determine whether the corresponding TTP is trustworthy.

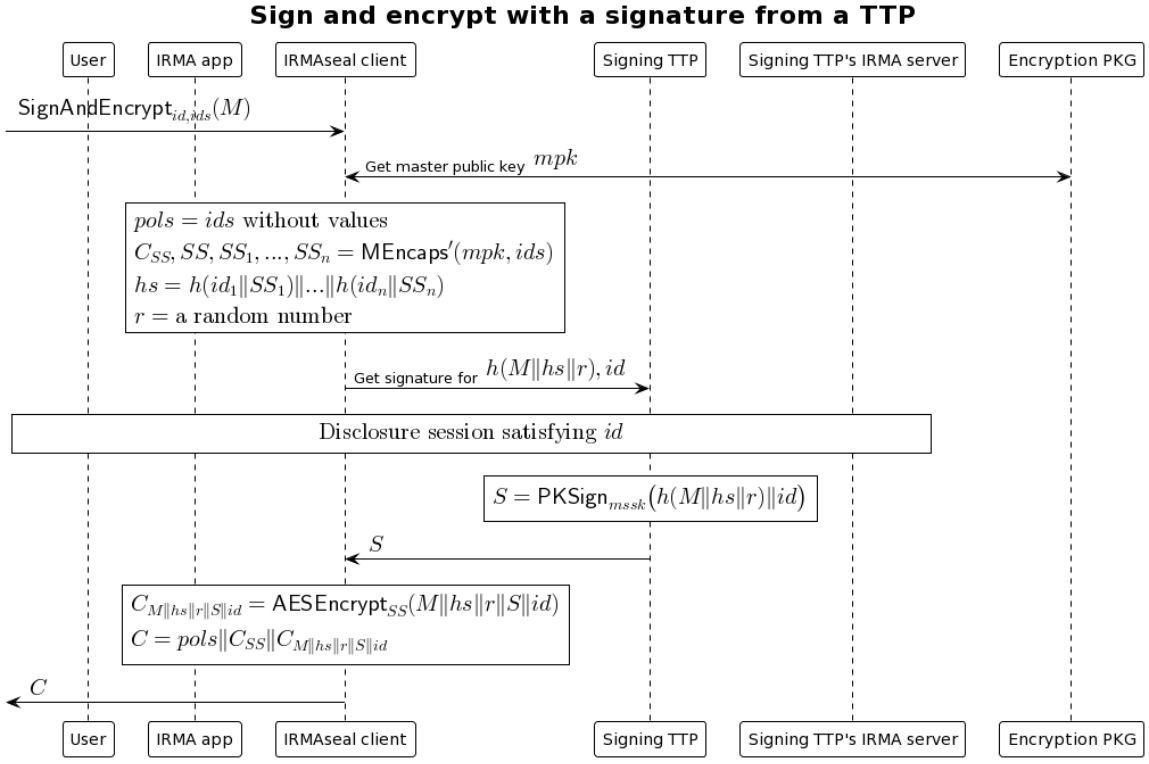

Figure 8.1: Signing and encrypting a message with a signature from a TTP.

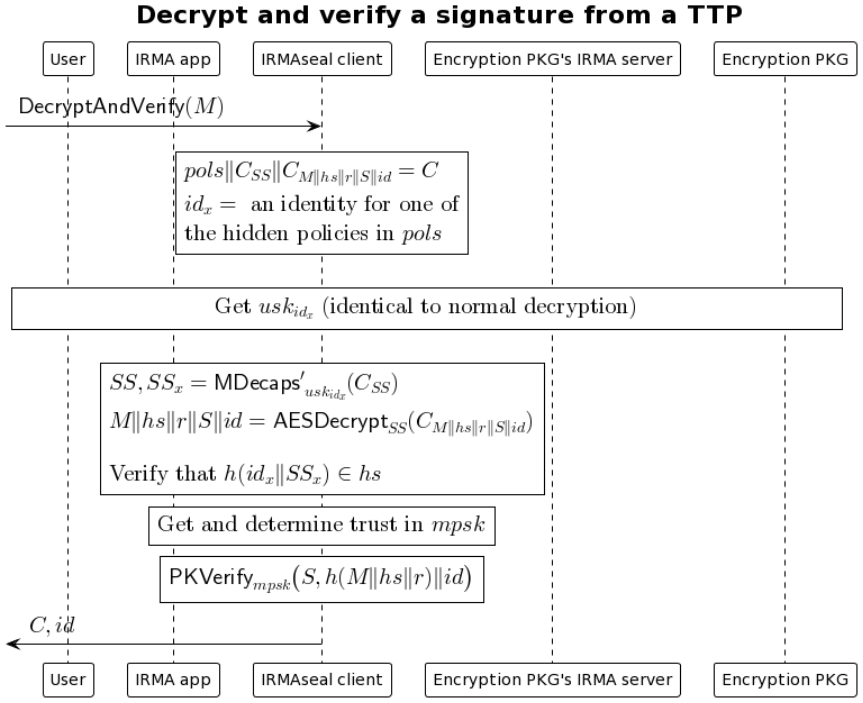

Figure 8.2: Decrypting and verifying a message signed with a signature from a TTP.

8.2 Analysis

As in the previous chapter, by entrusting a third party with authentication, this scheme makes it possible to use any kind of identity, at the cost of having to trust an additional party. The difficulty of determining which TTPs to trust remains unchanged. However, by not letting signing keys end up in the hands of users—at the cost of always requiring contacting a singing TTP in order to sign, rather than optionally reusing a *ussk*—this scheme directly provides non-repudiation[2]. Additionally, as a user needs to authenticate at the time of signing any message, a signature clearly indicates that the signer's attributes had neither expired nor been revoked at the time of signing. This is not straightforward in the IBS variant, where adding timestamps can only provide guarantees about the expiry of attributes, not about revocation.

From a technical perspective, the current approach also has some advantages. The scheme can be implemented with just a single and common public key signature scheme. There is no need to maintain local IRMA signature verification or an IBS. Signing TTPs will need to be developed, but

[2]given IRMA's assumption that users do not leak their own IRMA credentials

those can be quite simple, and it should be easy to make them stateless, except for having a keypair.

There are some concerns about the scalability of this solution, because every signature needs to be made by a TTP, whereas using an IBS, theoretically, many signatures can be made after a single session with a PKG. However, IRMAseal's scalability should not rely on the long-term reuse of signing keys for a number of reasons:

- Attributes can expire, so the potential lifetime of a signing key is limited anyway. Additionally, the 'older' a signing key is, the longer the timespan during which an attribute used to obtain it could have been revoked.
- Users may wish to sign different messages with many different combinations of attributes. Each signing identity requires a separate disclosure.
- Reused signing keys (for the *IBS* approach, this depends on the IBS that is used) can be linkable. This can be a privacy risk when keys are reused, which does not occur when keys are only used once or no signing keys are given to the user.

Furthermore, the scalability of signature creation will be less of a problem than that of the PKGs needed for decryption. For each IRMAseal decryption, a user needs to authenticate with a PKG already[3]. Under normal circumstances, the number of signatures created will always be lower than the number of decryptions, because each signed message is intended to be decrypted by *at least* one recipient.

Finally, the signing TTPs in this approach can be scaled quite well. The computational cost of creating a standard signature is negligible compared to the extraction of a USK for IBE, and a large number of separate master keypairs can be used for signing, and hence to scale signing TTPs over many separate machines without them having to share keys. This can be accomplished for example by setting up the trust in signing TTPs with certificate chains[4]: a user can trust a few root certificates, and a message can include a chain of certificates from a signing TTP to a root certificate. As a practical example, let's assume that the Dutch government wishes to provide signing TTPs to all dutch people. The Dutch government can then create a root keypair and publish its public key. Then, any number of TTPs can be hosted by the government by signing the separate public keys of each TTP, using the root keypair, and adding this signature to all of the signed messages. A verifying user then just needs to follow the chain of certificates and check if they trust the root.

[3]Technically, a USK may be valid for multiple messages, but in the current implementation of IRMAseal, a timestamp included in the identities for decryption limits this

[4]The same is possible for a certification-based IBS.

Chapter 9

Comparison

So far, four signature schemes have been proposed and analyzed in terms of security, privacy, and technical properties. We now proceed to compare the properties of these schemes, first considering security and privacy aspects, and then technical properties.

9.1 Security and Privacy

The four schemes all differ in the trust model behind the authenticity and integrity they offer, and in the parties that get to see the IRMA attributes of the signer. All four schemes succeed in fully protecting the confidentiality of the plaintext. Table 9.1 summarizes the parties that need to be trusted, as well as some other relevant notes, for each variant. IRMA is inherently a trusted party for authenticity and integrity in each scheme. Here, IRMA refers to a number of parts of the IRMA ecosystem: the issuers of the relevant attributes, the scheme manager (unless a fixed version of the scheme information is provided manually), and, in the case of IRMA signatures, the timestamp authority used for the signatures. Additionally, the IRMAseal client itself is assumed to be trusted. In section 11.3 we further discuss this assumption.

Likewise, the schemes differ in the degree to which they offer non-repudiation. In this regard, two categories can be distinguished:

- IRMA signatures on a public key and IBS, where it's plausible that key material can be leaked and used to create forgeries, and hence where a signer can claim they did not create a signature. In these cases, users create or get a signing key, that they can store to create multiple signatures for a longer period of time. Hence, it's possible to use such keys to create forgeries if they are leaked, and repudiation is possible by leaking a key on purpose.[1] Additionally, these options result in linkability of messages signed with the same signing keys unless an unlinkable IBS scheme is used in the *IBS* approach.

[1] While some techniques may help for non-repudiation, the common technique of trusted-timestamping and revocation is not practical for IRMAseal: revocation *requires* IRMAseal clients to keep track of the used signing keys, and that verifiers check somewhere whether a signature used a key that has been revoked, eliminating possibilities for offline verification.

Table 9.1: Parties that need to be trusted for authenticity (and integrity).

Scheme	Property	Trusted parties	Notes
IRMA ABS on message	Authenticity & Integrity	IRMA[1], IRMA servers	The IRMA servers used to create signatures can try tricking users into signing the wrong messages.
	Sender identity conf.	IRMA servers	
IRMA ABS on public key	Authenticity & Integrity	IRMA[1], IRMA servers	The IRMA servers used to create signatures can try tricking users into signing the wrong public key. Private keys can leak. If a signing key is used after the creation of an IRMA signature on it, the validity of the attributes is not certain.
	Sender identity conf.	IRMA servers	Signatures from the same private key are linkable.
Identity-Based Signature	Authenticity & Integrity	IRMA[1], signing PKG, signing PKG's IRMA server	Private keys can leak. If a signing key is used after the creation of an IRMA signature on it, the validity of the attributes is not certain. Recipients must be careful which signing PKGs they trust.
	Sender identity conf.	signing PKG, signing PKG's IRMA server	Attributes must be disclosed to a signing PKG that is trusted by the recipients.
Signature from TTP	Authenticity & Integrity	IRMA[1], signing TTP, signing TTP's IRMA server	Recipients must be careful which signing TTPs they trust.
	Sender identity conf.	signing TTP, signing TTP's IRMA server	Attributes must be disclosed to a signing TTP that is trusted by the recipients.

[1]The IRMA infrastructure (the relevant issuers and scheme manager) inherently needs to be trusted for all options.

- IRMA signatures on a message and signatures from a TTP, where users do not get key material that can be reused to create more signatures.[2] Since it is not plausible that keys are leaked in these approaches, they provide non-repudiation in a much stronger sense than the other two approaches.

A similar division can be made based on the parties that need to be

[2]Leaking the credentials in the IRMA app itself is possible, but this is inherent to IRMA. Furthermore, leaking IRMA credentials gives others full access to authenticate and sign with all attributes of the leaked credential, so it's unattractive to leak them just to repudiate a signature. Hence, in IRMA it's usually assumed that users do not do this.

trusted for authenticity and integrity. There are the IRMA signature-based approaches that only require trusting IRMA itself (the relevant issuers, scheme manager and TA, and to some degree an IRMA server used while signing), and the two schemes that use additional trusted parties. The latter requires a recipient to trust a signing TTP or PKG on top of the IRMA ecosystem. In a federated setting, this means that some mechanism for distributing trust in the signing TTPs or PKGs needs to be in place.

This split coincides with the possibility to use something else than a conjunction of IRMA attributes as signing identity, because the additional trusted party can translate disclosed attributes into any kind of identity. Likewise, the approaches with additional TTPs can also be applied to IBE systems that do not use IRMA to authenticate identities; they can actually be instantiated with any other authentication system.

In summary, the IRMA signature-based approaches require fewer trusted parties[3], and hence don't need a mechanism for distributing trust in signing TTPs or PKGs. However, they are tightly coupled to IRMA and less flexible in terms of the signing identity. The approaches that offer to sign multiple messages after using IRMA only once, do so at the cost of non-repudiation, and possibly at the cost of unlinkability.

9.2 Technical

Other differences exist in technical aspects. The solutions require different software to be implemented, which matters for the ease of implementation and maintainability, as well as the security risks associated with complexity and dependencies. The things that need to be implemented and maintained for each solution are indicated in Table 9.2. This table also shows an estimation of the signature size that needs to be sent along with a message (assuming a simple signing identity and a single recipient). Finally, the table also indicates whether a way to distribute trust in third parties is required in a federated setting. Such a mechanism might depend on the software consuming IRMAseal, but is still a significant aspect that would need to be implemented securely, and could be quite complex.

From the implementation requirements, we can see that the first and the last approach are simpler than the other two, because they use either IRMA signatures or standard signatures directly, and no other cryptography. This is directly related to the inability to sign multiple messages at once, and the offered non-repudiation: not giving the IRMAseal client secret key material makes the client simple, prevents leaking the key material, but also makes signing multiple messages impossible.

It is hard to compare the IRMA signature-based variants with the others because they require totally different components to be developed and

[3]However, the IRMA server used to create signatures is in a position that might be abused if it can trick users into signing a wrong hash.

Table 9.2: Implementation requirements

	Implementation requirements						Properties			
Scheme	Modify MKEM	IRMA signature creation	Local IRMA signature verification	Existing public-key cryptography	New crypto	Signing TTP/PKG	Supports multiple signatures per IRMA session	Requires mechanism for distribution of trust[2]	Tightly coupled to IRMA	Estimated signature size (for 1 recipient)
IRMA ABS on message	×	×	×						Yes	3 KB
IRMA ABS on public key	×	×	×	×			Yes		Yes	3 KB
Identity-Based Signature	×			?[1]	?[1]	×	Yes	Yes		?[1]
Signature from TTP	×			×		×		Yes		< 1 KB

[1] Depends on the IBS scheme.

[2] In a federated setting.

available. The IRMA signature-based versions can let the cryptography be handled by existing software but do require an IRMA server to be running, as well as maintaining software for local IRMA signature verification. On the other hand, the other two approaches require implementing a TTP or PKG for signing, and hence, if used in a federated fashion, they require a mechanism for determining trust in signing TTPs or PKGs. This mechanism for trust is a challenge in itself.

Some other properties of the solutions are also worth mentioning. First, the IRMA signature-based solutions are of course tightly coupled to IRMA. Apart from limiting the flexibility of signing identities (i.e. only a conjunction of IRMA attributes is supported), this also means that they are not applicable to systems similar to IRMAseal that use some other identification scheme.

Second, the IRMA signatures appear to be quite large. It is interesting to see that using *Signature from TTP* much smaller signatures can be made than with IRMA signatures. Verification of a signature from a TTP is also much simpler (so probably faster) than that of IRMA signatures.

Finally, the schemes differ in the information and parties that need to be available for signing and verification. For signing, the difference is simple and not very important: each version needs the IRMA scheme information (that is normally distributed by a scheme manager, but can be provided offline) to be available, the IRMA signature versions also need an IRMA server and a specific timestamp authority to be online, and the other two

versions need a signing PKG/TTP to be online. On a side note, using *IRMA signature on public key* or *IBS*, signatures can still be made despite the IRMA infrastructure and/or signing PKG being unavailable if the user already has a signing key for the right identity.

For verification, all 4 proposed schemes support verification as long as some information is available: for the IRMA signature-based schemes, verification can be done as long as the IRMA scheme information is available. This is a collection of files that can be stored offline. Similarly, *IBS* and *Signature from TTP* only require the public keys of the trusted PKGs or TTPs to be available. These, too, can in principle be stored offline, depending on the mechanism for the distribution of trust[4].

[4]For example, a certificate chain can be used by including a chain of certificates on the public keys of a signing PKG or TTP in each signed message. The clients then only need to know a single root public key to verify signatures from a great number of different PKGs or TTPs

Chapter 10

Related Work

In 1984, Shamir presented the concept of identity-based cryptography. He both presented a functional IBS scheme and proposed the idea of IBE, posing an open question for IBE implementations. While several IBE schemes have been proposed in the following years, the first practical scheme was presented in 2001 by Boneh and Franklin [4]. Since then, many improved schemes have been proposed, with increasing security and performance.

Originally, the use of IBE was to take a single identifying string as identity, such as an email address. Nonetheless, it wasn't long before it became clear that additional information can be embedded within an identity. For instance, an email address could be combined with a date (e.g. `"bob@example.com|2030-01-01"`) to send messages 'into the future', with a PKG only supplying the decryption keys when the included date has passed. Sahai and Waters [27] first generalized this idea, coining the term "Attribute-Based Encryption" (ABE), explicitly using a set of attributes with some access policy as identity. Along with the term ABE, they proposed an instance of ABE, called "Fuzzy Identity-Based Encryption", where recipients need to have *approximately* the right attributes. That is, recipients need to have *most* of the specified attributes, but needn't have all. This approach is aimed mainly at biometric identities, where each measurement is likely to differ somewhat.

IBE has been used in practice for email encryption, for instance in *FortiMail IBE*[1] and *Voltage SecureMail*[2]. Other proposed applications of identity-based cryptography include a repudiable alternative to DKIM (DomainKeys Identified Mail) [31] and a network layer security protocol [29].

PostGuard, which was originally also called IRMAseal (the email-specific application using IRMAseal has been renamed) combines email encryption based on IBE, with the attribute-based credential system IRMA. While IRMAseal and PostGuard have not been proposed in a published work at the time of writing, several Bachelor's theses [16, 32] at Radboud University investigated usability aspects of PostGuard. A usability study is currently being performed into signatures in PostGuard, but at the time of writing, no technical research has been done into signatures in IRMAseal.

Another relevant work proposing an encryption system for email using

[1] `https://www.fortinet.com/products/email-security`

[2] `https://www.microfocus.com/en-us/cyberres/data-privacy-protection/secure-mail`

IBE is [2] by Adida et al. This work is based on an earlier publication [1] from mostly the same authors, which proposes a signature scheme for email, using identity-based signatures with DNS as a means to distribute the master public keys per domain. Such a technique can also be considered to distribute master public keys for IRMAseal encryption, as well as public keys of trusted signing TTPs or PKGs for IRMAseal signatures in the *IBS* and *Signature from TTP* approaches we investigate in this thesis.

The literature around identity-based signatures is summarized in [20], describing categories of IBS schemes. Since IBS can be achieved with a simple certification approach, most literature about it aims at improved performance, not better security properties.

The two most known protocols used for signing messages, similar to what we try to achieve in this paper, are S/MIME and OpenPGP [28, 8]. Like IRMAseal, OpenPGP can be used apart from email. These end-to-end encryption and signature standards have in common that they need some way to store private keys, and distribute public keys. For OpenPGP, distribution of keys is usually done through a web of trust with some support from public key servers. S/MIME takes a more transparant approach, using certificate authorities. Related is DKIM (DomainKeys Identified Mail) [11], a signature scheme for signing of emails (usually) by the outgoing email server (typically not by the end user), which—like [1]—uses DNS to distribute public keys per server.

In our study, we find that unless an approach using IRMA signatures (with IRMA's own public key infrastructure used to distribute trust) is implemented, IRMAseal will also require some mechanism to distribute trust (and public keys of signing PKGs or TTPs).

[23] describes the problem of needing trust in some third party for signatures. The view that it's imperfect to trust any single third party (such as a certificate authority but also a PKG) is taken a bit further by Landrock and Pedersen in [22]. They point out the channel between a user and the 'signature processor' as a vulnerable area, introducing the What You See Is What You Sign (WYSIWYS) property. This property requires certainty that the user (not just a piece of software running on the user's device) is aware of what exactly is being signed. From this point of view, it becomes explicit that the software implementing S/MIME and OpenPGP needs to be trusted.

Taking the same perspective on DKIM, where typically the outgoing mail server is the *signature processor*, we can see that the channel between a user and the signature processor is significantly longer. Likewise, this perspective offers some useful insights for our work on IRMAseal, which we discuss further in section 11.3.

Chapter 11

Discussion

In this chapter, we first present our main findings in section 11.1 and 11.2. We then continue discussing related topics, limitations, and suggestions in section 11.3 through 11.8. Finally, we conclude in section 11.9.

11.1 IRMA signatures or TTPs

In this thesis, we have investigated 4 different approaches to provide authenticity, integrity and non-repudiation in IRMAseal.

Comparing the solutions, we have seen that two categories can be distinguished:

- The options based on IRMA signatures, that are strictly bound to using IRMA (and conjunctions of IRMA attributes) for authentication, but do not require trust in an additional third party[1].
- The options that rely on additional third parties (signing PKGs or signing TTPs). These variants are not inherently limited to IRMA, and hence support a wider range of (potentially more privacy-friendly) identities, and can be seen as more general approaches, applicable outside of IRMAseal. However, for these solutions, it is crucial that a good mechanism for the distribution of trust is implemented.

We have not found a decisive argument to prefer one of these categories over the other. A choice between the two will mainly require weighing the importance of using a generic, flexible solution (i.e. not strictly bound to IRMA) against the difficulties of finding a good way to distribute trust.

At first glance, requiring trust in another third party seems problematic. Nevertheless, [23] note that this problem is actually present in nearly all signature and authentication protocols. Many protocols are based on trust in one or more Certificate Authorities (CAs), which are essentially the same as signing PKGs. Even the schemes based on IRMA signatures similarly require trust in a number of third parties (most notably the IRMA issuers). Even though trusting third parties for confidentiality is often seen as a problem (it is one of the most important downsides of IRMAseal as a whole), it is often accepted for authenticity.

[1] Trust is still needed in the IRMA ecosystem and to some degree in an IRMA server by the signer's choice used when creating signatures.

11.2 Signing keys

Perpendicular to the categorization above, an argument can be made against the two solutions (*IRMA signature on public key* and *Identity-Based Signatures*) that allow signing multiple messages after a single IRMA session. Giving the IRMAseal client access to key material that can be used to create signatures opens up a possibility for that key material to leak, both intentionally by the user (thus non-repudiation is inevitably lost, at least to some degree) and unintentionally, e.g. through side-channels or compromised machines (which affects the trust that recipients can have in a signed message). These are issues that are conveniently avoided by the other two variants. The possibility that key material can leak also occurs in alternatives to IRMAseal signatures such as OpenPGP or S/MIME. However, OpenPGP at least offers an effective system for revocation. This is not practical in IRMAseal (because revocation is only possible as long as private keys are available to the user), which makes leaking keys a more serious problem for IRMAseal than it is for OpenPGP.

11.3 WYSIWYS

In all four discussed schemes, the IRMAseal client must be trusted, because the client can make signatures on behalf of the user, without their permission. This violates the WYSIWYS (What You See Is What You Sign) principle [22] if we don't consider the IRMAseal client fully trusted.

This is inherently the case if the IRMAseal client gets access to signing keys: it can then reuse the keys at will, without any user interaction. However, in the two schemes that do not expose keys to the IRMAseal client, the client can still trick users into signing a wrong message:

- In *IRMA signature on message*, the user creates a signature on a hash, which is necessary in order to not expose the plaintext to the IRMA server that is used to create the signature through. While the user can compare the hash displayed by the IRMA app with that shown by the IRMAseal client to prevent the intermediary IRMA server from modifying it, the user cannot be sure that the IRMAseal client is actually presenting the hash that the user intends to sign.
- In *Signature from TTP*, the user does not communicate directly with the TTP: the IRMAseal client asks the TTP to sign a message, and then presents the user with a QR code or URL for an IRMA disclosure session. The user then performs this disclosure, but cannot verify what message (hash) the IRMAseal client presented to the TTP. Even if the TTP can show the hash that is to be signed directly to the user, this is not legible by the user, because it is a hash and not the plaintext.

Having to trust the client—while not surprising, and also required in alternatives to IRMAseal signatures such as OpenPGP or S/MIME—is a security disadvantage that would be nice to avoid. This security issue is a limitation of all 4 proposed schemes.

11.4 IRMA signature creation without a trusted intermediary

One of the main challenges and security risks in the IRMA signature-based schemes—and the reason why in *IRMA signature on message* the IRMAseal client could trick users into signing wrong messages—is that for IRMA signature creation an IRMA server is needed to communicate between the IRMAseal client and the IRMA app. This IRMA server can read and modify all of the traffic between the parties, even though it essentially has no function other than to relay messages. If no fully trusted IRMA server exists (it's not always feasible to run one yourself, especially from within a browser environment), it hence becomes necessary to hide the plaintext to be signed from this IRMA server, as done in subsection 5.2.1. Even so, the IRMA server gets to observe the used attributes and can modify traffic, potentially tricking the user into signing a different message than the one they intended to sign.

If a secure channel could be established between the IRMAseal client and IRMA app, the measures to hide the plaintext from the IRMA server would not be necessary, which in turn allows us to let the IRMA app present a legible message to the user that is signing. This benefits the user experience and better ensures that users are aware of what they are signing (even if a malicious IRMAseal client tries to trick them into signing something else).

The possibility for IRMA to support creating signatures without interacting with an IRMA server, or by establishing a secure channel *through* (not legible/modifiable by) an IRMA server, would significantly improve the security and privacy offered by the IRMA signature-based schemes. Apart from signature schemes for IRMAseal, this possibility seems equally vital for many other applications of IRMA signatures in general. Hence, further research into this possibility seems valuable to us.

11.5 IRMA signature creation on illegible data

An issue related to what is described above is the user experience of creating IRMA signatures on a hash. The steps to safeguard the plaintext confidentiality in subsection 5.2.1 require that we sign a hash instead of a human-readable string. Because the IRMA app can currently only sign and display strings, the same problem of having to display a hash also occurs in many other applications of IRMA signatures, such as signing images, PDFs,

or other binary files.

While signing (and showing to the user) a hash technically works, it is not ideal for the user experience, nor for security: looking at a hash, the user cannot be sure what they are signing. While users can be instructed to compare a hash in the IRMA app with one shown by e.g. the IRMAseal client, this is not user-friendly, and only works to prevent an IRMA server from modifying the message that is signed. They cannot know with certainty that whatever software is presenting a hash to compare is not itself tricking the users.

Hence, signing a hash (in the IRMA signature-based schemes for IRMAseal but also in other applications of IRMA signatures) actually violates the WYSIWYS principle: the requesting software (e.g. IRMAseal client) is in a position to change the message that is signed without the user's knowledge.

Even though we don't see a straightforward solution to avoid this issue for all kinds of illegible data, the IRMA app can probably be modified to allow displaying at least some common types of documents, such as images and PDFs.

11.6 Signing multiple *existing* messages at once

In section 11.2 we have seen that there are some good arguments against the possibility of signing multiple messages after a single IRMA session: giving the IRMAseal client keys that can be used later comes with security issues related to both intentional and unintentional leaking of keys. However, there is a use case in which these arguments do not apply: when all messages to be signed exist *before* the IRMA session takes place. That is, a single IRMA session could be used to sign multiple *existing* messages, without enabling the later creation of signatures on messages that did not exist at the time of the IRMA session. In this case, because there is no key material that can be reused later, the problems with the leaking of keys do not occur.

While we have not investigated an option that allows signing multiple *existing* messages only in this thesis, we expect that rather simple extensions are possible to both the *IRMA signature on message*[2] and *Signature from TTP*[3] approaches, to support signing multiple messages at once without the possibility to use the same key material later.

We also imagine modifications of *IRMA signature on public key* and *Identity-Based Signatures* to be possible, using a timestamp at the time of

[2]For instance, a single IRMA signature could be made on multiple hashes of messages at once, and used as a signature on each of those messages (although this would leak information about the number of messages signed at once to the recipients).

[3]A TTP can simply return multiple signatures at once after a single disclosure satisfying the identities used for all of the messages. Here, it is not even required that all of the messages use the same attributes because a TTP can include only a subset of the disclosed attributes in the signature for each message.

creation in the IRMA signature or identity, and requiring trusted timestamps from before that time of creation to exist on every message to be signed. Such a timestamp would then ensure that the key material is only usable on messages that existed before the IRMA session.

Exploring these extensions in detail is left for future work.

11.7 Repudiability

Considering only schemes aiming to offer non-repudiation, we have not investigated options that provide *repudiability*[4], which, as discussed by [1], can be more privacy-friendly. This may be an interesting area for further research.

11.8 Alternative solutions to the naive sign-and-encrypt problem

In chapter 4 we have presented a technique to solve the naive sign-and-encrypt problem, by binding a signature to the identities of recipients. However, as pointed out in section 4.1, this technique may not be the only or best way to solve the problem. A promising alternative is the sign-encrypt-sign technique proposed in section 5.2 of [13]. Working this out in detail is left for future work.

11.9 Conclusions

In this thesis we have defined and analyzed four signature schemes for IRMAseal, aiming to provide attribute-based signatures for authenticity, integrity, and non-repudiation of messages encrypted with IRMAseal. Furthermore, we have proposed a technique to prevent the naive sign-and-encrypt problem from [13] in all of these schemes. Our main findings are:

1. that the schemes that are based on IRMA signatures do not require additional trusted parties, whereas the others need PKGs or other TTPs with some mechanism to distribute trust (11.1)

2. that schemes that give the IRMAseal client access to key material—allowing it to create multiple signatures later, without a separate IRMA session—come with security limitations (mainly for non-repudiation), directly because of the access to key material (11.2).

We have found no decisive argument between the *IRMA signature on message* and *Signature from TTP* schemes.

[4]The ability to deny having signed a message. This can co-exist with authenticity if recipients are convinced of, but cannot publicly prove, who created a signature.

Moreover, we identified some limitations with the creation of IRMA's attribute-based signatures, and more options that could be explored for IRMAseal signatures which we have not yet investigated in detail. These limitations and aspects for future work are:

1. All of our four options require trust in the IRMAseal client (11.3). A scheme that does not assume this is left for future work.

2. IRMA signatures need to be created via an IRMA server, which can read the message that is signed (11.4). A modification to IRMA to eliminate this intermediary is an interesting avenue for future work.

3. IRMA signatures can only be made on strings, leading to a poor user experience and reduced security when signing data that is not human-readable (11.5).

4. We expect that it is possible to extend the *IRMA signature on message* and *Signature from TTP* schemes to allow signing multiple *existing* messages at once, without downsides we found for our schemes that allow signing multiple messages later (11.6). Investigating this possibility in detail is left for future work.

5. The four proposed schemes are aimed at providing non-repudiation. We have not investigated options that provide repudiability, which may be an interesting area for further research (11.7).

6. The sign-encrypt-sign technique from [13] to solve the naive sign-and-encrypt problem may be a better alternative to the technique we have proposed (11.8). Working this out in detail is left for future work.

Bibliography

[1] Ben Adida, David Chau, Susan Hohenberger, and Ronald L Rivest. Lightweight signatures for email. In *Proceedings of the DIMACS Workshop on Theft in E-Commerce*. Citeseer, 2005.

[2] Ben Adida, Susan Hohenberger, and Ronald L. Rivest. Lightweight encryption for email. In Dina Katabi and Balachander Krishnamurthy, editors, *Steps to Reducing Unwanted Traffic on the Internet Workshop, SRUTI'05, Cambridge, MA, USA, July 7, 2005*. USENIX Association, 2005.

[3] Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Security proofs for identity-based identification and signature schemes. *Journal of Cryptology*, 22(1):1–61, 2009.

[4] Dan Boneh and Matt Franklin. Identity-Based Encryption from the Weil Pairing. In *Advances in Cryptology — CRYPTO 2001*, pages 213–229, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[5] Privacy by Design Foundation. IRMA docs: Revocation. `https://irma.app/docs/revocation/`. v0.11.0, accessed: 2022-12-03.

[6] Privacy by Design Foundation. IRMA docs: Technical overview. `https://irma.app/docs/overview/`. v0.10.0, accessed: 2022-10-02.

[7] Privacy by Design Foundation. IRMA docs: Zero-knowledge proofs. `https://irma.app/docs/zkp/`. v0.10.0, accessed: 2022-10-05.

[8] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. OpenPGP Message Format. RFC 4880, RFC Editor, November 2007.

[9] Jan Camenisch et al. Specification of the identity mixer cryptographic library. *IBM Research — Zurich*, pages 1–48, 2010.

[10] Jie Chen, Romain Gay, and Hoeteck Wee. Improved Dual System ABE in Prime-Order Groups via Predicate Encodings. In *Advances in Cryptology - EUROCRYPT 2015*, pages 595–624, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[11] D. Crocker, T. Hansen, and M. Kucherawy. DomainKeys Identified Mail (DKIM) Signatures. Internet Requests for Comments, September 2011.

[12] Joan Daemen and Vincent Rijmen. *The design of Rijndael: The Advanced Encryption Standard (AES)*. Springer-Verlag, 2002.

[13] Don Davis. Defective Sign & Encrypt in S/MIME, PKCS#7, MOSS, PEM, PGP, and XML. In *Proceedings of the General Track: 2001 USENIX Annual Technical Conference, June 25-30, 2001, Boston, Massachusetts, USA*, pages 65–78. USENIX, 2001.

[14] Amos Fiat and Adi Shamir. How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology — CRYPTO' 86*, pages 186–194, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.

[15] Craig Gentry and Alice Silverberg. Hierarchical ID-Based Cryptography. In *Advances in Cryptology — ASIACRYPT 2002*, pages 548–566, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[16] Quoc An Ha, Hanna Schraffenberger, and Bart Jacobs. Investigating Usability Problems in Email Encryption Tools Based on IBE, 2022.

[17] Brinda Hampiholi, Gergely Alpár, Fabian van den Broek, and Bart Jacobs. Towards Practical Attribute-Based Signatures. In *Security, Privacy, and Applied Cryptography Engineering*, pages 310–328, Cham, 2015. Springer International Publishing.

[18] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A Modular Analysis of the Fujisaki-Okamoto Transformation. In *Theory of Cryptography*, pages 341–371, Cham, 2017. Springer International Publishing.

[19] Cameron F. Kerry and Patrick D. Gallagher. Digital Signature Standard (DSS). *FIPS PUB 186–4*, 2013.

[20] Eike Kiltz and Gregory Neven. Identity-Based Signatures. *Identity-based cryptography*, 2:31–44, 2008.

[21] Eike Kiltz and Yevgeniy Vahlis. CCA2 Secure IBE: Standard Model Efficiency through Authenticated Symmetric Encryption. In *Topics in Cryptology – CT-RSA 2008*, pages 221–238, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[22] Peter Landrock and Torben Pedersen. WYSIWYS? — What you see is what you sign? *Information Security Technical Report*, 3(2):55–61, 1998.

[23] Albert Levi and M. Ufuk Çağlayan. The problem of trusted third party in authentication and digital signature protocols. In *Proceedings of the 12th International Symposium on Computer and Information Sciences*, pages 317–324, 1997.

[24] Daniel Ostkamp. IRMAseal signatures. Unpublished notes, 2022.

[25] Adrian Reuter, Karima Boudaoud, Marco Winckler, Ahmed Abdelmaksoud, and Wadie Lemrazzeq. Secure Email - A Usability Study. In *Financial Cryptography and Data Security*, pages 36–46, Cham, 2020. Springer International Publishing.

[26] R. L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM*, 21(2):120–126, feb 1978.

[27] Amit Sahai and Brent Waters. Fuzzy Identity-Based Encryption. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, pages 457–473, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[28] J. Schaad, B. Ramsdell, and S. Turner. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Message Specification. RFC 8551, RFC Editor, April 2019.

[29] Christian Schridde, Matthew Smith, and Bernd Freisleben. An Identity-Based Key Agreement Protocol for the Network Layer. In Rafail Ostrovsky, Roberto De Prisco, and Ivan Visconti, editors, *Security and Cryptography for Networks*, pages 409–422, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[30] Svenja Schröder, Markus Huber, David Wind, and Christoph Rottermanner. When SIGNAL hits the fan: On the usability and security of state-of-the-art secure mobile messaging. In *European Workshop on Usable Security. IEEE*, pages 1–7, 2016.

[31] Michael A. Specter, Sunoo Park, and Matthew Green. KeyForge: Non-Attributable Email from Forward-Forgeable Signatures. In Michael Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 1755–1773. USENIX Association, 2021.

[32] Niels Starren, Hanna Schraffenberger, and Bart Jacobs. Johnny can Encrypt? A Usability Study of IRMAseal, 2022.

[33] Christian Stransky, Oliver Wiese, Volker Roth, Yasemin Acar, and Sascha Fahl. 27 Years and 81 Million Opportunities Later: Investigating the Use of Email Encryption for an Entire University. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 860–875. IEEE, 2022.

Appendix A

Signatures from IBKEM

An observation attributed to Moni Naor by Boneh and Franklin (Section 6 in [4]) is that any IBE scheme can be transformed into a public-key signature scheme, where a signature on a message M is the user secret key for identity M (usk_M or $\mathsf{Extract}_{msk}(M)$). Verification of a signature S on M is done by generating a random challenge c, and verifying that $\mathsf{Decrypt}_S\big(\mathsf{Encrypt}_{mpk,M}(c)\big) = c$. This is effectively a public key signature scheme where the public key is the master public key mpk and the private key is the master secret key msk.

We expect without a formal proof that the transformation from an IBE to a signature scheme can also be applied to an IBKEM as follows:

Just as in the original transformation, we sign using an IBKEM with $S = \mathsf{Extract}_{msk}(M)$. Instead of verifying by encrypting and decrypting a challenge, we can encapsulate a shared secret, yielding as challenge c a ciphertext that can be decrypted with the signature, and as expected response r the shared secret that has been encapsulated: $c, r = \mathsf{Encaps}_{mpk}(M)$. We then decapsulate the challenge with the signature and verify that the result is correct: $\mathsf{Decaps}_S(c) = r$.

Intuitively, it follows from the idea that decapsulation can only be done using the correct secret key that the above verification succeeds if and only if the signature is indeed the user secret key for the identity M.