Towards Practical Attribute-Based Signatures

Brinda Hampiholi, Gergely Alpár, Fabian van den Broek, and Bart Jacobs

Institute for Computing and Information Sciences,
Radboud University, Nijmegen, The Netherlands
Email: {brinda, gergely, f.vandenbroek, bart}@cs.ru.nl

Abstract. An attribute-based signature (ABS) is a special digital signature created using a dynamic set of issued attributes. For instance, a doctor can sign a medical statement with his name, medical license number and medical speciality. These attributes can be verified along with the signature by any verifier with the correct public keys of the respective attribute issuers. This functionality not only makes ABS a much more flexible alternative to the standard PKI-based signatures, but also offers the ability to create privacy-preserving signatures. However, none of the ABS constructions presented in the literature is practical or easily realizable. In fact, to the best of our knowledge, there is currently no ABS implementation used in practice anywhere. This is why we put forward a new ABS technique based on the IRMA attribute-based authentication. IRMA already has an efficient and practical smart-card implementation, and an experimental smart-phone implementation too. They are currently used in several pilot projects.
In this paper, we propose an ABS scheme based on the existing IRMA technology, extending the currently available IRMA devices with ABS functionality. We study the practical issues that arise due to the introduction of the signature functionality to an existing attribute-based authentication scheme, and we propose possible cryptographic and infrastructural solutions. We also discuss use cases and implementation aspects.

Keywords: attribute-based signature, attribute-based credential, IRMA, authentication, timestamp, contextual privacy.

1 Introduction

Digital signatures are cryptographic primitives that are used by a person to digitally sign a message, thus declaring that he agrees with the message. The digital signature standard based on asymmetric cryptography requires a signer to sign with his private key and the corresponding public key is used for verifying this signature. The public key of the signer stated in the public-key certificate identifies the signer and links all the messages that he ever signed. This type of digital signature does not allow the signer to make the context or role of the signer explicit, and this limits the cases in which the signer can sign under a particular set of attributes and reveal nothing else about himself.

In the case of a medical statement signed by a doctor, the doctor's attributes such as his qualification, speciality and license number (for accountability purpose) are necessary along with his signature. With the current public-key infrastructure (PKI) based signature, a doctor has to reveal his full identity, public-key certificate every time he digitally signs a document. This results in the unnecessary (or undesired) disclosure of excess data about the doctor's identity and all his signatures will always be linkable to him irrespective of the context. This violates the principle of data minimization and harms the privacy of the signer.

Furthermore, a PKI-based signature provides the signer identity but the identity itself does not say much about the individual attributes of the signer that are relevant from the perspective of a signature verifier. Such a signature would just state that *"This message is signed by a signer with common name 'Jack' holding public key 'x' as attested in the corresponding public key certificate signed by CA 'y'."*. Considering the above example, the verifier does not know if the message was indeed signed by a doctor whose speciality is orthopaedics; he just knows that the message was signed by Jack holding public key x. This issue can be solved if the signer could sign under a set of attributes (*e.g. 'doctor', 'orthopaedics'*) specified by the verifier's signature policy.

Attribute-based signatures (ABSs) allow a person to sign under a set of selected authentic/certified attributes based on the context. The signature reveals no more than the fact that a signer, with a specific set of attributes satisfying a certain condition, has attested the message. Here, the signature proves that the attributes hold for him at the time of signing and the signature is generated using a secret key associated with the signer. The signature verification will fail if the message was changed after signing, which ensures the integrity of the signed message. In the case of an ABS, non-repudiation of the signer can be achieved by enforcing the disclosure of signer-identifying attributes in the signature policy (*e.g.* the attribute to be revealed may be *'is a doctor with medical license number 12345'*). However, when the disclosed attributes are non-identifying such as *'age* $\geq$ *18'* or *'is a doctor'*, the verifier cannot link the signed message to the real identity of the signer solely based on the signature, like for group signatures [1]. Some use cases in which privacy of the signer is essential such as anonymous voting, anonymous petitions *etc.*, can thus also benefit from ABSs.

Role-based signature generation with ABSs. A traditional digital signature seems to offer the same signing functionality as an ABS (to a limited extent), if it dedicates one signing key pair for *each* role under which a signer wishes to sign. It means that the signer who is a doctor, for instance, can generate a key pair, get a medical certification authority sign his public key and use the corresponding private key to sign *as a doctor*. Note, however, that he needs to generate another key pair and get the public key certified from the national government to be able to sign his tax declaration form *as a citizen*. Another example is the Belgian PKI-based national eID card, which allows the cardholders to perform digital signature function but always *as a particular Belgian citizen*. In principle, such a card would require n signing keys for n roles that a cardholder might rightfully assume while signing. This gives rise to complicated key management issues. In contrast, an ABS allows the signer to digitally sign messages or documents under different roles with a *single* signing key. Role-based signatures based on ABSs also make the role of the signer explicit to the verifiers thus, avoiding confusion. Consider the two instances when a notary handles the sale of his client's property and the sale of his own private property. With ABSs, he can sign *as a notary* in the former instance and *as the owner of a property* in the latter instance. The difference in the signer's roles is not apparent to the verifiers when the notary signs both sale documents with PKI-based signatures whereas ABS clearly states the role of the signer in each of his signatures. In sum, ABSs are a generalization of role-based signatures with possibly additional privacy guarantees.

Related work. Attribute-based signatures (ABSs) have been explicitly introduced by Shaniqng and Yingpei [2] rethinking attribute-based encryption. Maji *et al.* [3] proposed an ABS scheme in which attributes belonging to a user are represented as a *credential bundle*. They employed a non-interactive proof of knowledge system to prove the knowledge of a credential bundle that satisfies a given access formula. Okamoto et al. [4] propose a decentralized multi-authority ABS which supports non-monotone predicates and prove it to be fully secure in the random-oracle model. Herranz et al. [5] propose constant-size ABS schemes for the case of threshold predicates which can also be extended to admit other, more expressive kinds of monotone predicates. All these schemes employ bilinear pairings for their construction, which makes them complicated and less practical. Anada *et al.* [6] propose an ABS scheme without pairings in the random-oracle model. Their scheme first obtains a generic attribute-based identification (ABID) from a boolean proof system, combines ABID with a credential bundle scheme, and then applies the Fiat-Shamir paradigm to obtain a generic ABS scheme with attribute privacy.

As we see, there has been a certain amount of research done in the ABS field so far. However, the proposed schemes are very theoretical and focus only on the core cryptography. To the best of our knowledge, none of the above schemes is realized and put into practice. Some of these papers such as [3] mention some use cases which could use their ABS construction. Nevertheless, none of them talks about the implementation of their ABS constructions, nor do they discuss the practicalities in realizing attribute-based signatures.

Contributions. IRMA (I Reveal My Attributes)[1] [7] is an attribute-based authentication technology based on the Idemix [8] specification. A distinguishing feature of IRMA is that it has already an efficient and practical smart card implementation [9]. An important observation is that extending IRMA's authentication mechanism to support attribute-based signatures involves relatively little work. This paper explores the way IRMA can be used for generating practical attribute-based signatures. A similar idea is mentioned in an ABC4Trust[2] deliverable [10] where the authors suggest the possibility of including a application-specific message as an optional input to the credential presentation protocol that

[1] https://www.irmacard.org
[2] https://abc4trust.eu/

authenticates and signs the message with the user's private key. However, they do not discuss the way to do it in practice, whereas we focus on the *practical set-up*. Also, we suggest that *attribute-based* signatures provide a viable option to current digital signatures.

An attribute-based signature within IRMA is essentially a non-interactive proof of knowledge of authentic attributes (see Section 3). We use the phrase *IRMA signature* to refer to this particular realisation of an attribute-based signature, as opposed to the general concept of attribute-based signatures. Any device that carries the signer's attributes in IRMA is called an *IRMA token*. An IRMA token can be for instance, a smart card or a mobile phone. In this paper, we propose a practical ABS scheme arising from an existing implementation of the IRMA attribute-based authentication. The idea of merging both authentication and signature on the same token enables fast realization and roll-out of the technology.

2 About IRMA

The IRMA project aims to design and develop attribute-based credentials (ABCs) in practice. It is a partial implementation [9] of the Idemix technology [8,11]. Idemix is an attribute-based credential system, developed at IBM Research in Zürich. IRMA currently has implemented the privacy-enhancing features of ABC such as selective disclosure of attributes using zero-knowledge protocols.

The main idea behind IRMA is that authentic attributes stored on a token can be shown selectively via a zero knowledge proof. The token-holder has to give explicit permission to read a specified set of attributes (*e.g.* age, name) by entering a PIN code known only to him. For instance, an IRMA token can be used to prove the possession of a valid concert ticket or valid credentials to enter an office building by just revealing the 'ticket' or 'is an employee' attribute rather than revealing all the attributes on the token. In such cases there is no need to reveal a uniquely identifying attribute and this attribute-based method prevents linking of different proofs and implicit profiling of the token holder.

Attributes & Credentials. An *attribute* is a characteristic or a qualification of a person. Attributes can either be *identifying* or *non-identifying* properties. For example, 'full name', 'address', 'Social Security Number' are identifying attributes as the person can be uniquely identified by such attributes. Attributes, such as 'student' and 'age over 18' are non-identifying attributes as they do not uniquely identify a person; such properties can belong to other people as well. Collectively, these attributes can constitute the identity of a person.

Attributes are authentic. In the IRMA set-up, several related attributes are grouped into a cryptographic container known as a *credential* [7]. Authorities issue credentials to users following an authentication process. Each credential has an expiry date that denotes the validity of all the attributes contained in that credential. There could be n such credentials on the IRMA token issued by n issuers. However, for reasons of simplicity, in this paper, we consider all attributes to be contained in a single credential.

Cryptographic background of IRMA. IRMA is based on the Idemix protocol suite [8,11] and Camenisch-Lysyanskaya (CL) signature scheme [12,13]. All the zero-knowledge (ZK) proofs in the Idemix library are implemented as non-interactive ZK proofs using the Fiat-Shamir heuristic [14]. A brief description of Schnorr's schemes and the CL signature are provided in Appendix A and B.

3 IRMA's selective disclosure proofs as digital signatures

The concept of revealing only a selection of necessary attributes for completing a transaction is termed as *Selective Disclosure* in Idemix and IRMA. Selective disclosure is a zero-knowledge protocol currently used for authentication purposes. But, as will be shown here, it can also be used by an IRMA token holder for signing under selected attributes. When a selective disclosure (SD) proof is used for signing purposes, it becomes an *IRMA signature*. The key idea is that a *non-interactive zero-knowledge (NIZK) proof* (or so-called signature of knowledge) [14,15] signs a message. See Section A in the appendix for the description of the way a NIZK proof is constructed.

During an IRMA authentication, the verifier sends a nonce to the IRMA token to be included in the SD proof generation. This nonce is strongly bound to the proof and it helps the verifier check the freshness of the proof. It is meant to prevent a user from replaying the same proof to authenticate during different authentication sessions. We adapt this approach in a simple manner: If the hash of a message is used during an SD proof generation instead of the nonce, then the SD proof becomes the user's signature on the message. So, the main functional difference between an SD proof in IRMA authentication and IRMA signatures is the way the *nonce* is defined.

An SD proof acting as a user's signature is written as

$$SD\big((a_i)_{i\in D}; h(msg)\big),$$

where a_i is an attribute within a credential, D is the set of disclosed attributes, and $h(msg)$ is the hash of the message to be signed. Typically, an SD proof that becomes an IRMA signature proves the signer's possession of attributes and of the secret key involved in the proof generation. As this SD proof is a non-interactive zero-knowledge proof, first, the signer commits to a set of attributes and creates a commitment[3]; then he computes a challenge by hashing the commitment and the message to be signed. Conceptually, the challenge computation is denoted as

$$challenge = \mathcal{H}(commitment, h(msg)). \tag{1}$$

We input the hash $h(msg)$ of the message instead of the message itself to the hash function $\mathcal{H}$ that computes the challenge. This double hashing of the input might seem unnecessary. However, we intend as little change in existing IRMA

[3] As described in Appendix C, in a selective disclosure proof a commitment comprises an attribute issuer's randomized signature A' and the derived value $\tilde{Z}$.

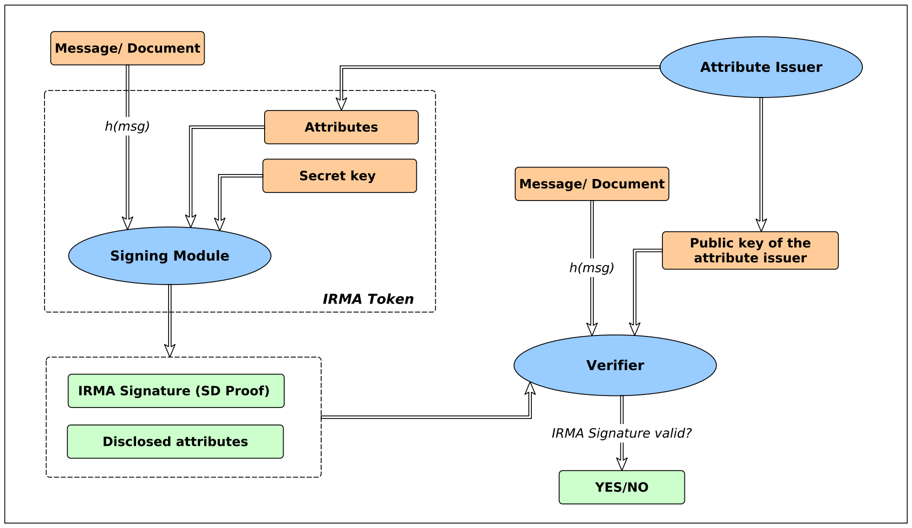

Fig. 1: IRMA signature generation and verification

authentication tokens as possible, and they expect a fixed-length nonce as the input to the selective disclosure proof. The hashes of messages are of fixed length, so they are functionally interchangeable with nonces. Also, computing the hash of a long message can be delegated to an external, more resourceful device from an IRMA token whose computational power and memory are limited. Here $\mathcal{H}$ and h can technically use the same hash algorithm *e.g.* SHA-3. The hash of a message or a document to be signed is denoted by '$h(msg)$' and the hash function to compute challenge in the non-interactive SD proof is denoted by '$\mathcal{H}$' throughout this paper.

3.1 IRMA signature scheme

The IRMA signature scheme depicted in Figure 1, consists of four algorithms: *Key Generation, Attribute Issuance, Signature Generation, Signature Verification.*

(1) *Key Generation.* Upon initialisation of an IRMA token, a secret key is generated and stored securely. It is used during attribute issuance, authentication and in signing. Since all these functions require this secret key, they are bound to the token, and hence to the signer.

(2) *Attribute Issuance.* An IRMA token owner can obtain attributes from authorized attribute issuers. An attribute issuer signs the credential containing attributes with its private signing key; the corresponding public key is used by verifiers, both in authentication and in signature verification.

(3) *Signature Generation.* A selection of the signer's attributes on an IRMA token forms the internal input to the signing module on the token. The hash

$h(msg)$ of the message to be signed forms the external input. The IRMA token outputs the required attributes and an SD proof ensuring that "the token owner has signed $h(msg)$ and possesses the attributes say, a_1 and a_2 issued by the issuer". A credential[4] on an IRMA token carries a CL signature (see Section B) which is randomized during a selective disclosure. We preserve the randomization of CL signature in IRMA signatures to ensure unlinkability among signature verifications. This randomization happens within the IRMA token. We also use the randomized CL signature to get trusted timestamps for the IRMA signature, as will be discussed in Section 4.1. Using the randomized CL signature, a signer generates an IRMA signature, a selective disclosure proof $SD\big((a_i)_{i\in D}; h(msg)\big)$ over $h(msg)$. The operations that have to be performed to generate a SD proof are described in Algorithm 1 in Section C.

(4) *Signature Verification.* Any verifier who wants to verify the signature on $h(msg)$ needs to have the public key of the issuer that has issued the relevant attributes to the IRMA token. The verifier calculates the hash of the message and uses it along with the issuer's public key parameters for verification. The verification steps are given in Algorithm 2 in Section C. During this verification, the verifier checks that the message was signed by an IRMA token holder who possesses the required attributes issued by an authorized issuer.

Privacy and security assurances provided by IRMA signatures. In terms of privacy, there are no public parameters of the IRMA token that act as an identifier of the token or token holder. Thus, it is impossible for the verifier to identify or link signatures to a particular signer if the disclosed attributes are non-identifying. This holds even if the signer signs the same document multiple times. In terms of security, IRMA signatures guarantee integrity and authenticity, more specifically:

- The message is not altered after signing.
- The attributes and the secret key are bound to the issuance and the signature.

3.2 Diversification between SD proofs used for authentication and signatures

Our goal is to use both the signature and authentication functions with the same set of attributes on the same IRMA token. An SD proof is used either for authentication with a fresh nonce or for signature generation with the hash of a message as input. As we already have an implementation of IRMA SD proofs for authentication, IRMA signatures can be easily realized in practice. Additionally, it is user friendly to have both attribute-based authentication and signature functions on the same token, along with an interactive user interface. We are well aware of the fact that, as the hash of a message and a random nonce look alike, an adversary could possibly send the hash of a message posing as a random nonce during an authentication session and make the user unknowingly

[4] A credential is conceptually comparable with a public-key certificate. But unlike a certificate, a credential is randomizable and enables selective disclosure.

sign this hash with the selective disclosure proof. This is a potential attack scenario in which an authentication session is misused to get a signature of the user without the user being aware of it. In this section, we propose a method to diversify signature and authentication protocol runs, in order to prevent the afore-mentioned attack.

Although two secret keys could be used for authentication and signing to separate the two functionalities on the token, all the user credentials would then have to be issued twice on that token as well, corresponding to both secret keys. This is because the secret key is associated with the issuance of every attribute to the IRMA token. Then, using two dedicated keys would be very similar to having authentication and signature functions on two different IRMA tokens. This contradicts our original goal. In order to avoid the duplication of all attributes on the token for authentication and signing purposes, we intend to use the *same secret key* for both purposes.

Domain separation is an efficient means to construct different function instances from a single underlying function. If the underlying function is secure, the derived functions can be considered as independent functions [16]. One can implement domain separation by appending or prepending different constants to the input for each of the function instances. We propose to apply *domain separation* for securely diversifying IRMA authentication and signing instances. We reserve a few bits, called *Dbit*, as the first input to the hash function while computing the challenge (see Line 11 in Algorithm 1) on the signer's end. Mathematically, the *Dbit* value is prepended to the rest of the inputs and indicates if the IRMA token is being used for authentication or for signing. In the current context, we need to separate two domains so we can use a single bit in *Dbit*. This bit will be set to 0 in case of authentication and 1 in case of signatures. We can program the signature generation module such that it takes user consent as the basis while deciding the value of *Dbit*. If the user gives his consent to sign a message msg, then the signature generation module (Algorithm 1) sets the *Dbit* to 1 and expects $h(msg)$ as one of the other inputs to the challenge computation. Thus, we rely on a correct token implementation.

The challenge computation during an IRMA signature generation previously denoted by (1) now becomes,

$$c = \mathcal{H}(Dbit = 1, commitment, h(msg)), \tag{2}$$

where c is the challenge, $\mathcal{H}$ is the hash function used to compute the challenge and $h(msg)$ is the hash function used to hash the message to be signed. If a valid signature is knowingly created by the legitimate signer, then during the verification, a verifier can successfully check its validity by reconstructing the challenge with $Dbit = 1$.

3.3 Brief security analysis of IRMA signatures

In this section, we informally analyze the security of our IRMA signature system by considering the possible ways in which an attacker can undermine the system.

We assume that the attacker has access to a polynomially bounded set of IRMA authentication transcripts denoted by AT and signature-message pairs denoted by SM. In the original IRMA authentication scenario, the attacker tries to spoof an authentication using the transcripts from the set AT. This is proven to be impossible in the paper by Camenisch *et al.* [12]. When we introduce IRMA signatures, three more possibilities arise for the attacker to undermine our system:

1. spoof an IRMA authentication by using the signatures from SM;
2. forge a new IRMA signature using the authentication transcripts from AT;
3. forge a new IRMA signature using the signature-message pairs from SM.

Case 1: Using IRMA signatures to impersonate a user during authentication. An attacker attempts to authenticate with one of the IRMA signatures from SM. We show that this is possible if he successfully finds either of the following two collisions. We also mention how we deal with such scenarios.

(i) Collision between the hash of a signed message and an authentication nonce.

$$h(msg) = nonce$$

where $h(msg)$ is the hash of a signed message msg from SM and $nonce$ is a random number sent by the verifier during an authentication session. The attacker can impersonate a user and maliciously authenticate if he finds a collision between the hash of the signed message that he already had and the nonce belonging to an authentication session.
However, because of the domain separation (see Section 3.2), the attacker cannot make the verifier accept this signature as a valid authentication proof.

(ii) Collision between the hash functions used for computing challenge in signature and authentication instances.

$$\mathcal{H}(Dbit = 1, commitment, h(msg)) = \mathcal{H}(Dbit = 0, commitment, nonce)$$

where $\mathcal{H}$ is the hash function used for computing the challenge within an SD proof. The inputs for $\mathcal{H}$ during the signature verification are $Dbit = 1$ and hash of a message $h(msg)$ whereas the inputs are $Dbit = 0$ and $nonce$ for an authentication proof verification. The attack succeeds if the attacker finds a collision between these two $\mathcal{H}$ instances. This is equivalent to having the same challenge results from the hash functions in authentication and signature sessions in spite of different inputs. If the attacker manages to find the above collision then he wins; he can then authenticate with a signature. We note that the attacker's chances of winning in this scenario depends on the collision resistance of the hash function being used in IRMA. If hash functions with no known collision attacks such as SHA-2 or SHA-3 is used then the above attack is highly improbable.

Case 2: Using IRMA authentication transcripts to forge a new IRMA signature. An attacker eavesdrops on many IRMA authentication sessions and collects authentication transcripts as denoted by AT at the beginning of this section. Then he tries to forge an IRMA signature out of an authentication transcript in the set AT. He is successful if he finds a collision in the two scenarios detailed in Case 1 and the same logic is followed here.

Case 3: Using IRMA signature-message pairs to forge a new IRMA signature. As we said before, the attacker possesses IRMA signatures for several messages of a user. In this case, the domain separation bit $Dbit$ is 1 for all signatures that the attacker already has. It is not sufficient if the adversary manages to find a hash collision between a previously signed message and a new message to create a valid signature; the attacker will also have to possess the right attributes of the user on his IRMA token to forge that user's signature. The security assumptions underlying the IRMA technology (same as the assumptions made by Idemix) prevent such forgery attacks. These assumptions are briefly mentioned below.

- As IRMA uses the Camenisch-Lysyanskaya (CL) signature scheme (see Section B for explanation), respective discrete logarithms based proofs prove the possession of valid attributes on the IRMA token.
- Unforgeability of IRMA signatures holds under the strong RSA assumption and the computational Diffie-Hellman assumption.
- In IRMA, a single proof involving all the attributes required by the signing policy, secret key of the IRMA token is considered as a valid signature. So, colluding users cannot combine their attributes associated with their secret keys in a single proof. This guarantees collusion resistance.

So we conclude that the adversary will not succeed in forging a new and valid IRMA signature with the help of given signature-message pairs, even if those pairs are of adversary's choice. Thus, an IRMA signature is *existentially unforgeable* under a chosen-message attack.

4 Infrastructural concerns for IRMA signatures

4.1 Timestamps in IRMA signatures

A practical aspect that becomes important when we migrate from authentication to signature functionality is the actual time of signing. A timestamp on the digital signature attests when the message or the document was signed. It provides a unequivocal proof that the contents of the signed document existed at a point-in-time and have not changed since then.

In the case of IRMA signatures, there are two kinds of dates or timestamps to be considered:

1. date and time at which the signature was generated,
2. expiry dates of the attributes under which the signer has signed $h(msg)$.

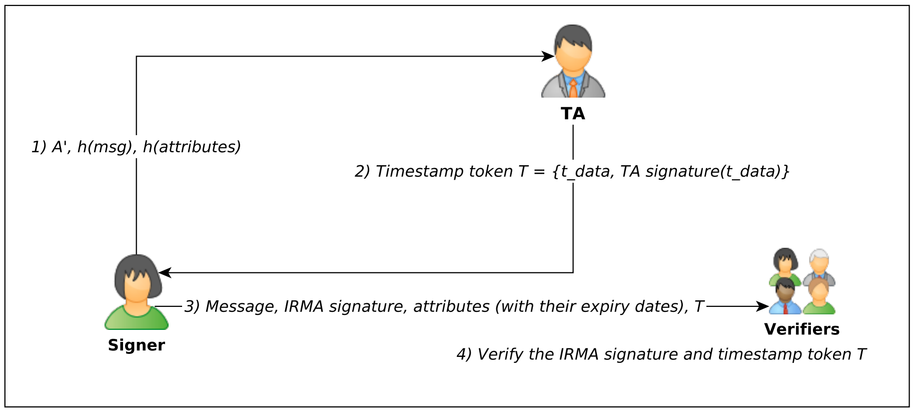
TA
1) A', h(msg), h(attributes)
2) Timestamp token T = {t_data, TA signature(t_data)}
Signer
3) Message, IRMA signature, attributes (with their expiry dates), T
Verifiers
4) Verify the IRMA signature and timestamp token T

Fig. 2: Timestamping the IRMA signature

To include the time of signing, a signer has two options: The signer can use the local time of his IRMA token (if available) as part of the message to be signed. If a verifier requires a more secure timestamp, the signer can obtain an authorized timestamp signed by a Timestamp Authority (TA). A TA is an entity that is trusted to provide accurate time information.

The selective disclosure proof in IRMA discloses the expiry dates of the attributes involved in the proof by default. These expiry dates are included in the set of disclosed attributes and they can easily be verified by the verifier. Since attributes should be valid when generating an IRMA signature, the expiry dates should be greater than the time included in the timestamp. Therefore, it is recommended to include a validity check at the beginning of an SD proof generation algorithm (see Algorithm 1 in the Appendix).

We propose a timestamping scheme that enables a signer to get a signed timestamp from a TA for an IRMA signature as shown in Figure 2. In our scheme, a signer sends a timestamp request to the TA to get a trusted timestamp. The timestamp request consists of randomized CL signature[5], hash of the message to be signed and hash of the attributes, denoted by A', $h(msg)$ and $h(attributes)$ respectively. This request does not reveal any information about the signer or the message to the TA, hence, it is privacy friendly. TA issues a signed timestamp. Upon receiving the timestamp, the signer inputs the timestamp to the IRMA signature generation. The timestamping scheme is illustrated in Figure 2 and elaborated in the following steps:

1. *Timestamp request.* The signer requests TA for a timestamp by sending $A', h(msg), h(attributes)$.
2. *Timestamp token calculation.* TA does the following:

[5] The randomized CL signature is used in both IRMA signature generation and verification (see Algorithms 1 and 2).

   - concatenates all the data sent by the signer in the timestamp request with the current timestamp t and digitally signs t_data.
   - combines t_data and TA's signature on t_data together into a *timestamp token* T and sends T to the signer.
3. *IRMA signature generation with the timestamp token.* The signer provides the timestamp token T as one of the inputs to the IRMA signature generation algorithm (see Algorithm 1). Now, during the challenge computation, T is hashed along with the diversifier $Dbit$, commitment and $h(msg)$ as,

$$c = \mathcal{H}(Dbit = 1, commitment, h(msg), T). \tag{3}$$

   Finally, the signer sends the IRMA signature on $h(msg), T$, and the disclosed attributes to the verifier.
4. *IRMA signature with timestamp verification.* Upon receiving the IRMA signature, the verifier verifies
   - the IRMA signature using the attribute issuer's public key and the disclosed attributes,
   - the TA's signature on t_data within the timestamp token T by using the TA's public key,
   - if $A', h(msg), attributes$ are the same in the IRMA signature and t_data that is within the timestamp token T.

   As we see, an IRMA signature comprises multiple logical layers. Table 1 summarizes the three signatures that have to be verified during an *IRMA signature with timestamp* verification.

Table 1: Abstraction of signature layers involved in IRMA signatures.

Signature	**Public key used to verify**	**Signing party**
Attribute-based signature	Disclosed attributes & issuer's signature	Signer
Randomized CL signature	Attribute issuer's public key	Attribute issuer
DSA or RSA signature	TA's public key	TA

Because of the verifications performed in the above step 4 (IRMA signature with timestamp verification), a verifier knows/or has cryptographic assurance of the following properties.

- The IRMA signature was not generated before the time indicated by the TA's timestamp t.
- The signature on the message with the enclosed attributes is bound to this timestamp t.
- The message that is signed has remained unchanged since the time t.

In addition, the implementation guarantees that the signer's attributes used for signing were valid at time t, since this is checked in Algorithm 1.

4.2 Revocation of credentials in IRMA

The IRMA project has been focused on preserving user-privacy in authentication. The revocation scheme [17] for IRMA authentication that has been proposed by the IRMA design team avoids identifiers in revocation that would enable linking the revocation checks to a single user. This scheme involves a semi-trusted party in the system, a *Revocation Authority (RA)* that is responsible for revoking the credentials. The RA keeps track of the revocation values of revoked credentials.

In the existing IRMA-revocation scheme [17] the time is split into epochs (time intervals) and the RA chooses a generator for each epoch and each verifier. When a credential is revoked the RA makes a global revocation list RL that consists of revocation tokens $R_{i_1..n}$ that are computed from the generators and the individual revocation values of the credential holders. The RA sends this revocation list RL to all the registered verifiers. During an authentication with a revocation check, the verifier sends the IRMA token its specific per-epoch generator, and the token calculates the revocation response R by embedding its token-specific revocation values. This value R is generated along with the selective disclosure proof and made available to the verifier. The verifier can just check $R \in RL$ to know if the credential used in the generation of the selective disclosure proof is revoked or not.

However, in the case of signatures, a signer need not know the verifiers in advance, so, the verifier-specific generators would not work. Also, the per-epoch concept will have to be modified to suit the signature verification scenario. If the signer calculates a revocation token for the current epoch along with the signature, the verifier has to do a revocation check in that epoch. In the case of a delayed verification in a different epoch, the verifier will have to retrieve the revocation token list from the RA corresponding to the epoch in which the signer signed. We realize that the design of a privacy-friendly revocation scheme for digital signatures that ensures complete unlinkability is not trivial. The existing revocation scheme for IRMA authentication has to be adapted to IRMA signatures which is subject to further research.

As a possible solution for revocation, attribute expiry dates can be short and re-issuing of attributes can be made simple. If a security breach or a key compromise is detected then the attribute issuer would just stop re-issuing the attributes to that particular IRMA token.

5 Discussion

We briefly describe a few use cases for attribute-based IRMA signatures, and give an estimate of their performance.

5.1 Use case scenarios

Use cases requiring the flexibility offered of *role-based* IRMA signatures.

1. In the introduction we briefly mentioned a medical doctor signing a medical statement about a patient using his own medical license number and specialisation attributes. This can be applied to many professionals signing documents in which their competence is a useful part of the signature. More generally, this leads to what may be called *role-based* signatures.
2. With such role-based signatures one can distinguish professional and personal signatures. For instance, the signature of a notary should be different when he is selling a house professionally or privately. Attribute-based signatures are ideally suited for making such differences explicit, for all verifiers to see.

Use cases for signatures that ensure signer *privacy/anonymity*.

3. *Anonymous voting.* In large scale elections, there are usually two main phases: (i) registration and (ii) vote casting. A crucial difference between these two phases is that the first one should be identifying, whereas the second one should not.
   During the registration phase, a potential voter can authenticate to a voting authority in a properly identifying manner, *e.g.* via his citizen registration number. This identity is needed to check if the person at hand is eligible to vote.
   If the check is successful, then the voting authority can blindly issue a random 'voting ID' attribute (via blind signatures) to his IRMA token. During the election phase, this voter can sign his vote with his IRMA token under his 'voting ID' attribute. The (random) voting IDs of all the voters are stored and if any voter tries to vote for the second time, then the voter ID matches with one of the previously stored voter IDs. This second vote can either be discarded or it can replace the first vote based on the voting authority's policy decision. Thus, we can keep the voters anonymous and also avoid double voting scenario by using IRMA signatures. Here we note that a potential voter can use the same IRMA token for authenticating during registration and for signing anonymously during the vote casting phase.
4. *Anonymous petitions* is another application similar to the anonymous voting that can benefit from IRMA signatures.
5. IRMA signatures can also be used by confidential sources who want to keep themselves unidentified to a journalist to whom they reveal information. But still they can include some relevant attributes in the signatures on their statements, in order to provide credibility.

5.2 Estimating the efficiency of IRMA signatures

As we have described in Section 3, a selective disclosure proof can serve both authentication and signing purposes. The applied method depends on the input values given to the proof generation algorithm. Thus, the performance times of an IRMA authentication and an IRMA signature are comparable. We use the performance results obtained and documented in the Idemix chapter of Pim

Vuller's PhD thesis [18] as a starting point for our estimation. Based on these values we assess the execution time of the IRMA signature generation and verification instances. The smart card type used in this calculation is an Infineon SLE78 chip with MULTOS platform (ML3-36K-R1). In IRMA, a typical credential consists of 5 attributes. For estimating the running time of a selective disclosure proof, we consider two cases. Disclosing one attribute takes 1.2 seconds, while disclosing four attributes takes 0.93 seconds. Note that the more attributes are disclosed, the fewer are hidden and the shorter time it takes. Furthermore, we also note that in practice the fifth attribute, the secret key, is never revealed.

Now let us turn from IRMA authentication to *IRMA signatures*. The total time taken for an entire signature generation operation is the sum of a selective disclosure proof and other minor operations like timestamp retrieval, denoted by δ. The value of δ is dependent on the network connection speed as the timestamp request and retrieval takes place online. Therefore, the total execution time for an IRMA signature generation (in the case when only 1 out of 4 attributes from a credential is disclosed) can be estimated as *(1.2 +* δ*) seconds*.

The signature verification consists of an extra modular exponentiation operation w.r.t. the signature (SD proof) generation. Thus, we expect that the time taken for SD proof verification is a bit more than the time taken for its generation. However, the signature verification terminals (*e.g.* personal computers) are usually computationally more powerful in terms of both time and memory than a smart card. So the time taken to verify an SD proof is mostly in the order of a *few milliseconds*.

We see that IRMA signature generation on smart cards is reasonably efficient in terms of execution time to be put into practice. The execution time could be further decreased if mobile phones are used as IRMA tokens for generating IRMA signatures.

6 Concluding remarks

We present the first practical and easily realizable form of attribute-based signatures by building on top of the IRMA technology. What we call IRMA signatures are created by extending the existing smart card (and phone) implementation of IRMA authentication. We show how we can securely use authentication and signature functions on a single IRMA token using the same secret key. In addition we elaborate on the infrastructural aspects related to usable digital signatures such as secure timestamping. There is ongoing work in adapting the existing revocation scheme for attribute-based signatures without forgoing any of its privacy and unlinkability guarantees.

In conclusion, IRMA signatures offer much greater functionality and flexibility than traditional PKI-based digital signatures in terms of role-based signing, contextual privacy guarantees to the signers, and ease of comprehending the signature semantics to the verifiers.

References

1. David Chaum and Eugène Van Heyst. Group signatures. In *Advances in Cryptology—EUROCRYPT'91*, pages 257–265. Springer, 1991.
2. Guo Shaniqng and Zeng Yingpei. Attribute-based signature scheme. In *Information Security and Assurance, 2008. ISA 2008*, pages 509–511. IEEE, 2008.
3. Hemanta K Maji, Manoj Prabhakaran, and Mike Rosulek. Attribute-based signatures. In *Topics in Cryptology–CT-RSA 2011*, pages 376–392. Springer, 2011.
4. Tatsuaki Okamoto and Katsuyuki Takashima. Efficient attribute-based signatures for non-monotone predicates in the standard model. In *Public Key Cryptography–PKC 2011*, pages 35–52. Springer, 2011.
5. Javier Herranz, Fabien Laguillaumie, Benoît Libert, and Carla Ràfols. Short attribute-based signatures for threshold predicates. In *Topics in Cryptology–CT-RSA 2012*, pages 51–67. Springer, 2012.
6. Hiroaki Anada, Seiko Arita, and Kouichi Sakurai. Attribute-based signatures without pairings via the fiat-shamir paradigm. In *Proceedings of the 2nd ACM workshop on ASIA public-key cryptography*, pages 49–58. ACM, 2014.
7. Gergely Alpár and Bart Jacobs. Credential design in attribute-based identity management. In *Bridging distances in technology and regulation, 3rd TILTing Perspectives Conference*, pages 189–204, 2013.
8. IBM Research, Security Team. Specification of the Identity Mixer Cryptographic Library, version 2.3.4. Technical report, IBM Research, Zürich, February 2012.
9. Pim Vullers and Gergely Alpár. Efficient selective disclosure on smart cards using Idemix. In Simone Fischer-Hübner, Elisabeth de Leeuw, and Chris Mitchell, editors, *Policies and Research in Identity Management (IDMAN)*, pages 53–67. Springer, 2013.
10. Jan Camenisch, Ioannis Krontiris, Anja Lehmann, Gregory Neven, Christian Paquin, Kai Rannenberg, and Harald Zwingelberg. D2. 1 architecture for attribute-based credential technologies–version 1. *ABC4Trust Deliverable D*, 2, 2011.
11. Jan Camenisch and Els Van Herreweghen. Design and implementation of the idemix anonymous credential system. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 21–30. ACM, 2002.
12. Jan Camenisch and Anna Lysyanskaya. An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation. In Birgit Pfitzmann, editor, *Advances in Cryptology — EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer Berlin / Heidelberg, 2001.
13. Jan Camenisch and Anna Lysyanskaya. A Signature Scheme with Efficient Protocols. In Stelvio Cimato, Giuseppe Persiano, and Clemente Galdi, editors, *Security in Communication Networks*, volume 2576 of *LNCS*, pages 268–289. Springer Berlin / Heidelberg, 2002.
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. Springer, 1987.
15. Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.
16. Keccak team. Note on keccak parameters and usage. http://keccak.noekeon.org/NoteOnKeccakParametersAndUsage.pdf. Online; accessed 6-July-2015.
17. Wouter Lueks, Gergely Alpár, Jaap-Henk Hoepman, and Pim Vullers. Fast revocation of attribute-based credentials for both users and verifiers. *IFIP Advances in Information and Communication Technology*, 2015.

18. Pim Vullers. *Efficient Implementations of Attribute-based Credentials on Smart Cards*. PhD thesis, Radboud University Nijmegen, The Netherlands, 2014.

A Schnorr's Identification scheme

Schnorr's identification scheme [15] is a simple three-way zero-knowledge proof scheme which proves the knowledge of a discrete logarithm x of a specific number $y \pmod n$:

$$PK\{(x) : y = g^x \pmod n\}$$

where PK is the proof of knowledge, x is the discrete logarithm of y and g is the generator belonging to the cyclic group G_q of order q.
In order to prove knowledge of $x = \log_g y$, the prover interacts with the verifier as follows:

1. The prover commits to randomness $r \in [0, q-1]$; therefore, the first message $t = g^r \in G_q$ is also called a commitment.
2. The verifier replies with a challenge $c \in [0, q-1]$ chosen at random. (In practice, c can be chosen from a smaller set – depending on the security parameter –, but here we omit these details.)
3. After receiving c, the prover sends the third and last message (the response) $s = r + cx \pmod q$.

The verifier accepts, if $g^s = ty^c$ in G_q.
The security of Schnorr's identification scheme relies on the hardness of the discrete logarithm problem. In the interactive proof, the verifier can be sure in the last step that the prover knows the discrete logarithm of y if it satisfies the correctness condition:

$$g^s = g^{r+cx} = g^r g^{cx} = t(g^x)^c = ty^c.$$

Applying the Fiat–Shamir heuristic [14], one can achieve a non-interactive scheme which reduces the number of rounds of information exchange between the prover and the verifier. This is often used to translate a zero-knowledge protocol into a signature scheme, or to reduce the communication overhead of the interactive protocols. To make a zero-knowledge protocol non-interactive the challenge c is not retrieved from the verifier but computed as

$$c = Hash(msg, t),$$

where msg is the message to be signed and t is the commitment. The Idemix technology uses similar non-interactive proofs of knowledge.

B Camenisch-Lysyanskaya (CL) signature

Camenisch *et al.* propose a provably secure signature scheme for issuing a signature on a set of attributes and proving the knowledge of those attributes [13].

The resulting signature from their scheme is termed as *CL signature* and it is used as a building block for IRMA. We briefly explain the structure of a CL signature in this section.

Let us consider the safe primes p and q as the signer's private keys. The signer randomly selects $a, b, c \in QR_n$. Then a, b, c are published as public keys. If a message m is to be signed, a random number v and a prime number e are chosen and the signature is computed as shown below:

$$A \equiv (a^m b^v c)^{e^{-1} \bmod |QR_n|} \pmod{n}.$$

Since we know the values of p and q and they are safe primes, we also know p' and q' and $\mid QR_n \mid = p'q'$. The CL signature over the message m is composed as

$$m: \ \{A, e, v\}$$

In the IRMA context, we can define the Camenisch-Lysyanskaya signature over the messages m as the triplet (A, e, v) such that e is the random prime used as the ephemeral RSA public key for this signature and v is a random number and A is the RSA signature over the message m. The following check is done in order to verify the correctness of the above CL signature:

$$A^e \equiv a^m b^v c \pmod{n}$$

If the verification equation holds, the signature is valid; otherwise, the signature is invalid. The CL signature described here can also be applied over a block of messages. The unforgeability of the CL signature scheme relies on the *Strong-RSA assumption*.

A CL-signature (A, e, v) can be randomized easily. First, one has to select a random value r from a specific, large interval, then, one performs the following computation:.

$$A' := A \cdot b^r \pmod{n}, \qquad v' := v + er.$$

Indeed, (A', e, v') is also a valid signature over message m:

$$A'^e \cdot b^{-v'} \equiv (A \cdot b^r)^e \cdot b^{-v-er} \equiv A^e \cdot b^{er} \cdot b^{-v} \cdot b^{-er} \equiv A^e \cdot b^{-v} \equiv a^m c \pmod{n}.$$

C IRMA signature and verification algorithms

In this section, we provide the technical details and algorithms used for the IRMA signature generation and verification. As we have described earlier in Section 3.1, a signer signs a message under a set of attributes with his secret key and a verifier uses the attribute issuer's public key to verify this IRMA signature. The public key of the attribute issuer is $(n, S, Z, \{R_i\}_{i \in M})$ where M denotes the set of attribute indices, and hence the maximum number of attributes issued by that issuer. In IRMA selective disclosure, D denotes the set of attributes to be disclosed from the IRMA token to the verifier and H denotes the set of attributes

Algorithm 1 IRMA signature generation algorithm.

```
1: function IRMA-Sign($\{a_i\}_{i \in D}, \{a_i\}_{i \in H}, (A', e, v'), h(msg), (n, S, Z, \{R_i\}_{i \in M}), T$)
2:     for all $i \in D$ do
3:         Verify the validity of each $a_i$ w.r.t. timestamp in T
4:         if invalid then Exit
5:     $\tilde{e} \leftarrow$ Random( )
6:     $\tilde{v} \leftarrow$ Random( )
7:     $\tilde{Z} \leftarrow {A'}^{\tilde{e}} \cdot S^{\tilde{v}} \mod n$
8:     for all $i \in H$ do
9:         $\tilde{a}_i \leftarrow$ Random( )
10:        $\tilde{Z} \leftarrow \tilde{Z} \cdot R_i^{\tilde{a}_i} \mod n$
11:    $c \leftarrow$ Hash($Dbit, A', \tilde{Z}, h(msg), T$)   //compute challenge using commitment,
   $h(msg)$, timestamp
12:    $\hat{e} \leftarrow \tilde{e} + c \cdot e$
13:    $\hat{v} \leftarrow \tilde{v} + c \cdot v'$
14:    for all $i \in H$ do
15:        $\hat{a}_i \leftarrow \tilde{a}_i + c \cdot a_i$
    return $(c, A', \hat{e}, \hat{v}, \{\hat{a}_i\}_{i \in H}, T)$
```

Algorithm 2 IRMA signature verification algorithm.

```
1: function IRMA-verify($(c, A', \hat{e}, \hat{v}, \{\hat{a}_i\}_{i \in H}, \{a_i\}_{i \in D}), h(msg), (n, S, Z, \{R_i\}_{i \in M}, T)$)
2:     $\hat{Z} \leftarrow Z^{-c} \cdot {A'}^{\hat{e}} \cdot S^{\hat{v}} \mod n$
3:     for all $i \in D$ do
4:         $\hat{Z} \leftarrow \hat{Z} \cdot R_i^{c \cdot a_i} \mod n$
5:     for all $i \in H$ do
6:         $\hat{Z} \leftarrow \hat{Z} \cdot R_i^{\hat{a}_i} \mod n$
7:     if $c \neq$ Hash($Dbit, A', \hat{Z}, h(msg), T$) then return Invalid
    return Valid
```

on the token that needs to be hidden from the verifier. Therefore, H and D are disjoint sets of attributes: $H \cup D = M$ and $H \cap D = \emptyset$.

Algorithm 1 describes the operations that have to be performed to generate a proof of knowledge of the secret key and the hidden attributes. The proof proves the remaining attributes $\{a_i\}_{i \in H}$, that are hidden during this phase, are known by the signer (i.e. token). In the case of IRMA signature generation, Algorithm 1 takes hash the of message to be signed, denoted by $h(msg)$ and timestamp, denoted by T as inputs while computing the challenge c. The IRMA signature is essentially a selective disclosure proof that is generated over $h(msg)$ and T. This signature can then be verified using the Algorithm 2.