PriMan: Facilitating the Development of Secure and Privacy-Preserving Applications

Andreas Put, Italo Dacosta, Milica Milutinovic, and Bart De Decker

KU Leuven, Dept. of Computer Science, iMinds-DistriNet
Celestijnenlaan 200A, 3001 Heverlee, Belgium
{firstname.lastname}@cs.kuleuven.be

Abstract. Security and privacy are essential in today's information-driven society. However, security technologies and privacy-enhancing technologies (PETs) are often difficult to integrate in applications due to their inherent complexity and steep learning curve. In this paper, we present a flexible, technology agnostic development framework that facilitates the integration of security and privacy-preserving technologies into applications. Technology-specific configuration details are shifted from the application code to configuration policies. These policies are configured by domain experts independently from the application's source code. We developed a prototype in Java, called PriMan, which runs on both desktops and Android based devices. Our experimental evaluation demonstrates that PriMan introduces a low and acceptable overhead (e.g., less than one millisecond per operation). In addition, we compare PriMan with other, freely available solutions. PriMan facilitates the integration of PETs and security technologies in current and future applications.

Keywords: Software Architecture, Security, Identity Management, Framework.

1 Introduction

Security and privacy are requirements that keep gaining importance in today's information-driven world. Every system or application connected to the Internet has to be sufficiently secured. However, information leaks and security breaches are commonplace, even though many can be prevented by the proper use of proven security and privacy technologies. For example, the Open Web Application Security Project (OWASP) has identified a list of important vulnerabilities for mobile applications [1]. Among the top vulnerabilities on this list are: insecure data storage, insufficient transport layer protection, poor authorization and authentication, etc. Developers, however, need to have knowledge of various technical concepts (e.g., security protocols, key management, cryptographic algorithms, etc.) to correctly implement technologies that mitigate these vulnerabilities. Furthermore, deploying and configuring PETs and security technologies requires additional expertise. As a result, it is challenging for average developers to implement adequate security and privacy mechanisms in their applications.

N. Cuppens-Boulahia et al. (Eds.): SEC 2014, IFIP AICT 428, pp. 403–416, 2014.

Another problem is that many security and privacy mechanisms are coded directly into applications and, hence, mixed with the business logic. This practice hinders code reusability, as large modifications to the code base are typically required to reuse mechanisms from one application into another one. In addition, these large modifications increase the possibility of errors and security and privacy weaknesses.

In this paper we present PriMan, a framework developed within the MobCom project[1] [2], that facilitates the integration of security and privacy technologies for authentication and protection of data while in transit (over communication channels) or being stored. The abstractions made by PriMan shift the technology-specific configuration details from the application code to configuration policies. Developers thus only work with abstract components, the interface of which is common for every technology used underneath.

A developer can use the framework to initialize a connection or to store an object, while the configuration policy defines which type of connection is used (e.g., HTTP or HTTPS) and how the object is stored (e.g., in a file or in an encrypted database). In addition, the framework simplifies application code, as the framework itself performs many of the common tasks associated with each operation. These features make it well suited for rapid prototyping, or even to test the effects of different technologies on applications. PriMan uses a lightweight and modular architecture. Its thin middleware layer resides between the application code and the different technology libraries, introducing a low performance overhead. Our benchmarks show that PriMan adds an overhead of less than one millisecond per operation. Moreover, PriMan's architecture is designed to run not only on servers but also mobile devices.

In short, we make the following contributions:

- We defined abstractions that capture common features and traits over several sets of technologies in such a way that they are independent from their underlying algorithms and/or cryptographic components. Although the technology specific features are hidden by these abstractions, they still can be specified in configuration policies.
- We designed and implemented a PriMan prototype in Java, which includes a full-fledged implementation of the Idemix anonymous credential system [3, 4]. We evaluated the efficiency of our framework by measuring how much overhead it introduces. We also compare PriMan to the CARL framework [5] and to ABC4Trust [6].
- We integrated privacy-preserving authentication policies with support for anonymous and traditional credentials (e.g., Idemix and X.509 certificates[7]). These policies allow service providers to easily specify what a client needs to prove in order to gain access to a service. Clients can inspect these policies before authentication in order to analyse what information they reveal to the service provider.

[1] *The Mobile Companion (MobCom)* project's main goal is to transform mobile devices such as smartphones into powerful, flexible and user-friendly tools to manage our identities in a privacy-friendly way.

2 The PriMan Framework

PriMan is an application development framework that focuses on security, privacy, flexibility and ease of use. The latter is mainly geared towards application developers, as they are the targeted users of the framework. Firstly, the framework offers PETs and security technologies to developers. Secondly, it provides a technology-agnostic API, i.e., code written with PriMan is completely independent from the actual technologies used underneath. The framework configures at runtime which components are used (e.g., X.509 or Idemix) by following configuration policies, which also contain the specific configuration details of the chosen technology. This concept is similar to the separation of document content from its style for web pages (i.e., HTML and CSS). By separating application code from configuration details, developers can focus on writing application logic without concerns about the details of the underlying technologies. Moreover, by relying on configuration policies, PriMan facilitates changes to current technologies or switching among different technologies (e.g. X.509 to Idemix). Furthermore, PriMan automatically performs common development tasks, reducing the programming work left to the developer.

2.1 Abstractions

PriMan defines abstractions over sets of technologies to create an intuitive and technology-agnostic API. This section explores these abstractions for two of the framework's components: the connection and credential components.

The starting point for making these abstractions is: what are the common, high level concepts and operations of a set of technologies, i.e., which high level operations do developers expect from these technologies. For example, for the set of connection technologies (table 1), the common concepts are *connections* and *connection listeners*, the latter being an object which listens for incoming connections. The specific configuration details for each technology are abstracted by *connection parameters*, which are created from configuration policies. After a connection is established, it can be used to send or receive data and finally, it can be shut down.

For example, TCP sockets are one of the simplest connection technologies, however, a significant amount of configuration options are possible. Input and output streams are used to write and read data from a socket, and the technology-specific parameters like hostname, port or time-out values are selected from a configuration policy.

As authentication is an important task in many systems, the credential component is a major part of PriMan. Table 2 shows the most important abstractions produced for this component. The table shows how three credential providers (X.509, Idemix and username-password) implement these abstractions. First, to create new credentials, an issuer has to be set up, which involves generating and installing the necessary cryptographic keys. For X.509 an issuing keypair

Table 1. The most important abstractions defined by PriMan's Connection component are listed in the first column. The second and third column list how these abstractions are implemented by the TCP and SSL providers.

Abstraction	**TCP socket**	**SSL socket**
Create Connection and listen (server) — create connection to server (client)	Open TCP socket and Listen (server) — create TCP connection (client)	Open (SSL) socket using an SSL Certificate and listen (server) — create SSL connection (client)
Send data	Send object through socket	Send object through SSL socket
Receive data	Receive obj. through socket	Receive obj. through SSL socket
Close connection	Clear and close TCP socket	Clear and close SSL socket
Connection params.	Port, address, timeout	Port, address, timeout, cert.-store file, type and pasword

and certificate is required. Similarly, an issuing keypair is required for Idemix. A username-password credential does not require a cryptographic keypair or an issuing certificate. Therefor, their issuer parameters are empty. The issuance of new credentials is a protocol between the issuer and the client, which requires an established connection. Framework connection objects are used to establish these connections and to send or receive data, independent of the technology used underneath.

Once credentials have been issued, they can be used to create proofs of certain claims. X.509 and passwords only support one claim: "I own this credential" (this certificate or this username). Idemix-claims are more complex[2] and defined in a proof specification. Each credential component creates and verifies the (cryptographic) proof of a claim in its own particular way. With X.509 certificates, a signature of a nonce or challenge is created and verified, while an Idemix proof consists of several parts: a proof of knowledge of a CL-signature [8], and the zero-knowledge proofs [9] for all predicates contained in the claim. Proving ownership of a username can be simply done by sending the corresponding password or a one-way function thereof to the verifier, who can verify whether it matches a stored copy.

PriMan's abstractions have been derived empirically in a two-step process. First, the common traits, concepts and operations of a set of technologies are identified and a first version of the abstractions is created. Second, these abstractions are validated in practice by developing applications with them. After this, a new iteration of this process is performed using the feedback gathered in the second stage, until the API stabilizes. In a following stage, the API will be validated and tested in a more formal manner and larger setting.

[2] In contrast to X.509, Idemix can hide the attributes not required in an authentication. In addition, it supports predicate proofs, in which (in)equality of attributes can be proven with other attributes or constants, without revealing the exact values of these attributes.

Table 2. The most important abstractions defined by PriMan's credential component are listed in the first column. The second and third column list how these abstractions are implemented by the X.509 and Idemix providers. The fourth column shows how to the same abstractions apply to the most used credential of all, the username-password pair.

Abstraction	X.509	Idemix	Passwords
Create Issuer	Create self-signed cert. or use existing cert. to create issuer	Create Idemix keypair	Do nothing
Issuer params.	Type, sign/hash algo, values keystore file and password	Location of idemix public parameters	
Issue credential (Server)	Use certificate and the client's cert. request to generate a new certificate	Use keypair and key-value pairs to interactively issue a new Idemix cred.	Store (salted) password hash
Get new cred. (Client)	Generate keypair and create cert. request	Use the user secret to interactively issue cred.	Create username and password
Issuance params.	Type, sign/hash algo, values keystore file and password	Location of idemix public parameters, values	Username-password
Claim	X.509 certificate	Idemix proof specification	The username
Create proof of a claim	Sign a nonce with the private key of the cert.	Generate Idemix proof with regard to a nonce	Password
Verify proof	Verify the signature with public key of the cert.	Verify the Idemix proof w.r.t. the proof spec.	Match password with stored one

2.2 Framework Architecture

Three main concepts are integral to PriMan's architecture: *Managers*, *Service Providers* and *Core Classes* (Figure 1).

Managers. A manager is the main access point to PriMan's functionality, and is situated in the *framework layer*. It forms part of the interface to a set of technologies, e.g., the connection manager allows a developer to set up a communication channel of a particular technology. The manager offers a technology-agnostic interface; the technology which is actually used, depends on the method parameters it receives from the *application layer*. These parameters often include a configuration policy, which specifies the technology-specific configuration details for the technology provider below. Furthermore, Managers provide a flexible mechanism responsible to select the appropriate provider.

Core Classes. The core classes are the technology-agnostic abstractions of the different technology-specific concepts. Core classes consist mostly out of abstract classes and interfaces and thus, their main feature is the definition of one well-fitting API, which is independent from algorithms or components used underneath.

Service Providers. A service provider is an implementation of a specific technology such as X.509 or Idemix credentials. Each service provider implements a specific interface related their associated technology, the *Service Provider Interface* (SPI). In addition, they contain the implementations of the core classes associated with the provider's technology. The layer between PriMan's API and a technology's library is made as thin as possible, which

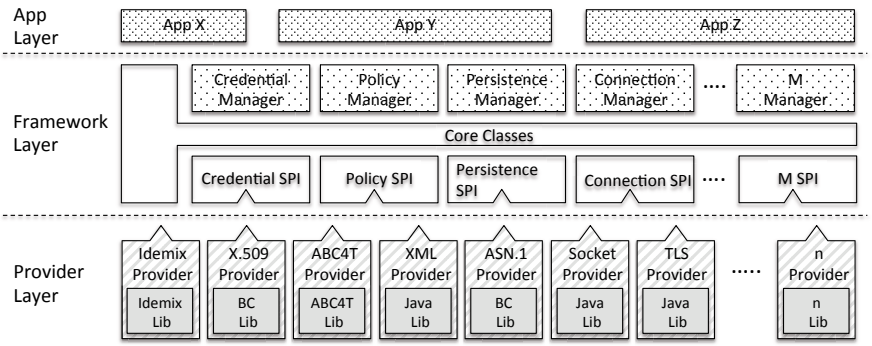

Fig. 1. Applications using PriMan are layered in three parts. The application-specific code resides in the *App Layer*. The technology-agnostic API for the application code is defined in the *Framework Layer*. Finally, the technology-specific implementations which plug into the framework reside in the *Provider Layer*.

significantly reduces the framework's overhead. This is done using the wrapper software design pattern, in which each method call is mapped to one or more operations of the software library that provides the implemented technology. Due to the framework's abstractions, providers often map one framework operation to multiple method calls of the technology library. This approach decreases the chances of developers' mistakes, as most of the low level details remain hidden for the developer while a significant part of the work is performed by the framework.

The manager can address its providers uniformly because each provider associated with a certain manager complies to the same SPI. This architecture provides a flexible plug-in mechanism, through which plug-ins (providers) can be loaded or unloaded at run-time.

2.3 Framework Components

The PriMan framework without any providers is just an empty shell. The framework defines the abstractions over sets of technologies and provides a plug-in mechanism for loading these providers, which are implementations of the defined abstractions. PriMan's pluggable architecture is well suited to deploy applications on mobile devices as well as desktops. Heavyweight components could be exchanged for more lightweight ones suitable for mobile platforms. The downside of such a plug-in architecture is that it introduces some additional overhead. However, this runtime overhead is can be significantly reduced by preloading service providers.

The PriMan framework is a good tool to perform rapid prototyping of applications. A developer can easily change the used technologies in her set-up, without having to recompile the code by simply changing one of the configuration policies. In doing so, the performance effects of different technologies (e.g., using Idemix instead of X.509) on these applications can be analyzed.

2.4 Implementation

We implemented the above architecture using the Java programming language. Java greatly simplifies the development of a mobile version of PriMan, as the Android operating system uses Java as its native programming language. Most of PriMan's desktop-version code can be reused in the mobile version. In addition, most framework providers can be plugged into the Android version of PriMan, if they do not depend on platform dependent features.

At the time of writing, ten different providers have been implemented for Desktop and Android platforms. The Idemix [4] and X.509 [7] credential systems, the CARL authentication policy language [5] and ABC4Trust presentation policies [6, 10], the XML and PEM/ASN.1 persistence provider [11], and the TCP and SSL connection providers. For Android, the framework supports two secure file encryption providers: one based on Android's secure keyring and one based on a tamper proof module.

3 Functional Evaluation

To evaluate PriMan's capabilities, we considered the following scenario. A municipal library offers inhabitants of the city access to its online portfolio. Citizens own a digital version of their identity card (eID) that contains an Idemix credential and can be managed on a smartphone. To access the library's online content, users need to use their credential to prove they live in the city. In addition, to access adult content, users need to prove that they are older than 18.

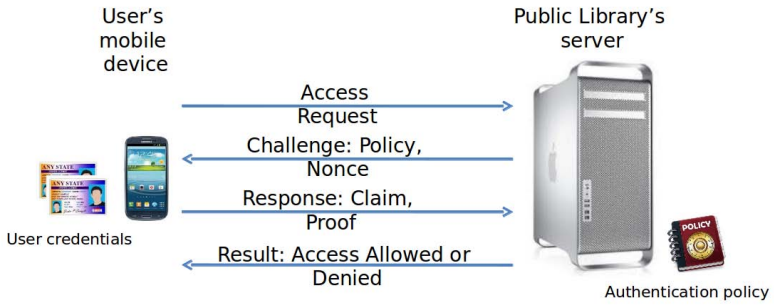

Fig. 2. Both client and server follow an interactive authentication protocol. The client uses her credentials to build a proof that satisfies the server's authentication policy.

The protocol (figure 2) follows the guidelines for authentication using anonymous credential systems set by ABC4Trust's architecture for developers [6]. First, the server will send an authentication policy and a nonce to a connecting user. The authentication policy states that the user must prove that she owns a government issued credential, that she lives in the city and that she is older then eighteen. Such authentication policies clearly improve the user's privacy,

as users can inspect these policies, learning what personal information will be revealed to whom, before any information is disclosed during the authentication. If she can do so, the user will construct a claim that satisfies the authentication policy, and create a proof of this claim. The server now verifies whether or not this claim satisfies the authentication policy and whether the proof is valid. If both tests succeed, the client may access the requested content, otherwise, access will be denied.

```
1  //INITIALIZATION
2  Priman priman = Priman.getInstance();
3  ConnectionManager cmgr = priman.getConnectionManager();
4  ServerPolicyManager spMgr = priman.getServerPolicyManager();
5  PersistenceManager pMgr = priman.getPersistenceManager();
6  CredentialManager cMan = priman.getCredentialManager();
7  priman.loadConfiguration(home.resolve("priman.conf"));
8  //INITIALIZE CONNECTION
9  ConnectionParameters cparams =
10     pMgr.load(home.resolve("clientParams.xml"));
11 Connection conn = cmgr.createConnection(cparams);
12 conn.send("ACCESS-REQUEST");
13 //GET POLICY AND NONCE
14 String polStr = (String)conn.receive();
15 Nonce nonce = (Nonce)conn.receive();
16 ServerPolicy sp=spMgr.parsePolicy(polStr);
17 //CREATE CLAIM USING USER CREDENTIALS AND POLICY
18 Claim claim = sp.getClaim(pMgr.getCredentials());
19 if(claim == null){//CANNOT SATISFY POLICY?
20     conn.send("Cannot satisfy policy");
21 } else{
22     //GENERATE/SEND PROOF
23     Proof proof = cMan.generateProof(claim,  nonce);
24     conn.send(cMan.serializeProof(proof));
25 }
```

Listing 1. Client side code of the public library application. The framework is involved in each step of the authentication. The technologies used in this applications are determined by the incoming authentication policy and the configuration policies which are read from disk. The URI 'home' is defined out of scope and points to the directory containing the configuration policies.

Listing 1 shows the Java code required to implement the client side of this application with PriMan. First, the framework is initialized, and the managers are loaded (2-7). Then, the connection manager is used to set up the connection, using the configuration policies, which are read from a file (9-11). Thereafter, an access request is sent through the newly created connection, and an authentication policy and a nonce is sent back. PriMan parses this policy (12-16).

The policy defines which types of credential are allowed in the proof and, therefore, defines the proof. At this point, we can clearly identify one of the

strengths of PriMan. *Because all technology-dependent details are absent in the application code, the developer is not concerned with what type of credential is actually used.* Whether it is a username-password, an Idemix or an X.509 credential, it will make no difference in the application code.

Using the authentication policy, a claim is created (18), which will later be used to generate a proof (23). In this scenario, the claim specifies that the user lives in a certain city, that she is older than eighteen and that she will prove this using her Idemix eID. However, depending on the authentication policy, the claim could require other facts (e.g., the fact that the user is female), or it could even specify that the user knows a password associated with a given username. While generating the proof, the credential manager will use the claim object to determine which technology and, thus, which provider is needed. However, the API also exposes the functionality to inspect the claim: which type it is, which credentials are used, what information will be disclosed, etc. . Finally, the client serializes and sends the proof to the service provider, which only needs to verify it before it can grant or deny the user access. By doing this verification, the server will only learn whether or not the client could satisfy the authentication policy.

Using PriMan, the scenario illustrated above would take little over forty lines of code to implement both the client and server side. Due to space constraints, the server-side code is not listed, but it is similar to the client-side code. Note that PriMan not only allows the developer to easily construct these complex scenarios, but also facilitates the substitution of the currently used technology for another, simply by modifying configuration policies.

4 Discussion

4.1 Design Decisions

This section discusses the design decisions and their impact with relation to *usability* from a developer's point of view and *performance*.

Usability. By providing a usable, technology-agnostic abstractions, the additional cost of integrating PETs and security technologies in applications becomes negligible. However, special attention has been invested in determining what information and operations are to be exposed by the technology-agnostic abstractions. These abstractions work in cooperation with configuration policies, which specify exactly which framework components are used and provide technology-specific parameters to these components. Developers are not required to write these policies themselves, as these policies can be written by domain experts, independently from the source code, or retrieved from an online repository. In addition, a policy creation tool can greatly increase the usability of these configuration policies for developers. Such a tool could receive as input a high level description of the desired properties of an application, like the privacy, availability or performance properties, and return as output the optimal combination of configuration policies.

Table 3. The Framework overhead and Idemix library execution time for prove and verify operations on a desktop (D) and smartphone (M). Two types of proof were evaluated: ownership of a credential with three hiden attributes (3,0) and with three revealed attributes (3,3). The mean times for 100 samples are listed in milliseconds with the standard deviation between parentheses.

	Prove D	**Verify D**	**Prove M**	**Verify M**
(3,0) Idemix	91 (5.4)	80 (4.9)	337.92 (10.52)	289.46 (8.73)
(3,0) Overhead	0.16 (0.09)	0.14 (0.02)	0.61 (0.08)	0.38 (0.04)
(3,3) Idemix	71 (2.7)	66 (3.2)	262.42 (8.63)	240.06 (4.73)
(3,3) Overhead	0.17 (0.1)	0.12 (0.01)	0.65 (0.06)	0.31 (0.03)

Performance. The framework's core consists mostly of abstract classes and interfaces. Almost all the logic is contained in the framework providers, which implement these abstract classes. They are implemented by the framework's providers, which only translate the incoming operations, defined by the abstract classes, to one or more operations of a technology-providing library (e.g., the Idemix library). Thereafter, the output of the technology-providing library is wrapped and returned in a PriMan object. This procedure is simple, and only introduces little overhead.

Table 3 shows the amount of overhead caused by the framework in milliseconds. These tests were performed on a machine with an Intel Core i5 @ 2.5 GHz with 4GB of RAM and on a Samsung Galaxy S3 smartphone. The implementation of PriMan is written in Java, uses the Idemix library (v. 2.3.4) and the tests were performed on Java SE 1.7 for Mac OS and Android version 4.1.2. For this test, the Idemix provider has been chosen because it is the most complex of PriMan's providers. It requires the most work and, hence, the most time to translate method calls from and to the Idemix library. 100 samples were taken, of which the mean execution times are listed with the standard deviation between parentheses.

As the results clearly show, the overhead caused by both the prove and verify tasks is negligible for both desktop and mobile, with no single overhead value larger than 0.2 ms and 0.7 ms respectively. In addition, the framework does not behave different on a smartphone, where each operation, including the framework's overhead, takes three to four times longer.

4.2 Comparison

CARL. We ran several benchmarks to compare the performance overhead of PriMan with the CARL framework [12]. To our knowledge, this is the only open, freely available available framework that combines authentication policies with the Idemix credential system. The setup for these tests is identical to the one used in the previous test and both frameworks use the same Idemix library. For every test, both the client-side and the server were run on the same machine.

Both the PriMan and CARL framework were used to implement an application that uses the authentication scheme similar to the one shown in figure 2.

Table 4. Performance comparison between PriMan and the CARL framework. PriMan performs its tasks consistently faster than the CARL framework. Because both frameworks use the same Idemix library, the generation and verification times are nearly the same. The mean times from 100 samples are listed in milliseconds, with the corresponding standard deviation between parentheses.

	CARL	**PriMan**
Claim/Proof generation (user)	2658 ms	799 ms
Policy parsing	1845 ms (203)	15 ms (1)
Claim generation	58 ms (3)	24 ms (2)
Idemix proof generation	755 ms (21)	760 ms (23)
Verification (server)	1152 ms	549 ms
Idemix proof verification	554 ms (19)	534 ms (12)
Policy verification	587 ms (10)	15 ms (2)

The actors in this scheme are the same: a client, who manages credentials on his device and wants to authenticate to a server. The client's six credentials are all anonymous Idemix credentials with a 2048-bit modulus and four to six attributes. When the client requests access to the server, the latter sends an authentication policy (in the CARL policy language). The client parses this policy, and tries to build a claim using the local credentials that can satisfy the initially sent authentication policy. If this is possible, a proof of this claim is created and sent to the server. This proof is verified by the server, which also checks if this proof satisfies the authentication policy.

The figures in table 4 clearly show that PriMan is consistently more efficient than CARL. 100 separate runs of the protocol were performed, of which the mean execution times are listed with the standard deviation between parentheses. The Idemix proof generation and verification times are similar for both frameworks, which is to be expected because the same idemix library is used. For the other subtasks, PriMan is more efficient due to three reasons. First, PriMan does not work with an intermediate, technology independent claim representation, which means that no "claim transformation" needs to be performed. This transformation step is performed by CARL in both the *claim generation* and *policy verification* step. Secondly, PriMan parses the policy to a simple but efficient internal representation, which means that parsing, claim generation and claim verification requires much less work. CARL, on the other hand uses a very extensive and elaborate parsing library, which causes the policy parsing time to explode. Finally, the layer between PriMan's API and the Idemix library is much thinner compared to the CARL framework. All these factors add up and result in a framework which supports the same features as CARL, but 2 to 3 times faster.

ABC4Trust. The goal of the ABC4Trust [10] project is to improve the federation and interchangeability of technologies that support trustworthy, privacy-preserving Attribute Based Credentials (ABC), or anonymous credentials. ABC4Trust has defined a common, unified architecture for ABC systems,

and develops an open reference implementation of ABC systems to deploy in actual pilots.

PriMan similarly defines abstractions that cover multiple technologies and systems under a common, unified architecture. However, the abstractions made by PriMan cover not only anonymous credential technologies, but also other kinds of credential technologies like X.509 certificates and the username-password pair. In addition, PriMan targets data storage and the protection of communication channels.

Nevertheless, PriMan and the ABC4Trust architecture are not mutually exclusive. The architecture defined by ABC4Trust can fit in the abstractions of PriMan, i.e., a provider can be implemented which implements this architecture. The advantage of this is that applications written with PriMan can be made compatible with systems based on the ABC4Trust architecture. Because Priman does not use its own technology-independent format, but wraps a translation layer around existing formats, messages from/to an ABC4Trust application can be interpreted by a PriMan application. The framework thus remains compatible with the technologies it implements.

5 Related Work

The PriMan framework is inspired by a framework developed within the ADAPID project [13] ("Advanced Applications for the Belgian eID card"). This framework offers a unified interface for the Belgian eID card and X.509 certificates. In contrast to ADAPID, the focus of PriMan is to facilitate the integration of PETs and security technologies in applications. In addition, PriMan also focusses on performance, usability and support for mobile devices.

Much work has already been performed in the field of privacy preserving identity management. The requirements of such systems are investigated in [14–16]. The European project ABC4Trust [10, 6] builds on this work by defining a common,unified architecture for anonymous credential systems. This work is validated by several pilot projects in which authentication is performed using anonymous credential systems.

Privacy-friendly identity management should allow for anonymous yet accountable transactions, which can be provided by using anonymous credential systems. First, the client and the service provider must agree on an authentication policy. Various privacy preserving policy languages for attribute based credentials have been proposed [5, 6, 17]. After receiving a policy, the user employs her credentials to prove that the information contained therein satisfies the authentication policy. Second, further constraints can even be made on the downstream usage of this information by the service provider or other third parties [6, 17, 18].

An authentication framework has been made available that combines the Idemix credential system with the CARL authentication policy [5], but this framework introduces a lot of overhead and is not suitable for mobile devices. PriMan supports

the same CARL authentication policies, but it introduces almost no computational overhead and it is optimized for mobile devices. The PriMan implementation of CARL supports authentication using Idemix, but also authentication using all other supported credential systems. Due to PriMan's technology-agnostic API it does not matter which technology is used underneath as every technology provides an implementation for the same set of operations.

A similar project, Opaak [19], implements the Idemix credential system and offers a set of authentication protocols which developers can easily integrate in their applications. Like PriMan, Opaak recognises the importance of mobile devices and supports Android. However, the scope of PriMan is bigger than that of Opaak. PriMan aims at offering the tools to build secure, privacy-friendly applications from start to finish; from credential storage mechanisms and the set-up of connections, to credential systems and various types of policies.

6 Conclusions and Future Work

This paper presents PriMan, a flexible framework that facilitates the development of secure, privacy-preserving applications by offering high-level technology-agnostic abstractions. Through its configuration policies, PriMan allows applications to seamlessly switch between different technologies. In addition, PriMan's API removes the need for developers to learn the specific details of a new technology. Furthermore, little work is required to integrate a new technology in the framework because of its flexible and extensible design.

PriMan has an efficient and modular implementation which allows it to run on a wide range of devices. It supports mobile devices and brings, among other components, a full-fledged anonymous credential system to these platforms.

We have implemented privacy-preserving authentication policies that can be used in combination with the credential systems supported by PriMan. These policies allow users to learn and review what information they will reveal during an authentication. The performance results show that PriMan's design introduces only a small amount of performance overhead.

One of the goals of PriMan is to facilitate the development of advanced, but privacy preserving application. The framework accomplishes this by offering a simple and intuitive technology-agnostic API. It is our intention to further investigate and validate the usability of this API, and its accompanying configuration policies. Furthermore, the framework could support various other policy types with which access control can be regulated and framework tasks automated. E.g., a user could specify what personal information contained in credentials can be revealed to which entities. The vocabulary and grammar of such a language, and the usability thereof are still open questions.

Acknowledgements. This research was funded by the IWT-SBO Project MobCom: A Mobile Companion (https://www.mobcom.org).

References

1. OWASP: OWASP Mobile Security Project — Top Ten Mobile Risks (2013), https://www.owasp.org/index.php/OWASP_Mobile_Security_Project
2. MobCom Project: MobCom: A Mobile Companion (2013), http://www.mobcom.org/
3. Camenisch, J.L., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 93–118. Springer, Heidelberg (2001)
4. Specification of the Identity Mixer cryptographic library – version 2.3.2, IBM Research – Zurich (2010)
5. Camenisch, J., Mödersheim, S., Neven, G., Preiss, F.S., Sommer, D.: A card requirements language enabling privacy-preserving access control. ACM (2010)
6. Camenisch, J., Krontiris, I., Lehmann, A., Neven, G., Paquin, C., Rannenberg, K.: H2. 1-abc4trust architecture for developers. Heartbeat (2012)
7. Housley, R., Ford, W., Polk, W., Solo, D.: Rfc 2459 - internet x.509 public key infrastructure certificate and CRL profile (1999)
8. Camenisch, J.L., Lysyanskaya, A.: Signature schemes and anonymous credentials from bilinear maps. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 56–72. Springer, Heidelberg (2004)
9. Rackoff, C., Simon, D.R.: Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 433–444. Springer, Heidelberg (1992)
10. ABC4Trust Project: ABC4Trus EU Project - Official Website (2013), https://www.abc4trust.eu/
11. Housley, R.: RFC 5652 - Cryptographic Message Syntax, CMS (2009)
12. Preiss, F.-S.: Credential-Based Authentication Framework With Built-In Ready-To-Use Identity Mixer Support (2011), http://www.zurich.ibm.com/~frp/com.ibm.zurich.authn.cb/
13. adapID Project: advanced applications for electronic IDentity cards in Flanders (2009), http://www.cosic.esat.kuleuven.be/adapid/
14. Camenisch, J., Shelat, A., Sommer, D., Fischer-Hübner, S., Hansen, M., Krasemann, H., Lacoste, G., Leenes, R., Tseng, J.: Privacy and identity management for everyone. ACM (2005)
15. PrimeLife Project: PrimeLife - Bringing sustainable privacy and identity management to future networks and services (2013), http://primelife.ercim.eu/
16. Hansen, M., Berlich, P., Camenisch, J., Clauß, S., Pfitzmann, A., Waidner, M.: Privacy-enhancing identity management. Information Security Technical Report (2004)
17. Hansen, M., Schwartz, A., Cooper, A.: Privacy and identity management. IEEE Security Privacy 6(2), 38–45 (2008)
18. Bichsel, P., Camenisch, J., Preiss, F.S.: A comprehensive framework enabling data-minimizing authentication. ACM (2011)
19. Maganis, G., Shi, E., Chen, H., Song, D.: Opaak: using mobile phones to limit anonymous identities online. ACM (2012)