Implementation of a pseudonym-based signature scheme with bilinear pairings on Android

Leonardo Oliveira, Victor Sucasas, Georgios Mantas, and Jonathan Rodriguez

Instituto de Telecomunicações, Aveiro, Portugal

Abstract. Privacy preservation is of paramount importance in the emerging smart city scenario, where numerous and diverse online services will be accessed by users through their mobile or wearable devices. In this scenario, service provides or eavesdroppers can track users activities, location, and interactions with other users, which may discourage citizens from accessing smart city services. Pseudonym-based systems have been proposed as an efficient solution to provide identity confidentiality, and more concretely pseudonym-based signature schemes have been suggested as an effective means to authenticate entities and messages privately. In this paper we describe our implementation of a pseudonym-based signature scheme, based on bilinear-pairings. Concretely, our implementation consists of an Android application that enables users to authenticate messages under self-generated pseudonyms, while still enabling anonymity revocation by a trusted third party in case of misbehavior. The paper presents a description of the implementation, performance results, and it also describes the use cases for which it was designed.

Key words: privacy-preserving; mobile applications; bilinear pairings

1 Introduction

Privacy-preserving solutions are required to address the users' privacy concerns and foster a rapid penetration of smart city applications in the real world. Providing anonymous and secure communication between mobile applications and the Smart City infrastructure is cornerstone in this scenario, and current technology faces the problem that traditional Public Key Management systems do not provide privacy. Moreover, the current digital anonymous credential systems in the state-of-the-art are too complex to be applied in wearable or mobile devices due to its computation delay and communication overhead.

Previous research efforts have pointed out several privacy issues that demand novel solutions [1]. For example, Public Key Infrastructure is not suitable for privacy preservation, and new Key Management Systems should be implemented [2]. In this framework, pseudonym-based systems have been suggested as promising research direction, since pseudonyms can be used to hide the users' real identity [3], and still can enable revocation mechanisms in case of misbehavior. Pseudonyms are issued by a certification authority and they can be used to sign messages, thus providing authentication and integrity [4][5]. Also, pseudonyms can be renewed on demand to avoid linkability of different data transactions, which could eventually allow eavesdroppers to infer the users identities. This pseudonym renewal requires a permanent contact with the Certification Authority, except systems that provide pseudonym self-generation, such as the work in [6], which has been followed by other works such as [7][8][9]. This paper describes the implementation of the system described in [7], which was originally designed for a Vehicular Network, and it has been now adapted into a mobile application.

The rest of the paper is structured as follows: Section 2 describes the system model, the system functionality, and some use case scenarios envisaged for this system; Section 3 provides a brief description of the cryptosystem; Section 4 provides the implementation details; Section 5 shows a performance evaluation; and finally, section 6 concludes this paper.

2 System Model

The system is composed of a set of smartphones (clients) running a mobile app that enables privacy-preserving authentication between each other. Namely, clients can send signed messages in an anonymous manner, i.e. the receiver can trust the sender of the message but it is not feasible to figure out the identity of the original sender (except if traffic analysis techniques are applied at the network level, which is out of the scope of this work and it should be addressed by appropriate countermeasures). These clients can sign the transmitted messages with their self-generated pseudonyms, which cannot be linked to the users real identities. Moreover, different pseudonyms from the same user can also not be linked to each other. Hence, all pseudonyms shared in the network will be statistically indistinguishable.

A message is any type of information that is being shared between the clients. The system also enables the transmissions of null messages, where the proposed message authentication application is used with the sole purpose of authenticating the client and not any message in particular. Section 2.2 provides a real scenario exemplifying the type of messages that could be exchanged.

Apart from clients, the system is also composed of a certification authority (CA) and verifiers. The CA is in charge of generating public parameters required for the cryptosystem implemented in the proposed system, and the generation of credentials for legitimate users. The CA is also in charge of revoking users credentials in case of misbehavior. The verifier is any entity receiving and validating a signed message, for this validation the verifier is not required to have a valid credential, only the public parameters generated and distributed by the CA are sufficient to validate messages.

2.1 System Functionalities

The system provides mechanisms to: i) enable the CA to provide certificates to trusted clients, so only legitimate clients can successfully send signed messages;

and ii) enables clients to use the certificates to generate pseudonyms on demand and sign messages with such pseudonyms; iii) enables verifiers to validate signed messages from legitimate clients and discard messages from dishonest clients that are not provided with valid certificates. The proposed system assures the privacy of each sender (client), since the verifiers cannot link the pseudonyms to the users real identities or to other pseudonyms of the same client. The fugure 1 illustrates the interaction between the system's entities, that will be described bellow.

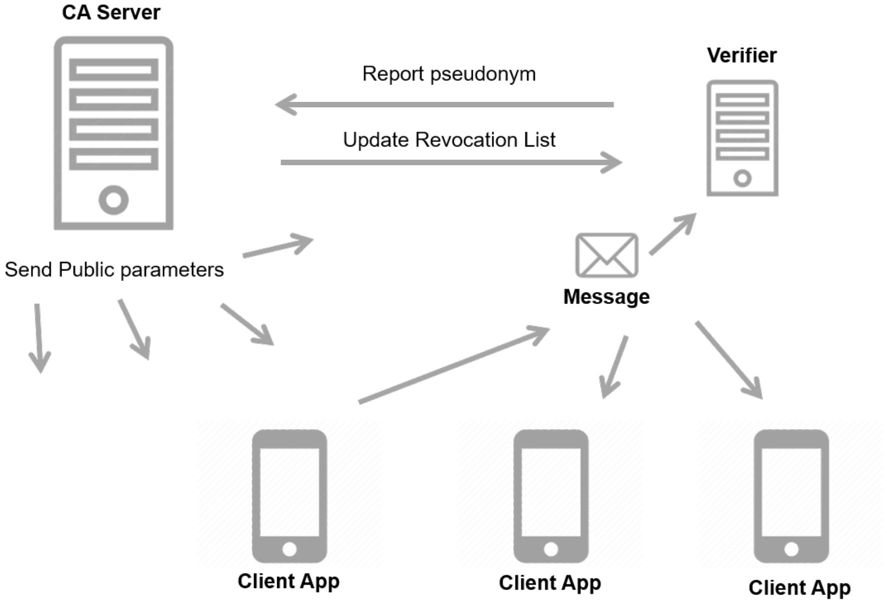


Fig. 1. The interactions between the entities in the system. Any client can act as a verifier.

The system also enables conditional privacy, i.e. it allows the trusted authority (the CA) to identify the origin of a message in exceptional situations. This can occur when a client is malfunctioning or misbehaving, which could lead to the transmission of signed messages with misleading information. In such a case, the CA can track the pseudonym used by the sender of these messages to the users real identity. Moreover, the CA can include the future pseudonyms of the blacklisted misuser in a Revocation List (RL), enabling other users to ignore future messages from the same sender.

Before being capable of sending messages, a client needs to be certified. This happens in a registration process that starts in a Certified Authority (CA). The CA holds all the information of the clients that belong to the system, including, of course, their real identity. For each registered client, the CA can issue credentials. Those credentials can be used by clients to authenticate themselves or authenticate transmitted messages towards a verifier.

When receiving a message, the verifier validates its authenticity. The verification can be done by other clients or verifying entities deprived of credentials. A message is valid if it comes from a valid client, which only happens with clients that use a credential issued from the CA. Nonetheless, the verifier still cannot know the origin of message, he can just perform the verification algorithm needed to validate it. Verification algorithm can be performed by any entity holding the public parameters generated by the CA.

The system is time based and requires loose synchronization between entities. Namely, clients can renew all the pseudonyms updating a public parameter that can be self-generated. In the verification process, a time restriction public key must also be renewed and distributed by the CA to all potential verifiers. Hence, the system is divided in time slots, in which the CA and the users update a restriction key and the pseudonyms respectively. The revocation list must also be renewed and distributed by the CA with all blacklist pseudonyms.

2.2 Use-case Scenarios

The proposed system can be used in any use case scenario requiring privacy-preserving authentication. The proposed system however has been designed and will be tested in a real environment in a crowdsourcing application, where users collect information about the surrounding environment and transmit this information to a back end server. Namely, users of the developed app running the client side of the system will send messages with information like air pollution, noise level and their landmarks. The information will be gathered by sensor integrated in the mobile devices or externalized and connected to the users smartphones through Bluetooth. These messages, after being signed with a pseudonym, will be sent to an data aggregator, i.e. a smart-city server, where they would be validated while preserving the senders privacy. Thus, this server would act as a verifier of the transmitted information. The stored information will perform data analysis to generate statistical information about air and noise quality of different locations. The data aggregator (verifier) will also be in charge of detecting misbehavior (e.g. users transmitting misleading or adulterated data) and eventually contact the CA to report the affected pseudonyms, hence triggering the anonymity revocation mechanism in the CA. After that, all pseudonyms from blacklist clients would be revoked and the client would be ignored in the future results. It is worth commenting that a number of research works have already proposed effective means to detect data outliers that could be used in the proposed use case scenario.

The proposed system is also suitable for a scenario where several clients need to authenticate themselves to get access to given service, while preserving their privacy. In that case, they would send empty messages, where the only important part would be the signature itself, that could be validated by the service provider. The service provider, playing the role of verifier, would grant access to authorized clients without the knowledge of their real identities. Cases of misbehavior, like the usage of the same pseudonym several times in a short space of time, could trigger anonymity revocation.

3 Pseudonym-based signature scheme

Although we would like to refer interested readers to the works [6] and [7] for more details on the pseudonym based signature scheme and the conditional privacy-preserving system respectively, in this section we describe briefly the mathematical operations included in our implementation. The system is devided in the following logical blocks: i) parameter generation; ii) credential generation; iii) pseudonym generation; iv) message signing; and v) signature verification.

For the public parameter generation, the CA performs the following steps, and publishes the tuple $(G_1, G_2, P, H_1, H_2, H_3, P, W)$ and the time variant tuple (Q_i, W_i). The time variant values must be computed again in each time slot. The value s is the CA secret key.

1. The CA selects two cyclic groups of prime order p , G_1 and G_2, in which the discrete logarithm problem is hard and with an efficient bilinear map e such that $e : G_1 \times G_1 \rightarrow G_2$.
2. The CA also picks two cryptographic hash functions $H_1 : \{0, 1\}^* \rightarrow G_1$ and $H_2, H_3 : \{0, 1\}^* \rightarrow Z_p$ (where Z_p is the multiplicative group).
3. The CA selects P as generator of G_1.
4. The CA computes a time variant public key $Q_i = H_1(T(time))$.
5. The CA selects a secret $s \in_R Z_p$.
6. The CA obtains $W = sP$ and a restriction key $W_i = sQ_i$

To generate a credential for a client, the CA performs:

1. The CA selects randomly $\mu \in Z_p$.
2. The CA computes the secret value as $Su = P\frac{1}{(s+\mu)}$.
3. The CA sends the user the credential (μ, Su).

It is worth commenting that in the implemented system, every user receives 10 credentials where the μ values are obtained with a hash chain $\mu_i = H_3^i(\mu)$ for $i = \{0, ..., 10\}$.

Then, each user/client can self-generate pseudonyms using these credentials:

1. The client computes $Q_i = H_1(T(time))$.
2. The client computes a pseudonym for each of the 10 credentials by computing $Pseu_j = \mu_j Q_i$ for $i = \{1, ...10\}$

Clients provided with a valid credential can sign a message m using their self-generated pseudonyms:

1. The client selects α, r, $r' \in_R Z_p$.
2. The client computes $T = \alpha Su$, $R_{G_1} = rQ_i$ and $R = e(Q_i, P)^{r'}$.

3. The client computes $c = H_2(M||T||R_{G_1}||R||Pseu_j||T(time))$, where the operator $||$ represents concatenation.
4. The client computes $s_1 = \alpha c + r'$ and $s_2 = \mu_j c + r$.

The signature of m with the pseudonym $Pseu_j$ is the tuple $\sigma = (T, c, s_1, s_2)$. Which can be verified by any entity having the public parameters by performing the following steps:

1. The verifier obtains $Q_i = H_1(T(time))$.
2. The verifier computes $R'_{G_1} = s_2 Q_i - cPseu_j$ and $R' = e(Q_i, P)^{s_1}/e(Pseu_j + W_i, T)^c$.
3. The verifier computes $c' = H_2(M||T||R'_{G_1}||R'||Pseu_j||T(time))$.
4. The verifier considers the signature valid if $c' = c$ holds.

These steps have been implemented into the described system in the different entities defined in the system model. Due to space constraints we do not detail here preliminary mathematical definitions or proofs of security. More details can be found in the works [6] and [7].

4 Implementation Details

Our implementation of the system is based in Java. This enables to compilation and deployment in different contexts. In the proposed scenario, it is deployed in a mobile app, running in Android, where the client side of the system is placed.

In what concerts to the algorithmic part of the system, we used Java Pairing-Based Cryptography Library (JPBC) [10], a Java library that allowed us to implement all the cryptographic calculations related to bilinear maps. This library can run out of the box on Android 2.1+, which was verified in our implementation.The system is mainly split into the entities described in the system model: The CA, the Verifier and the Client (The clients can also act as verifiers when receiving signed messages). The following subsections describe the implementation details of the different mechanisms of the proposed system:

4.1 Public Parameter Generation

Provides all the necessary cryptographic parameters used to perform several calculations in different steps of the system. These parameters are available publicly and distributed by the CA. It gives access to the cyclic groups G1 and G2, the multiplicative group of prime order p, Zp, the generator P, the permanent public key W, time restriction public key Wi, the current time slot, the time variant parameter and the hash functions H1, H2 and H3.

This class is wrapped by other classes, and it is never directly manipulated in the interface of the implementation. In fact, this layer of the package developed is the lower one. All the mathematical part is abstracted in the three main entities described below. The CA Server, the Verifier and the client are the outer layers of the system, representing the main logical entities.

For a rationale of the implementation, the lines below describe the enclosed modules, represented in the figure 4.1.

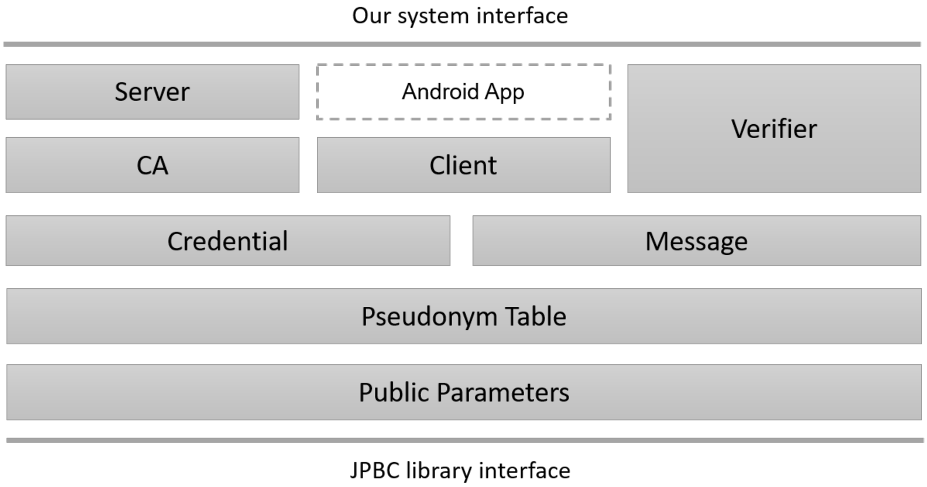


Fig. 2. The layered interconnection between the components, from thee JPBC library to our system interface.

4.2 CA Server

The CA Server is the central part of the system. It contains an instance of the CA, that handles all the confidential information of the system required to issue credentials and revoke anonymity. Once again, the server abstracts all the interactions with the CA instance. When it starts running, it is available to answer to requests of Public Parameters, Credentials - by the clients - and exclusion of pseudonyms - by verifiers. The CA instance, when created, generates the original copy of the Public Parameters, described above. Also, the number of time slots and the number of pseudonyms by time slot are defined here. The CA can generate a Credential for each client. The CA class can generate as many credentials as the server needs. Each Credential is later used by the client to generate its pseudonyms. Despite of this, the server also must have a list of all the pseudonyms to enable anonymity revocation if needed. Because of this, the pseudonyms are also generated in the server side using a copy of the credentials issued by the CA. The server uses other class to handle this, called Pseudonym Table. The Pseudonym Table provides a systematic manner to store lists of pseudonyms by time slot. In theory, it is a table of N pseudonyms by T time slots.

The class provides an interface to automatically manipulate those pseudonyms, including its generation. When the server or also the verifier and client - needs to generate a list of pseudonyms, it provides the correspondent credential and time slot.

4.3 Client

The Client holds its information, so it has a Credential, the Public Parameters and a Pseudonym Table. The first time the client runs, he connects to the CA server, asking for the Public Parameters and a credential. After, he generates its set of pseudonyms in the Pseudonym Table. The pseudonyms are linked to each time slot and they can be generated on demand. When the client needs to send a message, he uses a specific method to do it, sendMessage(). The method works using some dependencies: the message is first signed using the pseudonym, and after it is included inside a packet that also holds the signature, the pseudonym and the message itself. This packet is then sent to the verifier.

4.4 Verifier

The Verifier, when initializing, has to get the Public Parameters from the CA Server and an updated version of the pseudonym blacklist (also called revocation list) the list of blocked pseudonyms. This blacklist is refreshed at least when there is a new time slot. With that completed, the Verifier is ready to validate received messages. When it receives a validation of a message, first it checks the blacklist to verify if the pseudonym is not banned. If it is, it returns that the message is invalid. If not, it performs the calculations to check the validity of the message. According to the calculations, it returns whether the message is valid or not. If for some reason the verifier detects a suspicious activity for a given pseudonym, for instance, being used too many times in a small interval of time, it can report it to the server.

The figure 3 shows the implementation's class diagram, generated with ObjectAid UML Explorer, in the Eclipse IDE.

In this analysis, it is missing some implementation details of the client side. It is done in a mobile app, running on Android 4.4+. The android app uses the Client class of the package. For demonstration purposes, the app developed allows the client to connect to the server, receiving the credential and public parameters, and manually synchronize the time slots and generate pseudonyms. The pseudonyms can be selected manually to send messages. If for some reason the time slot is outdated, all the attempts to send messages will fail, so the synchronization must be accomplished.

In the figure 4, we represent the pseudonyms in the application side. A pseudonym is a point in an elliptic curve. To be exact, in our implementation they are portions of data described in 128 bytes. So, rather than numbers, the app shows something more visual and meaningful. The pseudonyms are mapped to a grid of colours, where they can be selected. The client can send a message using the desired pseudonym. The quantity of pixels represents the amount of possible variations that pseudonyms can have.

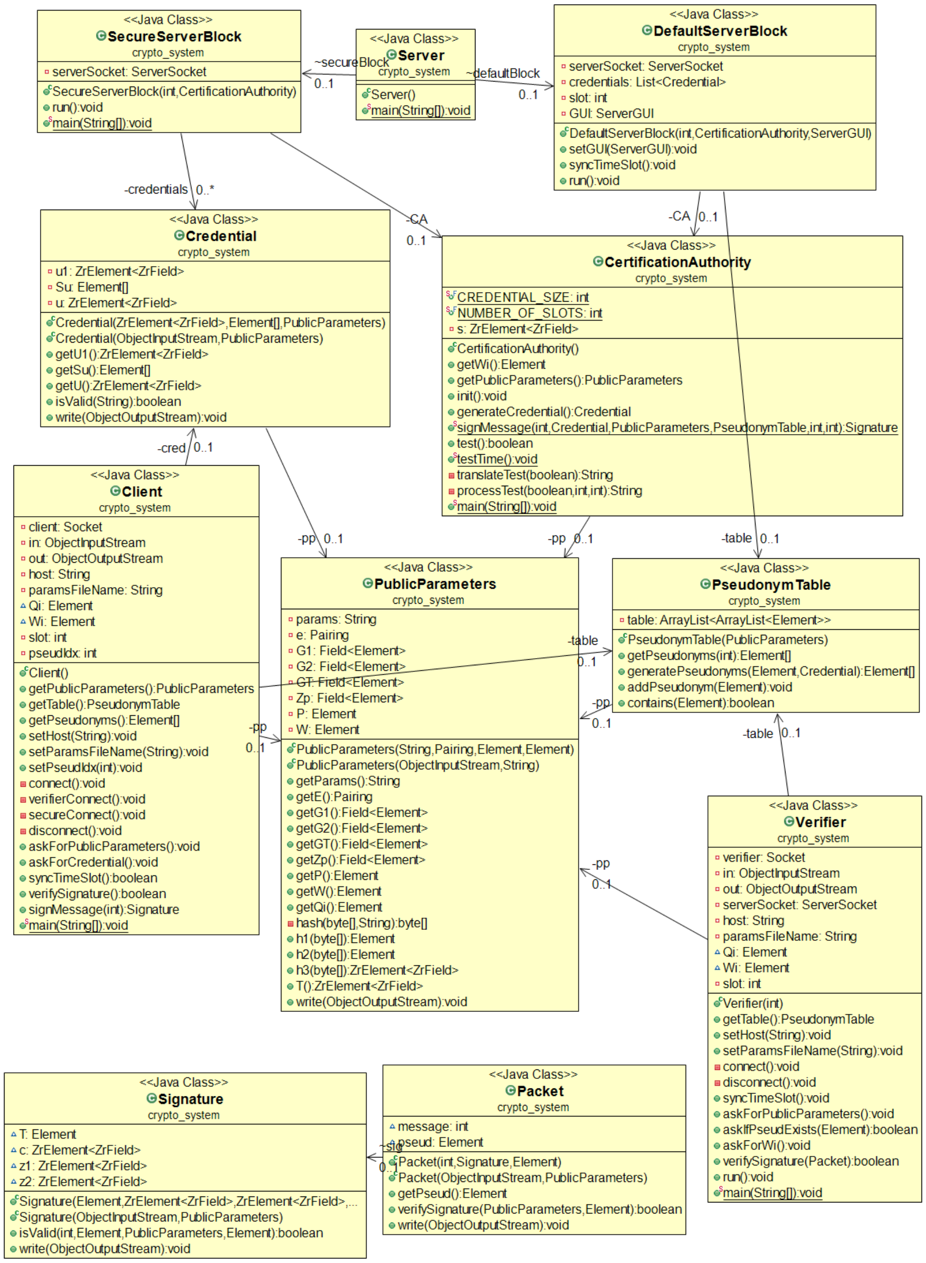
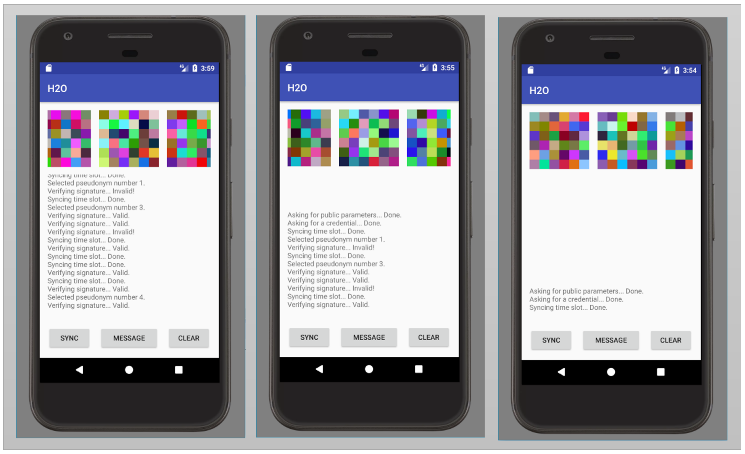


Fig. 3. System's class diagram, generated with ObjectAid UML Explorer, a tool available in the Eclipse IDE



```
Syncing time slot... Done.
Selected pseudonym number 1.
Verifying signature... Invalid!
Syncing time slot... Done.
Selected pseudonym number 3.
Verifying signature... Valid.
Verifying signature... Valid.
Verifying signature... Invalid!
Syncing time slot... Done.
Verifying signature... Valid.
Syncing time slot... Done.
Syncing time slot... Done.
Verifying signature... Valid.
Syncing time slot... Done.
Verifying signature... Valid.
Selected pseudonym number 4.
Verifying signature... Valid.
```

```
Asking for public parameters... Done.
Asking for a credential... Done.
Syncing time slot... Done.
Selected pseudonym number 1.
Verifying signature... Invalid!
Syncing time slot... Done.
Selected pseudonym number 3.
Verifying signature... Valid.
Verifying signature... Valid.
Verifying signature... Invalid!
Syncing time slot... Done.
Verifying signature... Valid.
```

```
Asking for public parameters... Done.
Asking for a credential... Done.
Syncing time slot... Done.
```

Fig. 4. Some screenshots to the mobile app. The client's available pseudonyms are represented on the top, and the actions are logged below.

5 Results

For performance evaluation, we have tested the performance in both the server and client side. Opposed to what happens in the server side, the clients resources are more limited since it runs in a mobile device. Due to the strong component of algorithmic computations performed in this cryptosystem, computations on a mobile device can be more time consuming.

We measured the lapsed times in the server side, in an Intel Core i5-6200 CPU, executing the following operations: i) initializing the server; ii) issuing a credential; iii) signing a message; iv) verifying a signature; and v) generating a pseudonym. The server takes 207ms to initialize, which involves generating the private and public parameters. For each credential generation, it takes 373ms. The signature generation takes 60ms, the signature verification takes 94ms. Generating a pseudonym takes 50ms. For a scenario of 1 million users, with 10 pseudonyms per time slot, it would take more than 8min to precompute all the pseudonyms. This pseudonym precomputation is advisable to speed up the anonymity revocation mechanism since the time of searching for a pseudonym in a list is negligible. For example, searching over 500.000 pseudonyms takes less than 2ms.

These time measurements were also performed in a smartphone (Samsung Galaxy S6), running a Exynos 7420 Octa CPU. The computations in the client side are: i) signing a message; ii) verifying a signature; and iii) generating a pseudonym. In general, it resulted in around 10x more time spent when compared to server side. The full results are presented in the figure 5. In terms of memory, the size of each component is negligible. Despite of this, it is worth analysing the amount of data transmitted between entities. The public parameters have 362 bytes. A credential has 1427 bytes (the credentials implemented are composed

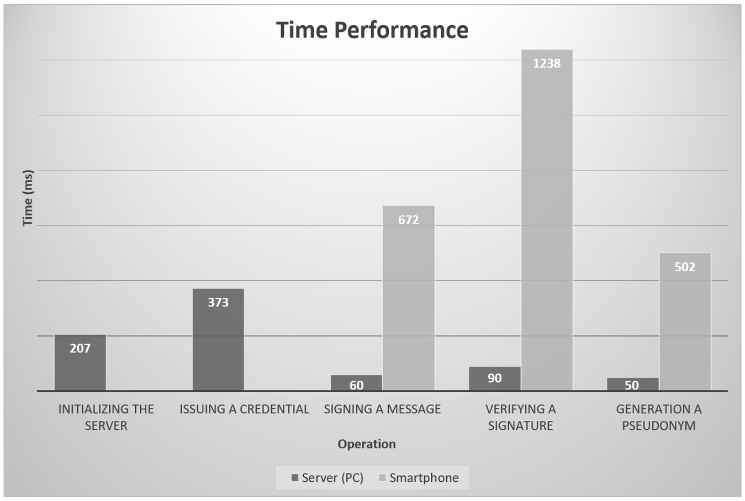


Fig. 5. Time performance in both the server and client side

by 10 secret values in G2 and 1 in G1). A packet composed by a signed message has 383 bytes plus the size of the message itself. Each pseudonym has 128 bytes. In the server side, for a scenario with 1 million users, with 10 pseudonyms per time slot, it would be necessary 1.28GBytes to precompute all the pseudonyms. The figure 5 illustrates the proportion between the size of such entities.

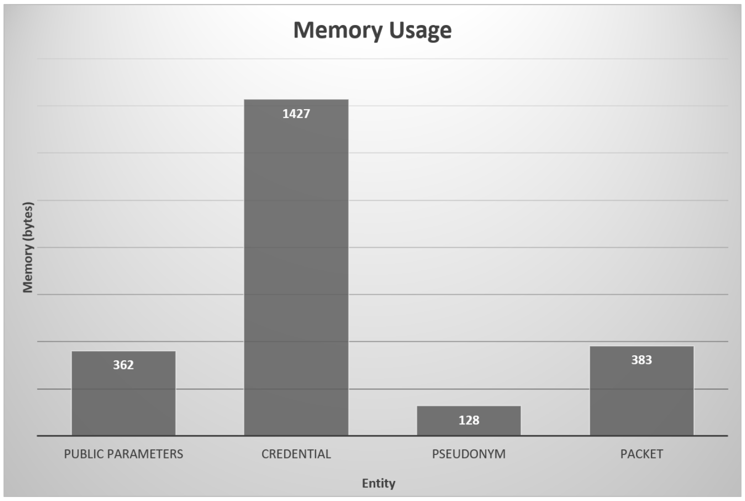


Fig. 6. Memory usage in both the server and client side

6 Conclusion

In this paper we describe the implementation of a privacy-preserving system based on a pseudonym-based signature scheme that relies on bilinear pairings. Despite of the heavy algorithmic computations associated to bilinear pairing operations, it was possible to obtain a system with a small footprint in terms of time performance and memory consumption, with the integration of JPBC library, running on Java. The same results were also promising when obtained in the client side (implemented in an Android smartphone), proving that bilinear pairings can be efficiently implemented on mobile devices for some specific use cases.

7 Acknowledgement

This work is supported by the European Structural Investment Funds, through CENTRO 2020 [Project Nr. 017785(CENTRO-01-0247-FEDER-017785)] and EU-H2020-MSCA-ITN-2016 SECRET-722424.

References

1. Li, Ming; Wenjing Lou; Kui Ren, "Data security and privacy in wireless body area networks," Wireless Communications, IEEE , vol.17, no.1, pp.51,58, February 2010.
2. Wasef, A; Rongxing Lu; Xiaodong Lin; Xuemin Shen, "Complementing public key infrastructure to secure vehicular ad hoc networks [Security and Privacy in Emerging Wireless Networks]," Wireless Communications, IEEE , vol.17, no.5, pp.22,28, October 2010
3. Dan Boneh and Matthew K. Franklin. 2001. Identity-Based Encryption from the Weil Pairing. In Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology (CRYPTO 01), Joe Kilian (Ed.). Springer-Verlag, London, UK, UK, 213-229.
4. Jiun-Long Huang, Lo-Yao Yeh, and Hung-Yu Chien. Abaka: An anonymous batch authenticated and key agreement scheme for value added services in vehicular ad hoc networks. Vehicular Technology, IEEE Transactions on, 60(1):248262, Jan 2011.
5. Rongxing Lu, Xiaodong Lin, Zhiguo Shi, and X.S. Shen. A lightweight conditional privacy-preservation protocol for vehicular traffic-monitoring systems. Intelligent Systems, IEEE, 28(3):6265, May 2013.
6. Yong Zhang and Jun-Liang Chen. A delegation solution for universal identity management in soa. Services Computing, IEEE Transactions on, 4(1):7081, Jan 2011.
7. Victor Sucasas, Georgios Mantas, Firooz B. Saghezchi, Ayman Radwan and Jonathan Rodriguez. An Autonomous Privacy-Preserving Authentication Scheme for Intelligent Transportation Systems. Elsevier, Computers and Security. 2016
8. Victor Sucasas, Georgios Mantas, Ayman Radwan and Jonathan Rodriguez. An OAuth2-based Protocol with Strong User Privacy Preservation for Smart City Mobile e-Health Apps. IEEE ICC 2016
9. Victor Sucasas, Georgios Mantas, Ayman Radwan, Jonathan Rodriguez: A Lightweight Privacy-Preserving OAuth2-Based Protocol for Smart City Mobile Apps. GLOBECOM Workshops 2016
10. Angelo De Caro and Vincenzo Iovino. jPBC: Java pairing based cryptography. Proceedings of the 16th IEEE Symposium on Computers and Communications, ISCC 2011.