J Supercomput (2014) 70:709–720
DOI 10.1007/s11227-014-1165-2

A decentralized protocol for mobile control access

J. A. Alvarez Bermejo · M. A. Lodroman · J. A. Lopez-Ramos

Published online: 18 March 2014

Abstract In this paper we use the Diffie–Hellman key exchange protocol to introduce a decentralized key agreement protocol based on elliptic curves. We do not use any public key infrastructure, which makes it suitable for light devices with low computational and storage capabilities. Thus mobile devices can directly authorize other mobile devices to exchange keys in order to get access to a service or system, in a secure and efficient manner.

Keywords Elliptic curves · Diffie–Hellman · Mobile user authentication · Content protection

1 Introduction

Authentication is the tool that we use to access any service or system that provides a set of them. It can be implemented using traditional public key cryptography [4,9], although due to the huge growth of applications for mobile devices, mobile authentication in an insecure channel is nowadays an important issue since these devices hold low computational and storage resources and thus traditional cryptosystems using modular exponentiation are rarely implemented. These are being substituted by the so-called elliptic curve cryptosystems (ECC) introduced in [6] and [7], mainly due to new standards for public key cryptography recommended in [8]. ECC has significant advantages with regard to key sizes and computational cost, which makes it a

J. A. Alvarez Bermejo (✉)
Department of Informatics, University of Almeria, 04120 Almería, Spain
e-mail: jaberme@ual.es

M. A. Lodroman · J. A. Lopez-Ramos
Department of Mathematics, University of Almeria, 04120 Almería, Spain

good alternative to authentication protocols for mobile devices. However, a traditional authentication protocol based on ECC also needs a public key infrastructure (PKI) to maintain certificates. As the numbers of users increase, the numbers of certificates and storage requirements also increase. Alternatives to PKI, like identity-based key agreement protocols have been considered in [1,2,10] and [11]. Their main advantage is that they do not need to store users' public key and certificates, which simplify certificate management. Main weakness is that users' private key must be generated by a unique entity, usually called the Key Generator Center. These protocols are based on Weil and Tate pairing techniques that have been shown to require a considerably higher computational cost than modulo arithmetic [14] and thus, also more costly than arithmetic on elliptic curves. Most are shown to not be secure against certain forms of attack: in [2] it is shown that the protocol given in [11] fails in several security aspects. Later, in [3], new security flaws were presented as affecting [2]. Sun and Hsieh showed in [12] that the protocol introduced in [10] fails against a man in the middle attack, which is, in general an important menace for those protocols that do not use certificates that authenticate information transmitted. The alternative we are introducing in this paper is a key agreement protocol based on the Diffie–Hellman problem for elliptic curves. Participants can play different roles and trust is based on these roles. The protocol has three main phases: Key Generation, where an authorized entity generates some information that will allow to build the key for the requester; Key Registration, where the user and the guard that grants access to the service agreed on a shared key in an authenticated and secure manner; and Authentication, where the guard grants the access in case of right authentication. In the Key Registration stage, we are using a three-party Diffie–Hellman key exchange protocol, even though just the two directly involved parties, a user and a guard, can access the key. We show that forging the identity of every entity is highly improbable. This avoids man in the middle attacks.

The paper is structured as follows: In the second section we introduce the protocol and its phases, roles and rules. In the third section we show how the final key is built in a secure and authenticated manner. In the fourth section we describe an implementation and some preliminary performance results. In the fifth section we offer our conclusions.

2 The protocol

Participants in this protocol may play four different roles: administrator, super user, visitor user and guard. The protocol is divided into five different phases: Initialization, where every guard is matched to at least one administrator; Key Generation, where a user demands a key to access a service from anyone else authorized to provide this service; Key Registration, where the generated key is registered with an appropriate guard; Authentication, where a registered key is used to authenticate and grant a service, and Key Revocation, where a registered key is revoked due to any security issue. There exist two basic rules in the protocol: on one hand we have the key owners, that will be administrator, super users and the visitor users; on the other hand, the key verifiers, that will be the guards. Among the keys we will have three different grades: those belonging to the administrator and super user will be called master keys and the

Table 1 Notation used in the protocol

Notation	Meaning
$\mathcal{A}$	Administrator status
$\mathcal{G}$	Guard status
$\mathcal{U}$	User status
$\mathcal{S}$	Superuser status
$ID_{\mathcal{S}}$	Identity of $\mathcal{S}$
KeyGen|KeyReg|KeyAuth	Key {Generation|Registration|Authentication} message tags
K_{AB}	Key shared by A and B
$E_K(x) \mid D_K(x)$	Encryption | Decryption of content x using key K
$VT_{\mathcal{U}}$	Valid time for keys involving user U
T_C	Valid time for message C

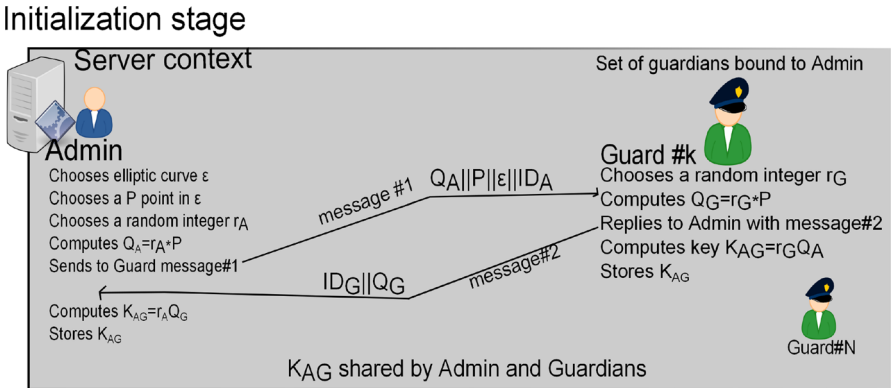

Fig. 1 Initialization stage of the protocol

others will be visitor keys. Thus the roles played for every participant will determine the attributes of each one, i.e., access to a determined system, and/or capability to generate other keys. Concerning this second issue, those holding a master key will be allowed to generate another master key or a visitor key, and those holding a visitor key will only be allowed to access the corresponding service. Table 1 summarizes the notation used in this section.

2.1 Initialization

As previously noted, each guard system $\mathcal{G}$ will match at least to one administrator $\mathcal{A}$. The initialization phase will be developed under a secure channel and will consist of the steps described in Fig. 1, where || denotes concatenation and Admin and Guardians are computing entities that operate in a high-performance computational context.

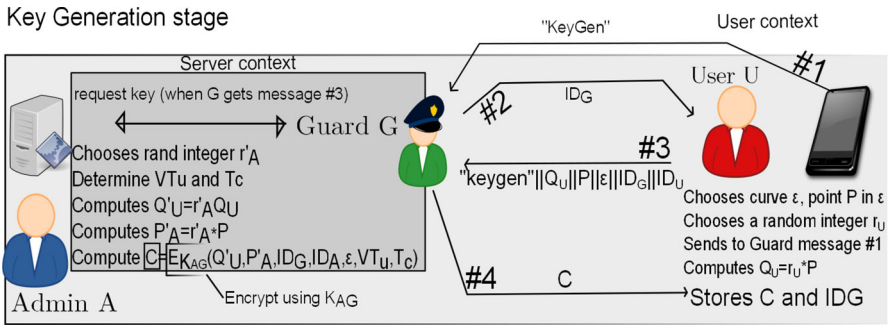

Fig. 2 Steps to generate a key

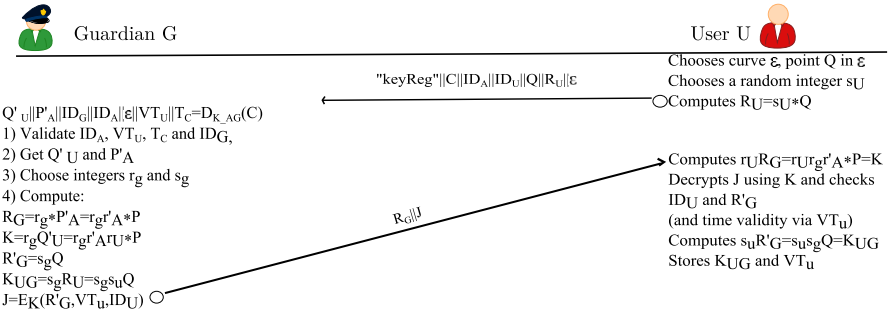

Fig. 3 Registering a Key with a Guardian

2.2 Visitor key generation

Let us assume that a user $\mathcal{U}$ wants to request access to a system service guarded by $\mathcal{G}$. He demands a visitor key from an administrator $\mathcal{A}$ or a super user $\mathcal{S}$ authorized to this end. Suppose that the request is sent to $\mathcal{A}$, who is sharing $K_{\mathcal{AG}}$ with $\mathcal{G}$. The steps, described in Fig. 2 shows a possible interaction scenario where Admin and Guardian computing entities cooperate to generate a Key upon a request submitted by a user. The computation associated with the user can be easily placed at the smartphone or at the server.

2.3 Visitor key registration

In this stage (see Fig. 3) user $\mathcal{U}$ will demand access to service whose guard is $\mathcal{G}$ using the information C that $\mathcal{U}$ received from $\mathcal{A}$ (or from $\mathcal{G}$ as in Fig. 2, there are a variety of possibilities). After that, both $\mathcal{G}$ and $\mathcal{U}$ will share a common key $K_{\mathcal{UG}}$.

Now $K_{\mathcal{UG}}$ will be used either to authenticate U when trying to access the system whose guard is $\mathcal{G}$ as we will show in Sect. 2.4, or to generate a new master or visitor keys, as shown in Sect. 2.6. Figure 3 also shows that the guard can be subject to heavy computational requirements during the Key registration stage.

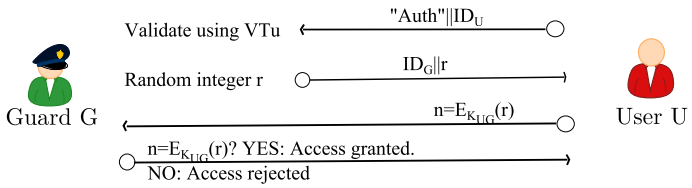

Fig. 4 Steps to access a service

2.4 Authentication

Authentication is the phase that grants access to any system or service supervised by $\mathcal{G}$. The guard $\mathcal{G}$ will authenticate any key owner $\mathcal{U}$ via their shared key using a typical challenge method. This is depicted in Fig. 4. This stage of the protocol is fast as only few messages are exchanged to decide whether the user is granted or denied access to the requested service.

2.5 Key revocation

Due to step #3 of the Key Generation stage of the protocol as described in Sect. 2.2, where ID_U is communicated in the clear, an attacker could forge any user's identity and get a valid visitor or even master key to be registered with a guard. The main consequence is that when the corresponding valid user tries to register a key with the same guard, he would receive a denial of service since there is someone else using his identity already authenticated with this guard. In this moment this should be communicated to the corresponding administrator and then revoke the registered key or in case of a master key, revoke also every key (visitor or master) that was created by this false user. The idea is then simple: the guard just has to represent every key in a directed graph where parent vertices correspond to key generators and child vertices to key requestors. Then if a key should be revoked as well as every key generated from the corresponding user to the first one, the guard will revoke this key and all the keys of users in its child vertexes. Suppose that identity of user $\mathcal{U}$ was forged to get $K_{\mathcal{UG}}$. Then the following steps are followed:

1. The guard $\mathcal{G}$ sends the message $E_{K_{\mathcal{AG}}}(ID_{\mathcal{U}}||\text{Revoke})$ to the administrator $\mathcal{A}$. In case $\mathcal{G}$ is authenticated to more than one administrator, the same message is sent to each of them by using the corresponding key shared with them.
2. The administrator sends the message $E_{K_{\mathcal{AG}'}}(ID_{\mathcal{U}}||\text{Revoke})$ to every guard $\mathcal{G}'$ that is authenticated to him in order to all of them revoke possible keys that were registered by $\mathcal{G}'$ corresponding to $\mathcal{U}$ and every possible visitor key that was created by him in case $\mathcal{U}$ holds a master key.

2.6 Master key generation

An important issue occurs in case an attacker is requesting a master key by forging some super user's identity as described in Sect. 2.5. To prevent this issue every administrator,

jointly to the corresponding guard, will be in charge to grant the Super User status. It is clear that not every user should hold a master key. Only those users that have shown a right behavior during a long period of time and whose identity was not previously detected as forged will be trust worthy. Creating a super user who is able to grant new visitor keys could be necessary in case an administrator observes that he is receiving many visitor key generation requests for a determined guard $\mathcal{G}$. Then taking into account the above remarks on possible trusted peers, the administrator $\mathcal{A}$ will send the following confidential message to $\mathcal{G}$

$$E_{K_{\mathcal{AG}}}(M\text{KeyGen}, ID_{\mathcal{A}}, ID_{\mathcal{U}}, N_1)$$

being $ID_{\mathcal{U}}$ the identity of a trusted peer $\mathcal{U}$ and N_1 is a unique security transmission number generated randomly by $\mathcal{A}$. $\mathcal{A}$ stores $ID_{\mathcal{U}}$ and N_1. In this case, since the user and the corresponding guard share a common key, namely $K_{\mathcal{UG}}$ the procedure would be as follows:

Getting super user status:

1. The Guard sends to the User the message

$$E_{K_{\mathcal{UG}}}(M\text{KeyGen}, ID_{\mathcal{U}}, ID_{\mathcal{A}}, E_{K_{\mathcal{AG}}}(ID_{\mathcal{A}}, N_1, N_2), N_2)$$

   where N_2 denotes a unique security transmission number randomly generated by $\mathcal{G}$, and $ID_{\mathcal{A}}$ is the identity of the administrator $\mathcal{A}$ to be contacted in order to get the corresponding master key. The Guard stores $ID_{\mathcal{U}}$ and N_2.
2. The User decrypts the message and stores N_2.

Master key generation

Then the Master Key Generation follows as introduced above, but in this case, user $\mathcal{U}$ sends the message to $\mathcal{A}$.

$$M\text{KeyGen}||ID_{\mathcal{G}}||ID_{\mathcal{U}}||E_{K_{\mathcal{AG}}}(ID_{\mathcal{A}}, N_1, N_2)||P||Q_{\mathcal{U}}||\mathcal{E}$$

Then $\mathcal{A}$ decrypts $E_{K_{\mathcal{AG}}}(ID_{\mathcal{A}}, N_1, N_2)$ and checks $ID_{\mathcal{A}}$ and N_1 and acts as in the Key Generation stage sending back to $\mathcal{U}$ the message C

$$C = E_{K_{\mathcal{AG}}}(Q'_{\mathcal{U}}, P'_{\mathcal{A}}, ID_{\mathcal{G}}, ID_{\mathcal{A}}, N_2, \mathcal{E}, VT_{\mathcal{U}}, T_C)$$

2.7 Master key registration

User $\mathcal{U}$ operates as in the Key Registration phase, but sends the message to $\mathcal{G}$.

$$M\text{KeyReg}||C||ID_{\mathcal{A}}||ID_{\mathcal{U}}||E_{K_{\mathcal{UG}}}(N_2, N_3)||P||Q_{\mathcal{U}}||\mathcal{E}$$

where N_3 denotes a unique integer number randomly generated by $\mathcal{U}$ that is stored.

When $\mathcal{G}$ receives the message, he decrypts C and gets all the information, including N_2. Then he decrypts $E_{K_{\mathcal{UG}}}(N_2, N_3)$ getting N_2 and N_3. The message is authenticated since N_2 is obtained from both previous values.

Now $\mathcal{G}$ operates similarly to step 4 in the Key Registration stage for the computation of $R_{\mathcal{G}}$ and J and sends back to $\mathcal{U}$ the message.

$$E_{K_{\mathcal{UG}}}(N_3, R_{\mathcal{G}}, J)$$

Then $\mathcal{U}$ decrypts it and the presence of N_3 will allow $\mathcal{U}$ to validate this message. After that $\mathcal{U}$ operates as in Fig. 3 after receiving $R_{\mathcal{G}}||J$ and stores the new generated master key and its corresponding valid time.

Remark We note that the Key Generation stage could be also carried out through a super user $\mathcal{S}$ who is authenticated to the corresponding guard $\mathcal{G}$. In that case the first message from $\mathcal{A}$ to $\mathcal{G}$ would be of the form

$$E_{K_{\mathcal{AG}}}(M\text{KeyGen}, ID_{\mathcal{S}}, ID_{\mathcal{U}}, N_1)$$

Then it would be necessary that the guard $\mathcal{G}$ sends to super user $\mathcal{S}$ the message $ID_{\mathcal{G}}||N_1$. This could be done also in a secure manner using the key $K_{\mathcal{SG}}$ shared by $\mathcal{S}$ and $\mathcal{G}$.

After that $\mathcal{G}$ sends to $\mathcal{U}$ the message

$$E_{K_{\mathcal{UG}}}(M\text{KeyGen}, ID_{\mathcal{U}}, ID_{\mathcal{S}}, E_{K_{\mathcal{SG}}}(ID_{\mathcal{S}}, N_1, N_2), N_2)$$

and $\mathcal{U}$ stores N_2 and starts the key generation stage by sending to $\mathcal{S}$ the message

$$M\text{KeyGen}||ID_{\mathcal{G}}||ID_{\mathcal{U}}||E_{K_{\mathcal{SG}}}(ID_{\mathcal{S}}, N_1, N_2)||Q||R_{\mathcal{U}}||\mathcal{E}$$

Then $\mathcal{S}$ checks N_1 obtained from $E_{K_{\mathcal{SG}}}(ID_{\mathcal{S}}, N_1, N_2)$ and goes on with the protocol exactly in the same way as previously.

We also note that the use of encryption and unique transmission security numbers allows every party in this stage to authenticate the source of the information and thus, repetition attacks, trying to forge any of the identities is not possible.

3 Security of the protocol

Through this section we will discuss security of the proposed protocol. To do so we will distinguish the attacks on its different participants. However, let us start by describing the problems that an attacker will face when trying to acquire a service access key guarded by $\mathcal{G}$ via brute force methods.

Let us suppose now that an attacker is observing all the information between $\mathcal{U}$ and $\mathcal{G}$ and tries to retrieve $K_{\mathcal{UG}}$. This secret key shared by the user and the guard can be derived from both the pair $(R_{\mathcal{U}}, R'_{\mathcal{G}})$ or J. Firstly we point out that getting $K_{\mathcal{UG}}$ from $(R_{\mathcal{U}}, R'_{\mathcal{G}})$ implies solving the Diffie–Hellman ECC problem, which is

computationally infeasible. Secondly, trying to get $K_{\mathcal{UG}}$ from J implies to decrypt J and thus involves getting K. However the only information related to K that is sent through the network is $R_{\mathcal{G}}$ and thus the attacker should know the secret number $r_{\mathcal{U}}$ held by $\mathcal{U}$. Thus the alternative is to get the secret K that is obtained by means of a Diffie–Hellman three parties key exchange (authority, user and guard), that is again computationally infeasible, even in case the authority or super user holding the master key and authorized to generate it, since although he can access the point P and its multiples

$$Q_{\mathcal{U}} = r_{\mathcal{U}} P, \quad Q'_{\mathcal{U}} = r'_{\mathcal{A}} r_{\mathcal{U}} P, \quad P'_{\mathcal{A}} = r'_{\mathcal{A}} P, \quad R_{\mathcal{G}} = r_{\mathcal{G}} r'_{\mathcal{A}} P,$$

$r_{\mathcal{U}}$ and $r_{\mathcal{G}}$ are kept secret by $\mathcal{U}$ and $\mathcal{G}$ respectively and he will have to solve any of the ECC problems given by the preceding multiples.

3.1 Forging key generation messages

In the Key Generation phase we are concerned on that the key generation is made through an authorized entity. Thus the message

$$C = E_{K_{\mathcal{AG}}}(Q'_{\mathcal{U}}, P'_{\mathcal{A}}, ID_G, ID_A, \mathcal{E}, VT_{\mathcal{U}}, T_C)$$

is created using the secret key $K_{\mathcal{AG}}$ shared by the guard and the corresponding authorized entity. Therefore, neither the user requesting the service nor anyone else can access or modify its content. The message C is only useful during Key Registration time and thus, this message can be used just to access the requested service and during the established time.

3.2 Forging user's identity

Such attack may occur at Key Registration where the attacker can attempt to get G to give it information to construct $K_{\mathcal{UG}}$, but cannot decrypt the response without r_U which is only stored locally at the real user $\mathcal{U}$ during the Key Generation phase, and never sent out in the clear. This leaves the second case where the attacker pretends to be $\mathcal{U}$ at the Key Generation phase (choose his own $r_{\mathcal{U}}$ and substitute $Q_{\mathcal{U}}$ in KeyGen$||Q_{\mathcal{U}}||P||\mathcal{E}||ID_{\mathcal{G}}||ID_{\mathcal{U}}$), which allows it to succeed until the real U comes along. The only solution would be changing the identity.

If somehow, the attacker is able to get a valid visitor key to access the service guarded by $\mathcal{G}$ corresponding to user $\mathcal{U}$, $K_{\mathcal{UG}}$, then the attacker will succeed with the precedent attack. However, this will be detected at the moment of $\mathcal{U}$ tries to get his new master key and thus the guard $\mathcal{G}$ will initiate the revoking key procedure.

In case the rogue entity pretends to be the victim of such an attack, this would be triggering a key revocation process falsely. There exists a solution to this fact that includes an authority and a process of authentication by using personal private keys at the moment of subscription. However, this may produce a bottle-neck at the authority and a major bandwidth usage. An alternative solution consists in changing the identity

of the user under attack, thus, a modification of the Key Generation stage that addresses this issue is one of our aims as future work.

3.3 Forging guard's identity

Pretending to be a Guard has to satisfy the following chain of secure dependencies in order to succeed at getting the shared key $K_{\mathcal{UG}}$.
(a) Attacker needs $\mathcal{U}$ to agree with him on the key $\mathcal{K}$ to proceed with the computation.
(b) This implies Attacker needs to know the contents of $\mathcal{C}$ sent by $\mathcal{U}$
(c) This implies Attacker needs to know $K_{\mathcal{AG}}$.
Under normal circumstances the attacker cannot trick $\mathcal{A}$ into sharing a key since the process is initiated by $\mathcal{A}$ during the Key initialization phase. However, a rogue entity cannot intercept the first message from $\mathcal{A}$ and send back a false response pretending to be $\mathcal{G}$, thus establishing a false $K_{\mathcal{AG}}$, from which the rogue Guard can subsequently use to trick users into false services as this is done during the Initialization phase through a secure channel.

Finally we assert that forging key revocation messages is computationally infeasible as encryption is used throughout this phase, with the keys exchanged through secure channels as shown in Sect. 2.1.

4 System implementation proposed

Elliptic curves are suitable for computation in embedded devices mainly because the length of the keys is not as big as it is in other cryptographic methods. There exist techniques that optimize ECC, like those in [13], according to the target platform.

As known, the performance of the ECC depends on the computation of scalar multiplication. In [13], authors using Java, proposed a method for speeding up the scalar multiplication of elliptic curves. This work shows an implementation in a smartphone, operated by an ARM processor. The user side of the system has been implemented using Java and ECC libraries built just with the functions that we needed for the protocol. The server side was implemented using Charm++ [5] which gave us the opportunity to implement a system that scales and that is able to easily implement latency hiding techniques. As depicted in Fig. 2, it is possible to shift an user compute object (operations that the smartphone needs to execute), either to the server side or to the client side. In case the user compute object is shifted to the server side, the number of messages through the Internet and the security are enhanced, instead, we placed the compute object in the smartphone to have an upper-bound limit of performance for our protocol. The main computational entities are:

- Server: creates a pool of guardian objects. The server has an initialization stage where it creates an elliptic curve, select as many points from the curve as the number of guardians specified and configure them to work concurrently attending requests from Internet. A server may have more than one administrator computing entity. Each administrator object has its own curve and set of guardians so the service can be authenticated by, for example, type of application or different sort of services.

Table 2 Time consumed by each computational entity, for each stage (in ms)

#users	Administrator	Guardian	User
Initialization stage			
n/a	1.251	117.52	n/a
Key generation$_{lat}$			
1	1.77	n/a	18.39
10	19.81	n/a	23.29
100	181.12	n/a	79.14
1,000	1,912.12	n/a	131.40
Key registration$_{lat(user)}$			
1	n/a	3.72	55.25
10	n/a	42.64	613.29
100	n/a	527	8,891.4
1,000	n/a	7,263.29	10,031.40
Authentication$_{lat}$			
1	n/a	0.19	0.71
10	n/a	0.47	1.09
100	n/a	1.27	1.94
1,000	n/a	4.26	3.10

- Guardian, it is in charge of authenticating the user upon access request and canceling the request if it is invalid.
- User, represents the activity at the user ($\mathcal{U}$) side.

The computation times due to the elliptic curve operations were reduced by using the recommendations from [13], the Internet high latencies were reduced by the fact that Charm++, a message driven language characterized by an efficient user level thread package able to fast context switch when a running thread stops for a communication to complete [i.e. the Guardian tries to authenticate an user and has to wait to the user to compute and send back the message $n = E_{K_{\mathcal{AG}}}(r)$] so the guardian might opt to attend a queued request or give way to a sleeping guardian otherwise. The tests presented in Table 2 were run on a second generation Intel Core i7-4770R Processor, the system was provided with 8GB of DDR3L-1333/1600 RAM.

Table 2 shows how the protocol performs in both scenarios (Server and Client side) for the realization of the protocol. Revealing the curve was not initially considered therefore points were sent in their uncompressed form. As curves were created according to NIST recommendations (secp256r1) points could also be sent in compressed form, enhancing latencies. User computing entity was placed in the smartphone for these tests. Although communications were not totally minimized it can be seen that the implementation scales almost linearly as the number of concurrent requests grow. Table 2 shows the time taken to serve an user request (as an average time computed by the smartphone) at any stage of the protocol, when there are number of users concurrent requests, that were emulated placing user objects at the server. There is a slight compromise between the number of simultaneous guardians created and the capacity to handle latencies properly. As future work, we intend to create guardians as the number of requests grow and remove them as the number of requests decreases. By

default, for tests in Table 2, the number of guardians was set to 100. In the Initialization stage it has been considered that the administrator creates 100 guardians, and that the curve and the point are the same for all, therefore as administrator and guardian are in the same address space the message is simply copied to the guardian. This stage is not affected by any sort of latency so it simply depends of how the guardians communicate (message passing) with the administrator. The presence of requests' bursts may affect performance of guardians (properly reentrant) as it is the case of the Key registration stage. In Table 2, during the Key generation stage, it is worth noting that both sides of the communication were affected by Internet latencies. In this case, the the guardian entities are just gateways for administrator entities which generate keys for users. Also, the performance of administrator objects are not severely affected by the number of user requests because adaptive overlap allows computation to be advanced during gaps in communication. As opposed to what happens in the generation stage, during the registration stage the guardian object is affected indirectly by the administrator object, as it is the one that provides the guardian with key information. The authentication stage is not computationally intensive but both, guardian and user are submitted to the Internet latencies as both need to exchange a minimum number of four messages. As the computation is few compared with the latencies, the service (execution) time is tightly related to the Internet exclusively.

5 Conclusions and future work

The work presented proposes an algorithm for a new decentralized security protocol with a focus on support for dynamic services on mobile devices with limited compute and storage capabilities. Details of the protocol were described, with attention to soundness, and security in the face of possible attacks, with key revocation as the primary means of response to a detected attack. The protocol was demonstrated to be secure and robust. This work also presents a implementation with performance results for overhead in the face of scaling. The protocol devised adopts algorithms especially suitable for resource-constrained platforms, comprising ECC-based protocols for reducing key sizes while ensuring high security levels. Our benchmark results are based on an implementation of the protocol with high performance computing techniques together with many optimizations recommended in the literature, both implementations on Java (for the client side) and on a high performance computing language (on the server side) show that the method proposed is powerful and fast. The server was able to attend to the request of a variable number of users in an acceptable time. As the number of users scaled up to 1,000, the performance of the protocol was also able to keep up.

Avoiding attacks as the one described in Sect. 3 (forging user's identity) where the rogue entity pretends to be the victim of such an attack, would require including an authority and a process of authentication by using personal private keys at the moment of subscription. This seems to be ineffective as may produce a bottle-neck at the authority. Designing an alternative solution to change the identity of the user under attack, that addresses this issue is one of our aims as future work.

Acknowledgments Supported by the Spanish Ministry of Science and Innovation Grants TEC2009-13763-C02-02, TIN2008-01117 and Junta de Andalucía FQM0211, P11-TIC7176.

References

1. Cao X, Kou W, Du X (2010) A pairing-free identity-based authenticated key agreement protocol with minimal message exchanges. Inf Sci 180(15):2895–2903
2. Chen L, Kudla C (2003) Identity based authenticated key agreement protocols from pairings. In: Proceedings 16th IEEE, Computer security foundations workshop, pp 219–233
3. Cheng Z, Nistazakis M, Comley R, Vasiu L (2004) On the indistinguishability-based security model of key agreement protocols-simple cases. In: Proceedings of ACNS, vol 4. Citeseer
4. ElGamal T (1985) A public key cryptosystem and a signature scheme based on discrete logarithms. Inf Theory IEEE Trans 31(4):469–472
5. Kale L, Arya A, Bhatele A, Gupta A, Jain N, Jetley P et al (2011) Charm++ for productivity and performance: a submission to the 2011 HPC Class II Challenge. 11–49
6. Koblitz N (1987) Elliptic curve cryptosystems. Math Comput 48(177):203–209
7. Miller V (1985) Use of elliptic curves in cryptography. In: CRYPTO: Proceedings of crypto, pp 417–426
8. NSA suite b cryptography. http://www.nsa.gov/ia/programs/suiteb_cryptography/
9. Rivest R, Sharmir A, Adleman L (1978) A method for obtaining digital signatures and public-key cryptography. Commun ACM 21(2):120–128
10. Shim K (2003) Efficient id-based authenticated key agreement protocol based on weil pairing. Electron Lett 39(8):653–654
11. Smart N (2002) Identity-based authenticated key agreement protocol based on weil pairing. Electron Lett 38(13):630–632
12. Sun H, Hsieh B (2003) Security analysis of Shim's authenticated key agreement protocols from pairings. In: Cryptography ePrint Archive, Report 113
13. Tsaur W-J, Chou C-H (2005) Efficient algorithms for speeding up the computations of elliptic curve cryptosystems. Appl Math Comput 168(2):1045–1064
14. Xuefei C, Weidong K, Yong Y, Rong S (2008) Identity-based authenticated key agreement protocols without bilinear pairings. IEICE Trans Fundam Electron Commun Comput Sci 91(12):3833–3836