International Journal of Information Security (2019) 18:181–197
https://doi.org/10.1007/s10207-018-0406-4

REGULAR CONTRIBUTION

Multi-device anonymous authentication

Kamil Kluczniak[1] · Jianfeng Wang[2] · Xiaofeng Chen[2] · Mirosław Kutyłowski[1]

Published online: 20 April 2018

Abstract
Recently, a few pragmatic and privacy protecting systems for authentication in multiple systems have been designed. The most prominent examples include Pseudonymous Signatures for German personal identity cards and Anonymous Attestation. The main properties are that a user can authenticate himself with a single private key (stored on a smart card), but nevertheless the user's IDs in different systems are unlinkable. We develop a solution which enables a user to achieve the above-mentioned goals while using more than one personal device, each holding a single secret key, but different for each device. Our solution is privacy preserving: it will remain hidden for the service system which device is used. Nevertheless, if a device gets stolen, lost or compromised, the user can revoke it (leaving his other devices intact). In particular, in this way we create a strong authentication framework for cloud users, where the cloud does not learn indirectly personal data. Our solution is based on a novel cryptographic primitive, called Pseudonymous Public Key Group Signature.

Keywords Signature schemes · Privacy · Pseudonyms · Group signatures · Authentication

1 Introduction

So far most authentication systems for web services or cloud servers were designed having in mind a single user or a group of users and a single service provider. Today such systems become increasingly popular, and the number of systems used per user is rapidly growing. If authentication is taken seriously (not based just on a login and a password), then for each service we get an independent authentication environment that requires generation and distribution of the secret keys for the users. Such a framework has serious disadvantages: the necessity of managing secret/public keys among certain parties, constant updates of user secret keys and maintaining large and costly PKI infrastructures. The task of switching between password-based authentication and strong cryptographic authentication mechanisms is made even more difficult by the growing amount of mobile devices used by a single user. As studied in [10], cryptographic authentication schemes usually fail to provide reasonable usability to the user, what is especially the case when we consider users carrying a dynamically changing set of devices. Thus, it seems that replacing passwords by public key cryptography is a hard task basically because of complicated key management.

In this paper, we develop a framework which aims to provide a cryptographically sound authentication scheme to a dynamically growing set of services, which preserves privacy for groups of devices and their users and does not require expensive, time and resource-consuming infrastructures as well as key management procedures.

Application scenario

In order to be more specific, we consider an application scenario of Multiple Mobile Devices and Authentication for Web Services, called below *domains*: Informally, we put the following requirements for our system:

– A user creates a single secret key on a secure token (e.g., smart card).

✉ Kamil Kluczniak
kamil.kluczniak@pwr.edu.pl

Jianfeng Wang
jfwang@xidian.edu.cn

Xiaofeng Chen
xfchen@xidian.edu.cn

Mirosław Kutyłowski
miroslaw.kutylowski@pwr.edu.pl

1 Department of Computer Science, Wrocław University of Science and Technology, Wrocław, Poland

2 State Key Laboratory of Integrated Service Networks (ISN), Xidian University, Xi'an, China

- The user registers to a given domain only once using his secure token and might derive a unique public key, called pseudonym, in each domain he registers in.
- Using the secure token, the user may personalize his mobile devices.
- Having a personalized device, the user may authenticate himself to each domain to which he has registered a pseudonym. Moreover,
  - in case the user registers to a new domain, no device needs to be updated in order to authenticate to the new domain, and
  - adding a new devices does not require updating any public key information in any service nor any other device.
- Finally, we also require that a user must be able to revoke each of his devices in case of theft, key leakage, etc.

For usability reasons, we assume that a user registers once in a domain by providing his public key for this domain. Moreover, no party except for the user and the service domain should be involved.

After registration, without any updates or interaction with any party, the participant should be able to delegate the right to run the authentication protocol on behalf of the user and sign digitally challenges in order to authenticate the user.

Privacy and unlinkability issues

One of the major threats in a multi-system environment is that the authentication means from one domain can be misused for getting unlawful access into user's accounts in another domain. For password-based systems, this is a severe threat as the users tend to use the same password in multiple places. Many recent examples are known where compromise of one system resulted in compromising users' accounts in another system.

Apart from unlawful access, it might be necessary to protect the information that a given physical person is a user in a domain. Therefore, after the phase of registration the user's identity should be anonymized. Moreover, the pseudonyms in different domains should be unlinkable, even when the data from authentication sessions are at hand. In this case, a potential data leakage is not threatening the principles of personal data protection.

Previous attempts to replace passwords

Over time there were several proposals to replace password-based authentication mechanisms to Web Services. An early idea was to run a trusted identity server which confirms user identity. The concept is called federated single sign-on and is utilized in systems like Kerberos [24] which is based on the Needham–Schroeder protocol [27]. Another prominent example for federated authentication is the OpenID protocol [31], where any server might act as an identity provider. In practice however, only a few web services are willing to accept user authentication relying on third parties [33]. Thus, one of our requirements is the resignation from third parties between a user and the service to which the user authenticates.

Another type of solutions is to use hard tokens like RSA SecurID or solutions based on smartphones [29]. Although these systems offer a strong authentication mechanism, they suffer from a few important drawbacks. A user needs to carry these tokens wherever he goes and needs to type in one-time passwords generated by the token each time he wants to login to a web service. Moreover, the tokens need to be synchronized with a web service and only a few tokens are designed to handle multiple services.

Comprehensive surveys of the above-mentioned techniques and including biometric identification can be found in [10,28,32].

We may observe that our framework shares some similarities with hardware tokens. In particular, we assume a user to have a single token for device personalization and domain registration. In contrast to existing hardware tokens, in our framework we need to use the token only in two situations: in order to register in a service, to personalize a device and to revoke a device. Besides these situations, a user does not need to carry or use the token whenever he wants to login.

Group signatures

We noticed that a primitive called group signatures may fit our application scenario, but, as we explain later, group signatures do not cover all necessary functionalities. Thus, we briefly recall the notion of group signatures below, and then we discuss some drawbacks of this primitive in the context of our application.

Group signatures as defined in [3] or [5] are signature schemes in which a *group manager* admits users to the group. Each group member may sign data anonymously on behalf of the group. Only an entity called an *opener* may "*open*" a signature and derive the signer's real identity. Informally, a group signature scheme has to fulfill the following properties:

anonymity: it is infeasible to establish the signer of a message. To be more specific, having two signatures one cannot even say whether they originate from one signer or from two different signers.

unframeability: it is infeasible, even for a coalition of malicious group members, to forge a signature which would open to the identity of a group member not belonging to the coalition.

traceability: it is infeasible to produce a signature which would open to an identity not added to the group by the group manager.

Group signatures is a well-studied cryptographic primitive. There are many variants of them, with security proofs based either on the random oracle model (e.g., [8]), or on the standard model (e.g., [11]). Many variants of group signatures have been developed, like Verifier Local Group Signatures [9], Traceable Signatures [22], Hierarchical [34], Attribute [1] and Identity-Based Group Signatures [19].

Ad hoc solution based on group signatures

At a first look, group signature schemes address our practical problem pretty well. The user plays the role of the group manager for group signatures, while his devices play the role of group members (admitted by the manager). Note that this construction gives some functionalities for free:

- the user can delegate his rights to authenticate on behalf of him to any number of his devices—indeed, the number of group members is typically unlimited,
- the devices are indistinguishable from the point of view of the verifier—this is the basic feature of group signatures,
- in case of a misbehavior, the user may open a signature and find which device has created it.

Unfortunately, there are also some drawbacks that have to be addressed. The main problem is that we have to create separate and unlinkable authentication means for different domains. Creating a new independent group for each domain separately would solve this problem; however, this would require installing separate keys for each domain on each single device. For practical reasons, this is not really acceptable.

Unfortunately, existing group signature schemes have been designed having in mind single groups or a hierarchy of groups with central authorities. In particular, existing schemes assume that a group of such a hierarchy is identified by a public key determined by the scheme setup. This makes such schemes unsuitable for our application. Our aim is therefore to design a group signature scheme in which group public keys may be derived spontaneously from a domain specific bit string (e.g., www.some-service.dom), a secret key of the group manager, and with no involvement of PKI and/or trusted authorities.

Moreover, group public keys or, as we will call it, *domain pseudonyms* must be unlinkable, what means that having two or more domain pseudonyms from distinct domains it is infeasible to tell whether the pseudonyms correspond to a group manager.

Such an anonymity notion is known from Domain Pseudonymous Signature schemes (see, e.g., [13]), (see, e.g., Direct Anonymous Attestation [12]) and Anonymous Credential Systems (see, e.g., [14]). What is important, creating new public keys by a group manager does not require from group members to update their secret keys or any other information and they might automatically sign data corresponding to the new public key.

Contribution and paper overview

Our main technical contribution is a new concept of group signatures, where group public keys are domain pseudonyms which might be derived spontaneously. The particular setting is tailored for the above-mentioned application of delegating authentication chores to multiple devices of a user.

In Sect. 3, we give a formal definition for our new primitive. This is followed in Sect. 4 by a relatively efficient construction based on pairings. In Sect. 4.2, we evaluate the efficiency of our construction. We give also some additional remarks, and we show how to apply our scheme to solve our practical problem. In Sect. 5, we formulate theorems corresponding to the security of our scheme and give formal proofs of these theorems. The proofs of these theorems are based on the random oracle model assumption, which is dictated mainly by efficiency and practical needs of the construction. Finally, in Sect. 6 we describe and analyze a Σ-protocol on which the construction of our scheme is based.

Previous version This is the full version of the paper that has been presented in NSS 2016 [23]. The main differences from the conference version are as follows: Firstly, we present additional related work on authentication schemes based on federated identity management and secure hard tokens. Furthermore, we give a comprehensive efficiency evaluation and comparison with VLR group signature schemes. Secondly, we add the formal security analysis of our scheme in Sect. 5. Finally, we included the description of an honest verifier zero-knowledge proof of knowledge protocol which forms the basis of our signature scheme. Moreover, we proved the zero-knowledge and proof of knowledge properties of the proposed protocol.

2 Preliminaries

In this section, we recall some preliminaries necessary for understanding the rest of the paper. In particular, we recall the definition of groups with bilinear maps, and assumptions and techniques necessary for the security analysis of our scheme.

2.1 Bilinear groups

Let $\mathbb{G}_1$, $\mathbb{G}_2$ be cyclic groups of a prime order p, generated by $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$. In our scheme, we make use of bilinear maps $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, which are:

– *bilinear*: for $a, b \in \mathbb{Z}_p$, we have $e(g_1^a, g_2^b) = e(g_1, g_2)^{a \cdot b}$,
– *non-degenerate*: the element $e(g_1, g_2) \in \mathbb{G}_T$ is a generator of $\mathbb{G}_T$.

Additionally, we require that e and all group operations are efficiently computable.

Throughout the paper, we will use Type-3 pairing according to the classification from [17]. We call a pairing of Type-3, if $\mathbb{G}_1 \neq \mathbb{G}_2$ and no efficiently computable homomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$ is known.

2.2 Forking lemma

In this paper, we are mainly interested in signature schemes, hence we briefly recall the forking lemma from [4]. The forking lemma, as stated in [4], tells us that having a forger of a signature scheme who outputs a valid forge with probability ϵ after q_{H} hash queries (counted together with signature queries), we may run the experiment again, change the output of a random hash query, and obtain a forge for the same message with probability

$$frk \geq \epsilon^2 / q_{\mathsf{H}} - 1/2^{\ell}$$

where ℓ is the bit length of the output of the hash function.

2.3 Security assumptions

Definition 1 (*Discrete Logarithm Problem* (*DLP*)) Let $\mathbb{G}$ be a cyclic group of prime order p with a generator $g \in \mathbb{G}$. An algorithm A has advantage ϵ in solving the DLP if

$$\Pr\left[\mathsf{A}\left(g, g^{\alpha}\right) \to \alpha\right] \geq \epsilon,$$

where the probability is taken over the random choice of the generator $g \in \mathbb{G}$, the random choice of $\alpha \in \mathbb{Z}_p$, and the random bits of A.

We say that the (t, ϵ)-DL assumption holds in $\mathbb{G}$ if no time t algorithm has advantage ϵ in solving DLP in $\mathbb{G}$.

Definition 2 (*Decisional Diffie–Hellman Problem* (*DDH*)) Let $\mathbb{G}$ be a cyclic group of order p with a generator $g \in \mathbb{G}$. An algorithm A has advantage ϵ in solving the DDH problem if

$$\left|\Pr\left[\mathsf{A}\left(g^{\alpha}, g^{\beta}, g^{\alpha \cdot \beta}\right) \to 1\right] - \Pr\left[\mathsf{A}\left(g^{\alpha}, g^{\beta}, g^{\gamma}\right) \to 1\right]\right| \geq \epsilon,$$

where the probability is taken over the random choice of $g \in \mathbb{G}$, the random choice of $(\alpha, \beta, \gamma) \in \mathbb{Z}_p^3$, and the random bits of A.

We say that the (t, ϵ)-DDH assumption holds in $\mathbb{G}$, if no time t algorithm has advantage at least ϵ in solving the DDH problem in $\mathbb{G}$.

Definition 3 (*Symmetric eXternal Diffie–Hellman assumption* (*SXDH*)) Let $\mathbb{G}_1, \mathbb{G}_2$ be cyclic groups of a prime order and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a bilinear map. The SXDH assumption says that the DDH assumption holds in both $\mathbb{G}_1$ and $\mathbb{G}_2$.

Definition 4 (*Bilinear Decisional Diffie–Hellman Assumption*) Let $\mathbb{G}$ be a cyclic group of a prime order and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a bilinear map. An algorithm A as advantage ϵ in solving the BDDH problem if

$$\left|\Pr\left[\mathsf{A}\left(g^{\alpha}, g^{\beta}, g^{\gamma}, e(g, g)^{\alpha \cdot \beta \cdot \gamma}\right) \to 1\right] - \Pr\left[\mathsf{A}\left(g^{\alpha}, g^{\beta}, g^{\gamma}, e(g, g)^{\delta}\right) \to 1\right]\right| \geq \epsilon,$$

where the probability is taken over the random choice of $g \in \mathbb{G}$, the random choice of $(\alpha, \beta, \gamma, \delta) \in \mathbb{Z}_p^3$, and the random bits of A.

We say that the (t, ϵ)-BDDH assumption holds in $\mathbb{G}$, if no time t algorithm has advantage at least ϵ in solving the BDDH problem in $\mathbb{G}$.

Definition 5 (*Collusion attack algorithm with q traitors* (*q-CAA*)) Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be groups of a prime order p and generated by $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$. Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a bilinear map which maps into a target group $\mathbb{G}_T$.

An algorithm A has advantage ϵ in solving the q-CAA problem, if

$$\Pr\left[\begin{array}{c} \mathsf{A}\left(g_1, g_1^z, \left(m_1, g_1^{\frac{1}{z+m_1}}\right), \ldots, \left(m_q, g_1^{\frac{1}{z+m_q}}\right), \\ g_2, g_2^z\right) \to \left(m, g_1^{\frac{1}{z+m}}\right) \wedge m \notin \{m_1, \ldots, m_q\} \end{array}\right] \geq \epsilon,$$

where the probability is taken over the random choice of $(g_1, g_2) \in \mathbb{G}_1 \times \mathbb{G}_2$, the random choice of $z \in \mathbb{Z}_p$, the random choice of $(m_1, \ldots, m_q) \in \mathbb{Z}_p^q$, and the random bits of A.

We say that (q, t, ϵ)-CAA assumption holds in $(\mathbb{G}_1, \mathbb{G}_2)$, if no time t algorithm has advantage at least ϵ in solving the q-CAA problem in $(\mathbb{G}_1, \mathbb{G}_2)$.

3 Formal model of Pseudonymous Public Key Group Signature

A Pseudonymous Public Key Group Signature scheme consists of the following procedures:

$\mathsf{Setup}(1^{\lambda})$: On input a security parameter λ, it outputs global parameters $param$.
$\mathsf{CreateUser}(param)$: On input the global parameters $param$, it creates and outputs the user's master secret key mSK.
$\mathsf{ComputePseudonym}(param, mSK, \mathtt{dom})$: On input the global parameters $param$, the master secret key mSK and

a domain name dom, it returns a pseudonym nym within domain dom for the user holding mSK.
AddDevice($param, mSK, i$): On input the global parameters $param$, the master secret key mSK and a device identifier i, this procedure returns a device secret key uSK_i.
CreateRevocationToken($param, mSK$, dom, i, j): On input the global parameters $param$, user index i and his secret key mSK, the domain name dom and a device identifier j, this procedure computes and outputs a device revocation token $uRT_{i,j,\mathtt{dom}}$ within the domain dom.
Sign($param, uSK$, dom, m): On input the global parameters $param$, a device secret key uSK, a domain name dom and a message m, it returns a signature σ on the message m.
Verify($param, nym$, dom, σ, m, uRT): On input the global parameters $param$, a pseudonym nym with regard to a domain name dom, a signature σ on a message m, and a revocation token uRT, this algorithm returns 1 (accept), or 0 (reject).

Below we discuss the required properties of Pseudonymous Public Key Group Signatures.

Correctness A Pseudonymous Public Key Group Signature is correct, if for every $\lambda \in \mathbb{N}$, $param \leftarrow \mathsf{Setup}(1^\lambda)$, domain name $\mathtt{dom} \in \{0,1\}^*$, and message $m \in \{0,1\}^*$, if

$$
\begin{aligned}
&mSK_i \leftarrow \mathsf{CreateUser}(param)\\
&uSK_{i,j} \leftarrow \mathsf{AddDevice}(param, mSK_i, j)\\
&nym \leftarrow \mathsf{ComputePseudonym}(param, mSK_i, \mathtt{dom})\\
&uRT_{i,j,\mathtt{dom}^*} \leftarrow \mathsf{CreateRevocationToken}(param,\\
&\qquad\qquad mSK_i, \mathtt{dom}^*, j)\\
&\sigma \leftarrow \mathsf{Sign}(param, uSK_{i,j}, \mathtt{dom}, m)
\end{aligned}
$$

then

$$
\begin{aligned}
&\mathsf{Verify}(param, nym, \mathtt{dom}, \sigma, m, R) = 1\\
&\quad for \;\; R \neq uRT_{i,j,\mathtt{dom}^*}\\
&\mathsf{Verify}(param, nym, \mathtt{dom}, \sigma, m, uRT_{i,j,\mathtt{dom}^*}) = 0\,.
\end{aligned}
$$

In order to define the remaining properties, we use the following notation: $\mathcal{U}_{SET}$ stands for the list of users and their secret keys, $\mathcal{D}_{SET}$ contains triples (i, j, uSK), where i denotes a user index, j is a device index and uSK is its secret key, $\mathcal{CD}$ is a list pointing to corrupted devices and $\mathcal{S}$ is a list of signature query records. Then we define the following oracles used by the adversary during the security games:

$\mathcal{O}_{\mathsf{CreateUser}}$: On input i, if there exists an entry $(i, .)$ in $\mathcal{U}_{SET}$, the oracle aborts. Otherwise the oracle runs $mSK_i \leftarrow \mathsf{CreateUser}(param)$ and adds the pair (i, mSK_i) to $\mathcal{U}_{SET}$.

$\mathcal{O}_{\mathsf{GetNym}}$: On input dom and i, the oracle finds the secret key mSK_i in $\mathcal{U}_{SET}$ corresponding to i. If no such entry exists, then the oracle aborts. Otherwise the oracle computes $nym_{i,\mathtt{dom}} \leftarrow \mathsf{ComputePseudonym}(param, mSK_i, \mathtt{dom})$ and returns $nym_{i,\mathtt{dom}}$.

$\mathcal{O}_{\mathsf{AddDevice}}$: On input a user index i and a device identifier j, the oracle finds an entry $(i, mSK_i) \in \mathcal{U}_{SET}$ and checks that $(i, j, \cdot) \notin \mathcal{D}_{SET}$. If $(i, j, \cdot) \notin \mathcal{D}_{SET}$, then the oracle aborts. Then the oracle computes $uSK_{i,j} \leftarrow \mathsf{AddDevice}(param, mSK_i, j)$ and $uSK_{i,j} \leftarrow \mathsf{AddUser}(param, mSK, j)$ and adds the tuple $(i, j, uSK_{i,j})$ to $\mathcal{D}_{SET}$.

$\mathcal{O}_{\mathsf{AddCorruptedDevice}}$: On input a user identifier i and a device identifier j, the oracle finds $(i, mSK_i) \in \mathcal{U}_{SET}$ and checks that $(i, j, \cdot) \notin \mathcal{D}_{SET}$ (if this is not the case, then the oracle aborts). Otherwise the oracle runs $uSK_{i,j} \leftarrow \mathsf{AddDevice}(param, mSK, j)$, adds the tuple $(i, j, uSK_{i,j})$ to $\mathcal{D}_{SET}$ and $\mathcal{CD}$, and outputs $uSK_{i,j}$.

$\mathcal{O}_{\mathsf{GetRT}}$: On input a user identifier i and his master key mSK_i, a device identifier j and a domain name dom, the oracle checks that $(i, j, \cdot) \in \mathcal{D}_{SET}$, (if this is not the case, then the oracle aborts). Then the oracle computes $uRT_{i,j,\mathtt{dom}} \leftarrow \mathsf{CreateRevocationToken}(param, mSK_i, \mathtt{dom}, j)$ and returns $uRT_{i,j,\mathtt{dom}}$.

$\mathcal{O}_{\mathsf{Sign}}$: On input a user identifier i, a device identifier j, a domain name dom and a message m, the oracle finds the corresponding secret key $uSK_{i,j}$ in $\mathcal{D}_{SET}$, (if such an entry does not exist, then the oracle aborts). Otherwise, the oracle runs $\sigma \leftarrow \mathsf{Sign}(param, uSK_{i,j}, \mathtt{dom}, m)$, adds $(\sigma, m, \mathtt{dom}, j, i)$ to $\mathcal{S}$ and returns σ.

$\mathcal{O}_{\mathsf{CorruptDevice}}$: On input a user identifier i and a device identifier j, the oracle finds the secret key $uSK_{i,j}$ in $\mathcal{D}_{SET}$ corresponding to i and j. (If such an entry does not exist, then the oracle aborts.) Then the oracle returns $uSK_{i,j}$ and adds (i, j) to $\mathcal{CD}$.

Unforgeability This property says that no coalition of malicious devices of a user can forge a signature on behalf of a device not belonging to the coalition. We define the unforgeability property by the following experiment:

Experiment $\mathsf{UNF}^S_{\mathsf{A}}(\lambda)$**:**

- $(param) \leftarrow \mathsf{Setup}(1^\lambda)$.
- $\mathcal{O} \leftarrow \{\mathcal{O}_{\mathsf{CreateUser}}, \mathcal{O}_{\mathsf{GetNym}}, \mathcal{O}_{\mathsf{AddDevice}}, \mathcal{O}_{\mathsf{GetRT}}, \mathcal{O}_{\mathsf{Sign}}, \mathcal{O}_{\mathsf{CorruptUser}}\}$.
- $(\sigma^*, m^*, \mathtt{dom}^*, nym^*) \leftarrow \mathsf{A}^{\mathcal{O}}(param)$.
- If
  - $\mathsf{Verify}(param, nym^*, \mathtt{dom}^*, \sigma^*, m^*, \perp) = 1$ and
  - There exists $(i, mSK_i) \in \mathcal{U}_{SET}, (i, j, \cdot) \in \mathcal{D}_{SET}$ such that
    - $nym^* = \mathsf{ComputePseudonym}(param, mSK_i, \mathtt{dom}^*)$,
    - $uRT_{i,j,\mathtt{dom}^*} \leftarrow \mathsf{CreateRevocationToken}(param, mSK_i, \mathtt{dom}^*, j)$,
    - $\mathsf{Verify}(param, nym^*, \mathtt{dom}^*, \sigma^*, m^*, uRT_{i,j,\mathtt{dom}^*}) = 0$,
    - $(i, j) \notin \mathcal{CD}$ and $(\sigma^*, m^*, \mathtt{dom}^*, j, i) \notin \mathcal{S}$,

  then the challenger returns 1.
- Otherwise the challenger returns 0.

Definition 6 A Pseudonymous Public Key Group Signature S is (t, ϵ)-unforgeable if $\Pr[UNF^S_{\mathsf{A}}(\lambda) = 1] \leq \epsilon$ for any adversary A running in time t.

Seclusiveness Seclusiveness means that it is infeasible to produce a signature on behalf of the user and that does not correspond to any device of the user. In other words, it is infeasible to create a signature that corresponds to none of the revocation tokens. Seclusiveness is formally defined by the following experiment.

Experiment $\mathsf{SEC}^S_{\mathsf{A}}(\lambda)$**:**

- $(param) \leftarrow \mathsf{Setup}(1^\lambda)$.
- $\mathcal{O} \leftarrow \{\mathcal{O}_{\mathsf{CreateUser}}, \mathcal{O}_{\mathsf{GetNym}}, \mathcal{O}_{\mathsf{AddCorruptedDevice}}, \mathcal{O}_{\mathsf{GetRT}}\}$.
- $(\sigma^*, m^*, \mathtt{dom}^*, nym^*) \leftarrow \mathsf{A}^{\mathcal{O}}(param)$.
- If
  - $\mathsf{Verify}(param, nym^*, \mathtt{dom}^*, \sigma^*, m^*, \perp) = 1$ and
  - there exists $(i, mSK_i) \in \mathcal{U}_{SET}$ such that
    - $nym^* = \mathsf{ComputePseudonym}(param, mSK_i, \mathtt{dom}^*)$,
    - for all j such that $(i, j, \cdot) \in \mathcal{D}_{SET}$: $uRT_{i,j,\mathtt{dom}^*} \leftarrow \mathsf{CreateRevocationToken}(param, mSK_i, \mathtt{dom}^*, j)$ and $\mathsf{Verify}(param, nym^*, \mathtt{dom}^*, \sigma^*, m^*, uRT_{i,j,\mathtt{dom}^*}) = 1$

  the challenger returns 1.
- Otherwise the challenger returns 0.

Definition 7 We say that a Pseudonymous Public Key Group Signature S is (t, ϵ)-seclusive, if $\Pr[\mathsf{SEC}^S_{\mathsf{A}}(\lambda) = 1] \leq \epsilon$ for any adversary A running in time t.

Anonymity We require that it is infeasible to correlate two signatures of the same device (unless its revocation token is used). For the anonymity experiment, we define an additional oracle:

$\mathcal{O}_{\mathsf{Challenge}}$: This oracle takes as input a bit b, a user index i^*, a domain name $\mathtt{dom}^*$, two device indexes j_0^*, j_1^* and a message m^*. If

- $(i^*, \cdot) \notin \mathcal{U}_{SET}$ or $j_0^* = j_1^*$, or
- $(i^*, j_0^*, \cdot) \notin \mathcal{D}_{SET}$ or $(i^*, j_1^*, \cdot) \notin \mathcal{D}_{SET}$, or
- $(i^*, j_0^*) \in \mathcal{CD}$ or $(i^*, j_1^*) \in \mathcal{CD}$, or
- the $\mathcal{O}_{\mathsf{GetRT}}$ oracle was called on input $(i^*, j_0^*, \mathtt{dom}^*)$ or $(i^*, j_1^*, \mathtt{dom}^*)$,

then the oracle returns $\perp$ and aborts. Otherwise, the oracle computes $\sigma \leftarrow \mathsf{Sign}(param, uSK_{i^*,j_b^*}, \mathtt{dom}^*, m^*)$ and returns σ.

After calling the $\mathcal{O}_{\mathsf{Challenge}}$ oracle, the adversary cannot call the $\mathcal{O}_{\mathsf{GetRT}}$ on input $(i^*, j_0^*, \mathtt{dom}^*)$ or $(i^*, j_1^*, \mathtt{dom}^*)$, and the $\mathcal{O}_{\mathsf{CorruptUser}}$ on input (i^*, j_0^*) or (i^*, j_1^*).

Experiment $\mathsf{Anon}^S_{\mathcal{A}}$**:**

- $(param) \leftarrow \mathsf{Setup}(1^\lambda)$.
- choose $b \in \{0, 1\}$ at random,
- $\mathcal{O} \leftarrow \{\mathcal{O}_{\mathsf{CreateUser}}, \mathcal{O}_{\mathsf{GetNym}}, \mathcal{O}_{\mathsf{AddDevice}}, \mathcal{O}_{\mathsf{GetRT}}, \mathcal{O}_{\mathsf{Sign}}, \mathcal{O}_{\mathsf{CorruptUser}}, \mathcal{O}_{\mathsf{Challenge}}(b, \cdot, \cdot, \cdot, \cdot)\}$.
- $\hat{b} \leftarrow \mathsf{A}^{\mathcal{O}}(param)$.
- If $\hat{b} = b$, then output 1, otherwise output 0.

Definition 8 A Pseudonymous Public Key Group Signature S is (t, ϵ)-anonymous if $|\Pr[\mathsf{Anon}^S_{\mathsf{A}}(\lambda) = 1] - \frac{1}{2}| \leq \epsilon$ for any adversary A running in time t.

Domain unlinkability Informally, domain unlinkability means that it is infeasible to correlate two domain pseudonyms with a single user. We will give a simulation based definition for the domain unlinkability property.

First we need to define the following data structures: $\mathcal{D}$ denotes a set of domain names, $\mathcal{U}^I_{SET}$ is the set of user indexes, $\mathcal{K}$ denotes an associative map which maps a pair $(\mathtt{dom}, i) \in \{0, 1\}^* \times \mathbb{N}$ into a master secret key from the secret key space $\mathcal{USK}$. Then we define an associative map $\mathcal{UK}$ which maps a tuple $(\mathtt{dom}, i, j) \in \{0, 1\}^* \times \mathbb{N}^2$ into a device secret key.

Then we define the following oracles which implement the ideal functionality, where the keys of the user for different domains are independent (note that for Pseudonymous Public Key Group Signature they are the same):

$\mathcal{O}^{Ideal}_{\mathsf{CreateUser}}$: The query requests to create a secret key for the ith user. If $i \notin \{1, \ldots, n\}$ or $i \in \mathcal{U}^I_{SET}$, then the oracle aborts. Otherwise, the oracle adds i to $\mathcal{U}^I_{SET}$ and for each

$\mathtt{dom} \in \mathcal{D}$, the oracle chooses a secret key $mSK_{i,\mathtt{dom}}$ at random from $\mathcal{USK}$ and sets $\mathcal{K}[(i, \mathtt{dom})] \leftarrow mSK_{i,\mathtt{dom}}$.

$\mathcal{O}^{Ideal}_{\mathsf{AddDevice}}$: The query requests to create the jth device for user i. For each $\mathtt{dom} \in \mathcal{D}$ the oracle obtains $mSK_{i,\mathtt{dom}} \leftarrow \mathcal{K}[(i, \mathtt{dom})]$ and runs $uSK_{\mathtt{dom},i,j} \leftarrow \mathsf{AddDevice}(param, mSK_{\mathtt{dom},i}, j)$, and sets $\mathcal{UK}[(\mathtt{dom}, i, j)] \leftarrow uSK_{\mathtt{dom},i,j}$.

$\mathcal{O}^{Ideal}_{\mathsf{GetNym}}$: The query requests the pseudonym of the ith user with regard to a domain name $\mathtt{dom}$. If $i \notin \mathcal{U}^{I}_{SET}$, then the oracle aborts. If $\mathcal{K}[(i, \mathtt{dom})]$ is undefined, then the oracle chooses a secret key $mSK_{i,\mathtt{dom}} \in \mathcal{USK}$ at random and sets $\mathcal{K}[(i, \mathtt{dom})] \leftarrow mSK_{i,\mathtt{dom}}$. Then the oracle runs $nym_{i,\mathtt{dom}} \leftarrow \mathsf{ComputePseudonym}\ (params, mSK_{i,\mathtt{dom}}, \mathtt{dom})$ and outputs $nym_{i,\mathtt{dom}}$.

$\mathcal{O}^{Ideal}_{\mathsf{GetRT}}$: The query requests a revocation token for the jth device of user i with regard to a domain name $\mathtt{dom}$. If $i \notin \mathcal{U}^{I}_{SET}$, then the oracle aborts. If the entry $\mathcal{UK}[(\mathtt{dom}, i, j)]$ is undefined, then the oracle runs the procedure $uSK_{\mathtt{dom},i,j} \leftarrow \mathsf{AddDevice}(param, mSK_{\mathtt{dom},i}, j)$, and sets $\mathcal{UK}[(\mathtt{dom}, i, j)] \leftarrow uSK_{\mathtt{dom},i,j}$. Then the oracle runs $uRT_{i,j,\mathtt{dom}} \leftarrow \mathsf{CreateRevocationToken}(param, mSK_{\mathtt{dom},i}, \mathtt{dom}, j)$ and outputs $uRT_{i,j,\mathtt{dom}}$.

$\mathcal{O}^{Ideal}_{\mathsf{Sign}}$: The query requests to sign a message m by the jth device of user i with regard to a domain name $\mathtt{dom}$. If $i \notin \mathcal{U}^{I}_{SET}$, then the oracle aborts and returns $\perp$. If $\mathcal{UK}[(\mathtt{dom}, i, j)]$ is undefined, then the oracle runs $uSK_{\mathtt{dom},i,j} \leftarrow \mathsf{AddDevice}(param, mSK_{\mathtt{dom},i}, j)$, and sets $\mathcal{UK}[(\mathtt{dom}, i, j)] \leftarrow uSK_{\mathtt{dom},i,j}$. Finally, the oracle runs $\sigma \leftarrow \mathsf{Sign}(param, uSK_{\mathtt{dom},i,j}, \mathtt{dom}, m)$ and returns σ.

Definition 9 We say that a Pseudonymous Public Key Group Signature S is (t, ϵ)-domain unlinkable if for any adversary A running in time t we have

$$\left|\Pr\left[(param) \leftarrow \mathsf{Setup}\left(1^{\lambda}\right); A^{\mathcal{O}_{Real}}(param)\right] - \Pr\left[(param) \leftarrow \mathsf{Setup}\left(1^{\lambda}\right); A^{\mathcal{O}_{Ideal}}(param)\right]\right| \leq \epsilon,$$

where $\mathcal{O}_{Real} = \{\mathcal{O}_{\mathsf{CreateUser}}, \mathcal{O}_{\mathsf{AddDevice}}, \mathcal{O}_{\mathsf{GetNym}}, \mathcal{O}_{\mathsf{GetRT}}, \mathcal{O}_{\mathsf{Sign}}\}$ and $\mathcal{O}_{Ideal} = \{\mathcal{O}^{Ideal}_{\mathsf{CreateUser}}, \mathcal{O}^{Ideal}_{\mathsf{AddDevice}}, \mathcal{O}^{Ideal}_{\mathsf{GetNym}}, \mathcal{O}^{Ideal}_{\mathsf{GetRT}}, \mathcal{O}^{Ideal}_{\mathsf{Sign}}\}$.

4 Efficient construction

4.1 Scheme specification

In this section, we describe our implementation of a Pseudonymous Public Key Group Signature.

The idea behind the construction is as follows. First a user chooses a secret key for the Boneh–Boyen signature scheme [7], i.e., $z \in \mathbb{Z}_p$ chosen at random. This key is then used to compute "pseudonymized" public keys as $nym \leftarrow \mathsf{H}_0(\mathtt{dom})^z$, where H_0 is a hash function and $\mathtt{dom}$ is a domain name (a bit string identifying the domain). The same key is then used to issue Boneh–Boyen signatures $A_j \leftarrow g_1^{1/(z+u_j)}$ on a secret key $u_j \in \mathbb{Z}_p$ of his device j. Note that according to our security definition from Sect. 3, the user generates all secret keys for his devices and we do not define a Join/Issue procedure to ensure exculpability.[1] We intentionally defined our group signature scheme in this way due to our specific use case.

Now, a device j holding a "certified" secret key (u_j, A_j), computes a signature of knowledge which is based on a Σ-protocol and turned into a signature scheme using the Fiat–Shamir paradigm. Informally, the signature carries a proof that the signer knows a secret key with a certificate which verifies correctly with a "pseudonymized" public key nym. This signature of knowledge may be summarized using the Camenisch–Stadler notation as follows:

$$SoK\left\{\left(u_j, A_j\right) : e\left(A_j, nym \cdot \hat{g_2}^{u_j}\right) = e\left(g_1, \hat{g_2}\right)\right\}$$

The tricky part of our construction is that the signer does not know the "pseudonymized" public key to which his certificate verifies. The only information which allows to sign with regard to a pseudonymized public key is the basis of the public key, i.e., $\hat{g_2} \leftarrow \mathsf{H}_0(\mathtt{dom})$. Additionally, the signature of knowledge reveals enough information to be able to blacklist a device with a domain specific revocation token.

Bellow, we describe our scheme more formally.

$\mathsf{Setup}(1^{\lambda})$:

1. Choose groups $\mathbb{G}_1, \mathbb{G}_2$ of a prime order p, a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, and choose a generator $g_1 \overset{\mathcal{R}}{\leftarrow} \mathbb{G}_1$ at random.
2. Define a hash function H_0 which maps into $\mathbb{G}_2$ and a hash function H which maps into $\mathbb{Z}_p$.
3. Output the global parameters $param = (p, \mathbb{G}_1, \mathbb{G}_2, e, g_1, \mathsf{H}_0, \mathsf{H})$.

$\mathsf{CreateUser}(param)$:

1. Choose $z \in \mathbb{Z}_p$ at random and output $mSK \leftarrow z$.

$\mathsf{ComputePseudonym}(param, mSK, \mathtt{dom})$:

1. Compute $\hat{g_2} \leftarrow \mathsf{H}_0(\mathtt{dom})$ and output $nym \leftarrow \hat{g_2}^z$.

$\mathsf{AddDevice}(param, mSK, i)$:

[1] The exculpability property is known from dynamic group signatures [5] and assures that even the group manager cannot forge signatures on behalf of a user.

1. Choose $u_i \in \mathbb{Z}_p$ at random.[2]
2. Compute $A_i = g_1^{1/(u_i+z)}$, return $uSK[i] \leftarrow (A_i, u_i)$ and store u_i for future use.

CreateRevocationToken$(param, mSK, \texttt{dom}, i)$:

1. Retrieve the user secret key u_i, compute $\hat{g_2} \leftarrow \mathsf{H}_0(\texttt{dom})$ and return $uRT \leftarrow \hat{g_2}^{u_i}$.

Sign$(param, uSK, \texttt{dom}, m)$:

1. Compute $\hat{g_2} \leftarrow \mathsf{H}_0(\texttt{dom})$.
2. Choose $(r_1, r_2) \in \mathbb{Z}_p^2$ at random and compute $R_1 \leftarrow A_i^{r_1}$, $R_2 \leftarrow g_1^{r_2}$ and $R_3 \leftarrow e(R_2, \hat{g_2})^{u_i}$.
3. Compute the following signature of knowledge:

$$S \leftarrow SoK\{(\alpha, \beta, \gamma) : R_1 = g_1^{\beta/(z+\alpha)} \wedge R_2 = g_1^{\gamma} \wedge R_3 = e(g_1, \hat{g_2})^{\alpha\cdot\gamma}\}(m)$$

   (a) Choose $t_1, t_2, t_3 \in \mathbb{Z}_p$ at random and compute

$$T_1 \leftarrow e\left(A_i, \hat{g_2}\right)^{-t_1\cdot r_1} \cdot e\left(g_1, \hat{g_2}\right)^{t_2},$$
$$T_2 \leftarrow g_1^{t_3} \text{ and}$$
$$T_3 \leftarrow e\left(R_2, \hat{g_2}\right)^{t_1}.$$

   (b) Compute the challenge $c = \mathsf{H}(param, m, \mathsf{dom}, T_1, T_2, T_3)$.
   (c) Compute $s_1 \leftarrow t_1 + c \cdot u_i$, $s_2 \leftarrow t_2 + c \cdot r_1$ and $s_3 \leftarrow t_3 + c \cdot r_2$.
   (d) Set $S = (c, s_1, s_2, s_3)$
4. Output the signature $\sigma = (S, R_1, R_2, R_3)$.

Verify$(param, nym, \texttt{dom}, \sigma, m, uRT)$:

1. Compute $\hat{g_2} \leftarrow \mathsf{H}_0(\texttt{dom})$.
2. Parse the signature as $\sigma = (S, R_1, R_2, R_3)$, where $S = (c, s_1, s_2, s_3)$.
3. Restore the values

$$\tilde{T_1} = e\left(R_1, nym\right)^{-c} \cdot e\left(R_1, \hat{g_2}\right)^{-s_1} \cdot e\left(g_1, \hat{g_2}\right)^{s_2}$$
$$\tilde{T_2} = g_1^{s_3} \cdot R_2^{-c}$$
$$\tilde{T_3} = e\left(R_2, \hat{g_2}\right)^{s_1} \cdot R_3^{-c}$$

4. If $c \neq \mathsf{H}(param, m, \mathsf{dom}, \tilde{T_1}, \tilde{T_2}, \tilde{T_3})$, then return 0 (reject).
5. If $e(R_2, uRT) = R_3$, then return 0 (reject).
6. Return 1 (accept).

Theorem 1 *Pseudonymous Public Key Group Signature is correct.*

[2] This value may be derived in a deterministic way, e.g., $u_i \leftarrow \mathsf{H}(z, i)$.

Proof The proof is simply due the inspection of the following equations:

$$\begin{aligned}
\tilde{T_1} &= e\left(R_1, nym\right)^{-c} \cdot e\left(R_1, \hat{g_2}\right)^{-s_1} \cdot e\left(g_1, \hat{g_2}\right)^{s_2} \\
&= \left(e\left(R_1, \hat{g_2}\right)^{-t_1} \cdot e\left(g_1, \hat{g_2}\right)^{t_2}\right) \\
&\quad \cdot e\left(R_1, nym\right)^{-c} \cdot e\left(R_1, \hat{g_2}^{-c\cdot u_i}\right) \cdot e\left(g_1^{c\cdot r_1}, \hat{g_2}\right) \\
&= T_1 \cdot e\left(A_i^{r_1}, \hat{g_2}^{-c\cdot z} \cdot \hat{g_2}^{-c\cdot u_i}\right) \cdot e\left(g_1^{c\cdot r_1}, \hat{g_2}\right) \\
&= T_1 \cdot e\left(g_1, \hat{g_2}\right)^{-r_1\cdot c} \cdot e\left(g_1, \hat{g_2}\right)^{r_1\cdot c} = T_1 \\
\tilde{T_2} &= g_1^{s_3} \cdot R_2^{-c} = g_1^{t_3} \cdot g_1^{c\cdot r_2} \cdot g_1^{-c\cdot r_2} = T_2 \\
\tilde{T_3} &= e\left(R_2, \hat{g_2}\right)^{s_1} \cdot R_3^{-c} \\
&= e\left(R_1, \hat{g_2}\right)^{t_1} \cdot e\left(g_1, \hat{g_2}\right)^{r_2\cdot c\cdot u_i} \cdot e\left(g_1, \hat{g_2}\right)^{-r_2\cdot c\cdot u_i} = T_3
\end{aligned}$$

For the revocation procedure, let $uRT \leftarrow \hat{g_2}^{u_i}$ be a revocation token. Then we have

$$e\left(R_2, uRT\right) = e\left(g_1^{r_2}, \hat{g_2}^{u_i}\right) = R_3.$$

□

4.2 Notes on the construction

In this section, we discuss some efficiency and implementation details. Moreover, we point also some potential modifications which may be valuable in a real-world deployment.

4.2.1 Efficiency and optimizations

Creating a signature as described in Sect. 4 takes 3 exponentiations in $\mathbb{G}_1$, 4 exponentiations in $\mathbb{G}_T$, 1 multiplication in $\mathbb{G}_T$ and 3 pairing operations. However, we may optimize the scheme by computing only $e(A, \hat{g_2})$ and $e(g_1, \hat{g_2})$, what also may be precomputed, but then we have to store this values for all domains separately. Then, the values R_3 and T_3 may be computed as $R_3 \leftarrow e(g_1, \hat{g_2})^{r_2\cdot u}$ and $T_3 \leftarrow e(g_1, \hat{g_2})^{r_2\cdot t_1}$. Thus, the number of pairing evaluations may be reduced to 2 pairings, or zero pairings in case the values corresponding to a domain are precomputed.

Signature verification, as described in Sect. 4, takes 2 exponentiations and 1 multiplication in $\mathbb{G}_1$, 5 exponentiations and 3 multiplications in $\mathbb{G}_T$, and 4 pairing evaluations. We may get rid of one pairing evaluation if $e(g_1, \hat{g_2})$, what is quite likely in a real-world implementation. We also may reduce the amount of exponentiations in $\mathbb{G}_T$, by computing $e(R_1^{-c}, nym)$, $e(R_1^{-s_1}, \hat{g_2})$ and $e(R_2^{s_2}, \hat{g_2})$. Later, the revocation part of the signature verification takes one pairing evaluation per revocation token. Unfortunately, we need to check all revocation tokens corresponding to the pseudonyms nym. Hence the revocation procedure runs in time $O(|RT|)$,

where $|RT|$ stands for the number of revocation tokens published the owner of nym in the domain.

4.2.2 Implementation

Now we will describe the efficiency of our scheme in a concrete implementation. We consider Barreto–Naehrig (BN) curves. In our setting, the group $\mathbb{G}_1$ is instantiated over $\mathbb{F}_q$, the group $\mathbb{G}_2$ is instantiated over $\mathbb{F}_{q^2}$ and the target group over $\mathbb{F}_{q^{12}}$, where q is the size of the underlying finite field $\mathbb{F}_p$. BN curves are of the form $E : y^2 = x^3 + b$, for $b \neq 0$. For our tests, we chose the Fp256BN curve which has been standardized by ISO/EIC [20] and specified in IETF draft [21]. However, due to recent results [2], it provides approximately 100-bit security.

We did our tests on a proof-of-concept implementation using the Java Pairing-Based Cryptography Library v.2.0.0 [15], a Java port of the PBC library written in C [25,30]. We run our tests on a machine with Intel i7-2670QM 2.20 GHz CPU and 8 GB of random access memory.

Although our tests give some intuition on the concrete timings, they may differ depending on the quality of the implementation, underlying hardware, etc. In order to highlight the timing differences between different groups, we will additionally use the framework given in [18]. We will normalize the operations made in every group by estimating the cost in multiplications made in $\mathbb{F}_q$ what we denote as M_q. So, according to [18] one exponentiation (scalar multiplication) in $\mathbb{G}_1$ costs approximately $2738 \cdot M_q$, exponentiation in $\mathbb{G}_2$ costs approximately $6590 \cdot M_q$ and in $\mathbb{G}_T$ approximately $9252 \cdot M_q$. Similarly, multiplication (point addition) costs $11 \cdot M_q$ in $\mathbb{G}_1$, $29 \cdot M_q$ in $\mathbb{G}_2$ and $54 \cdot M_q$ in $\mathbb{G}_T$. Computing a pairing costs approximately $16{,}336 \cdot M_q$.

Since we introduce a new notion for group signatures which aims to fit an application for which previous group signature scheme where not designed for, it cannot directly be compared with previous work. Despite this, we compare our solution with verifier-local revocation group signature schemes (VLR group signatures) from [6,9,26] since these primitives share some functionalities with our proposed solution. VLR group signatures are group signatures which provide a revocation functionality, which does not require any group member (device in our setting) to update its state. However, we emphasize that the above-mentioned VRL group signatures do not support pseudonymous group public keys as in our scheme. We compare the signature size in Table 1, the complexity of signature generation in Table 2 and the complexity of signature verification in Table 3. For the signature size, we give the bit length according to the chosen parameters, and for signature generation and verification we provide the timings in milliseconds.

Table 1 Signature size

Scheme	$\mathbb{G}_1$	$\mathbb{G}_2$	$\mathbb{G}_T$	$\mathbb{Z}_q$	Bit size
Our	2	0	1	4	4608
[9]	2	–	–	5	1792
[6]	3	–	–	2	1280
[26]	3	–	1	8	5888

Table 2 Computational cost of signature generation

Scheme	Exp.	Mul.	Pairing	$\psi : \mathbb{G}_2 \to \mathbb{G}_1$	M_p	Impl. (ms)
Our	$3 \cdot \mathbb{G}_1 + 4 \cdot \mathbb{G}_T$	$1 \cdot \mathbb{G}_T$	–	–	45,276	176
[9]	$6 \cdot \mathbb{G}_1 + 2 \cdot \mathbb{G}_2$	$2 \cdot \mathbb{G}_1 + 1 \cdot \mathbb{G}_2 + 1 \cdot \mathbb{G}_T$	2	2	62,385	482
[6]	$3 \cdot \mathbb{G}_1 + 1 \cdot \mathbb{G}_T$	–	–	–	35,970	165
[26]	$13 \cdot \mathbb{G}_1 + 4 \cdot \mathbb{G}_T$	$6 \cdot \mathbb{G}_1 + 2 \cdot \mathbb{G}_T$	1	–	89,062	477

Table 3 Computational cost of signature verification

Scheme	Exp.	Mul.	Pairing	$\psi : \mathbb{G}_2 \to \mathbb{G}_1$	M_p	Impl. (ms)
Our	$6 \cdot \mathbb{G}_1 + 1 \cdot \mathbb{G}_T$	$2 \cdot \mathbb{G}_1 + 2 \cdot \mathbb{G}_T$	3	–	74,818	457
[9]	$6 \cdot \mathbb{G}_1 + 2 \cdot \mathbb{G}_2 + 1 \cdot \mathbb{G}_T$	$2 \cdot \mathbb{G}_1 + 3 \cdot \mathbb{G}_T$	3	2	88,052	552
[6]	$3 \cdot \mathbb{G}_1$	$1 \cdot \mathbb{G}_1 + 1 \cdot \mathbb{G}_T$	4	–	73,623	442
[26]	$12 \cdot \mathbb{G}_1 + 2 \cdot \mathbb{G}_2 + 3 \cdot \mathbb{G}_T$	$7 \cdot \mathbb{G}_1 + 2 \cdot \mathbb{G}_2 + 4 \cdot \mathbb{G}_T$	2	–	106,815	667

As shown in Table 1, the size of our signatures is between the solutions from [6,9,26]. As for signature generation and verification, we may observe that our scheme is faster than [9,26]. However, it is slightly slower than the solution LRSW based solution [6].

4.2.3 Additional procedures and scheme variants

Here we describe briefly some additional procedures and variations of our scheme, which may be useful for certain practical situations.

First, note that the signing device needs only to know its private key consisting of an SDH pair $(u, g_1^{1/(z+u)})$ and nothing else, in order to create a signature. In particular, the signing device does not need to know the public key, a.k.a. the pseudonym, with which the signature will later be verified. Moreover, it seems that the signing device alone is not even able to compute the pseudonym by itself. However, in some cases it may be desirable that the signing device can compute a pseudonym, what in our case may be $nym' = e(Z, \hat{g_2})$, assuming the user also issues the value $Z = g_1^z$. Such nym' may serve as a temporal pseudonym, until the owner of the device confirms this pseudonym by proving his knowledge of the secret key $z \in \mathbb{Z}_p$.

Proving the knowledge of the master key may be required as a part of user registration. This may be simply done by designing a Σ-protocol [16] which will prove the knowledge of $\log_{g_1}(nym)$. Such standard protocol may be transformed into a zero-knowledge proof of knowledge protocol or into its non-interactive version in the random oracle model.

5 Security analysis

In this section, we formally proof the security properties of our proposed scheme. In order to facilitate the understanding of the proofs, we will first informally outline the proof and then give the full formal proof of the corresponding theorems.

Zero-knowledge and witness extraction Our construction is based on a known technique of using a Σ-protocol converted into a signature scheme via the Fiat–Shamir heuristic. For such a construction, we may show that, in the random oracle model, there is a witness extractor (so the protocol is a proof of knowledge) and a simulator (so the protocol is zero-knowledge). Using the witness extractor, from a forged signature for a pseudonym nym within domain $\mathtt{dom}$, we may extract values $\tilde{u}$, $\tilde{r_1}$, $\tilde{r_2}$ and $\tilde{A}$, such that $g_1^{\tilde{r_2}} = R_2$, $e(R_2, \mathsf{H}_0(\mathtt{dom}))^{\tilde{u}} = R_3$ and $e(g_1, \mathsf{H}_0(\mathtt{dom})) = e(\tilde{A}, \mathsf{H}_0(\mathtt{dom})^{\tilde{u}} \cdot nym)$. Using the simulator, we may generate a correct signature having only $g_1, \hat{g_2}, nym$ and a revocation token $\mathsf{H}_0(\mathtt{dom})^{\tilde{u}}$.

In this section, we will use the zero-knowledge and witness extraction properties as subprocedures. We describe the underlying Σ-protocol and proof the above-mentioned properties in Sect. 6

5.1 Unforgeability

Theorem 2 *If DLP is (ϵ', t')-hard in $\mathbb{G}_2$, then the Pseudonymous Public Key Group Signature is (ϵ, t)-unforgeable, where $\epsilon \approx q_U \cdot n \cdot \sqrt{q_\mathsf{H}(\epsilon' + 1/p)}$ and $t \approx t'$, and n, q_U and q_H are the upper bounds on the number of invocations of, respectively, $\mathcal{O}_{\mathsf{CreateUser}}$, $\mathcal{O}_{\mathsf{AddDevice}}$ and hash queries.*

The unforgeability property relies on the DLP problem. Here we put a DL problem instance $\Lambda \in \mathbb{G}_2$ into the revocation tokens of a chosen device. We may program the random oracle to output $g_2^{r_{\mathtt{dom}}} \leftarrow \mathsf{H}_0(\mathtt{dom})$, and then we may compute the revocation tokens as $uRT \leftarrow \Lambda^{r_{\mathtt{dom}}}$. If the adversary successfully forges a signature for that device, we use the extractor and extract the discrete logarithm $\alpha = \log_{g_2}(\Lambda)$.

Proof Let A be an adversary breaking the unforgeability of our scheme. We will construct a solver B, which exploits A to compute $\log_{g_2}(\Lambda)$.

The solver creates random oracles H, H_0 and sets the public parameters as $param = (p, \mathbb{G}_1, \mathbb{G}_2, e, g_1, \mathsf{H}_0, \mathsf{H})$. Then the solver chooses a user index $i' \in \{1, \ldots, n\}$ and a device index $j' \in \{1, \ldots, q_U\}$ at random. The solver starts the adversary giving him as input the public parameters $param$.

Then B handles the queries as follows:

- Hash Queries: In case the adversary queries the H hash oracle, B responds with a random value and saves the response for a future call. In case the adversary queries the H_0 hash oracle on input $\mathtt{dom}$, the solver chooses $r \xleftarrow{\mathcal{R}} \mathbb{Z}_p$ at random, programs the oracle to return g_2^r and saves the pair $(r, \mathtt{dom})$ for a future call.
- $\mathcal{O}_{\mathsf{CreateUser}}$: The adversary requests to create the ith user. The solver runs $mSK_i \leftarrow \mathsf{CreateUser}(param)$ as in the protocol description and adds the pair (i, mSK_i) to $\mathcal{U}_{SET}$.
- $\mathcal{O}_{\mathsf{GetNym}}$: The adversary requests the pseudonym of the ith user with regard to domain $\mathtt{dom}$. The solver runs $nym_{i,\mathtt{dom}} \leftarrow \mathsf{ComputePseudonym}(param, \mathtt{dom}, mSK_i)$ and returns $nym_{i,\mathtt{dom}}$.
- $\mathcal{O}_{\mathsf{AddDevice}}$: The adversary requests to add a device j by the ith user. If $i \neq i'$, then B runs $uSK_{i,j} \leftarrow \mathsf{AddDevice}(param, mSK, j)$ as in the original protocol. If $i = i'$, then B acts as follows:
  - if $j \neq j'$, then B runs $uSK_{i',j} \leftarrow \mathsf{AddDevice}(param, mSK, j)$ as in the original protocol.
  - if $j = j'$, then B sets $uSK_{i',j'} \leftarrow \perp$ (the solver will later simulate the signatures for this device).

Finally, the solver adds the tuple $(i, j, uSK_{i,j})$ to $\mathcal{D}_{SET}$.

- $\mathcal{O}_{\mathsf{GetRT}}$: The adversary requests the domain revocation token of the ith user, jth device with regard to domain $\mathtt{dom}$. If $i = i'$ and $j = j'$, then the solver restores the pair $(r, \mathtt{dom})$ from the H_0 hash oracle, computes $uRT \leftarrow \Lambda^r$, and outputs uRT. Otherwise, the solver computes $uRT \leftarrow \mathsf{CreateRevocationToken}(param, mSK, \mathtt{dom}, i)$ as in the protocol description and returns uRT.
- $\mathcal{O}_{\mathsf{Sign}}$: The adversary requests a signature for the ith user, jth device within domain $\mathtt{dom}$ and message m. If $i = i'$ and $j = j'$, then B simulates the signature of knowledge. Otherwise, the solver computes $\sigma \leftarrow \mathsf{Sign}(param, uSK_{i,j}, \mathtt{dom}, m)$ as in the protocol description. Finally, the solver outputs σ.
- $\mathcal{O}_{\mathsf{CorruptDevice}}$: The adversary requests a secret key of the ith group manager and jth user. If $i = i'$ and $j = j'$, then the solver aborts. Otherwise, B returns $uSK_{i,j}$.

At some point, the adversary returns a signature σ^*, a message m^*, a domain string $\mathtt{dom}$ and a pseudonym nym^*.

First, the solver restores the pair $(r, \mathtt{dom})$ from the H_0 hash oracle. With probability at least $1/n$, the solver choose the index i properly; thus, nym^* corresponds to the i'th user. Then, with probability at least $1/q_U$, the signature corresponds to the j'th device. If the signature corresponds to another device, then the solver aborts. Otherwise, B runs the extractor of the signature of knowledge and obtains values $(\tilde{u}, \tilde{r}_1, \tilde{r}_2) \in \mathbb{Z}_p^3$ and $\tilde{A} \in \mathbb{G}_1$ which satisfy $R_2 = g_1^{\tilde{r}_1}$, $R_3 = g_1^{\tilde{r}_1 \cdot \tilde{u}}\, e(\tilde{A}, g_2^{r \cdot \tilde{u}} \cdot nym)$, and the revocation equation $e(R_2, \Lambda^r) = e(R_3, g_2^r)$ holds. Thus, we have that $e(g_1^{r_1}, \Lambda) = e(g_1^{r_1 \cdot \tilde{u}}, g_2)$ and, what follows, $\tilde{u} = \log_{g_2}(\Lambda)$.

Simulating, the signature of knowledge may cause abortion due to a collision in the hash oracle H. Since the hash oracle takes three independently chosen random values T_1, T_2 and T_3, the probability of a collision is at most $2 \cdot q_{\mathsf{H}}^2 / p^3$. Assuming $q_{\mathsf{H}} << p$, this probability is negligible hence we omit it for readability.

Finally, the solver obtains a signature forged for the i'th user and j'th device with probability $\epsilon'' = \frac{\epsilon}{q_U \cdot n}$. By applying the forking lemma from [4], the solver may extract $\log_{g_2}(\Lambda)$ with probability at least $\epsilon''^2 / q_{\mathsf{H}} - 1/p$. □

5.2 Seclusiveness

Theorem 3 *If q-CAA is (ϵ', t')-hard in $\mathbb{G}_1$, then the Pseudonymous Public Key Group Signature is (ϵ, t)-seclusive, where $\epsilon \approx n \cdot \sqrt{q_{\mathsf{H}}(\epsilon' + 1/p)}$, $t \approx t'$, and n, q and q_{H} are the upper bounds on the number of, respectively, $\mathcal{O}_{\mathsf{CreateUser}}$, $\mathcal{O}_{\mathsf{AddDevice}}$ and hash queries.*

Seclusiveness follows from the fact that device secret keys are CAA instances, i.e., they consist of pairs $(u, g_1^{1/(u+z)}) \in \mathbb{Z}_p \times \mathbb{G}_1$. If an adversary would forge a signature, then from the extractor we may obtain a pair $(\tilde{u}, \tilde{A})$. If the forged signature cannot be revoked, then from the revocation equation $e(R_2, \hat{g_2}^{u_i}) \neq R_3$ follows that $\tilde{u} \neq u_i$ for each device secret key u_i issued by the user holding z. Thus, $(\tilde{u}, \tilde{A})$ is the solution to the CAA problem instance.

Proof Let A be an adversary breaking the seclusiveness of our scheme. We construct a solver B, which exploits A to solve the CAA problem instance.

The solver obtains $(g_1, g_1^z, g_2, g_2^z) \in \mathbb{G}_1^2 \times \mathbb{G}_2^2$ and q pairs $(A_j, u_j) \in \mathbb{G}_1 \times \mathbb{Z}_p$ form the problem instance. Then, the solver creates random oracles H and H_0, sets the global parameters as $param = (p, \mathbb{G}_1, \mathbb{G}_2, e, g_1, \mathsf{H}_0, \mathsf{H})$ and chooses a user index $i' \in \{1, \ldots, n\}$ at random. The solver programs the H_0 hash oracle such that on input $\mathtt{dom}$, the solver chooses $r \overset{\mathcal{R}}{\leftarrow} \mathbb{Z}_p$ at random, programs the oracle to return g_2^r and saves the pair $(r, \mathtt{dom})$ for a future call.

The crucial observation is that the pairs (A_j, u_j) form valid device secret keys. Hence, for the i'th call of the $\mathcal{O}_{\mathsf{CreateUser}}$ oracle the solver adds the pair $(i', \perp)$ to $\mathcal{U}_{SET}$. It is easy to see that all oracle queries, except those involving the i'th user, are handled according to the protocol description.

For the i'th user, the solver handles the $\mathcal{O}_{\mathsf{GetNym}}$ queries with domain string $\mathtt{dom}$ by computing $nym_{i',\mathtt{dom}} \leftarrow (g_2^z)^r$, where $r \in \mathbb{Z}_p$ is recovered from the H_0 hash oracle for input $\mathtt{dom}$. Then B returns $nym_{i',\mathtt{dom}}$. In case of $\mathcal{O}_{\mathsf{AddCorruptedDevice}}$ oracle query for a device j and the i'th user, the solver simply returns the pair (A_j, u_j) from the problem instance. If the adversary calls the $\mathcal{O}_{\mathsf{GetRT}}$ oracle for a user j and the i'th user, then the solver obtains $\hat{g_2} \leftarrow \mathsf{H}_0(\mathtt{dom})$ and returns $uRT_{i,j} \leftarrow \hat{g_2}^{u_j}$.

At some point the adversary returns a signature σ^*, a message m^*, a domain string $\mathtt{dom}$ and a pseudonym nym^*. The solver obtains the pair $(r, \mathtt{dom})$ from the hash oracle H_0. Then the solver may check whether the forge was made for the i'th user by inspecting the equation $(g_2^z)^r = nym$. If this is not the case then the solver aborts. The solver does not abort with probability $1/n$.

Then, the solver applies the extractor of the signature of knowledge and obtains $(\tilde{u}, \tilde{r}_1, \tilde{r}_2) \in \mathbb{Z}_p^3$ and $\tilde{A} \in \mathbb{G}_1$ which satisfy $R_2 = g_1^{\tilde{r}_1}$, $R_3 = g_1^{\tilde{r}_1 \cdot \tilde{u}}\, e(\tilde{A}, g_2^{r \cdot \tilde{u}} \cdot nym)$. Then, for each $i \in \{1, \ldots, q\}$, we have

$$e\left(R_2, g_2^{u_i \cdot r}\right) = e\left(g_1^{\tilde{r}_1}, g_2^{u_i \cdot r}\right) \neq e\left(g_1^{r_1 \cdot \tilde{u}}, g_2^r\right).$$

Thus, $\tilde{u} \neq u_i$ and the pair $(\tilde{A}, \tilde{u})$ are the solution to the CAA problem instance.

So, the solver obtains a forged signature for user i' with probability at least $\epsilon'' = \epsilon / n$. By applying the forking lemma from [4], the solver may extract $(\tilde{A}, \tilde{u})$ with probability at least $\epsilon''^2 / q_{\mathsf{H}} - 1/p$. □

5.3 Anonymity

Theorem 4 *If BDDH is (ϵ', t')-hard in $\mathbb{G}_1, \mathbb{G}_2$, then the Pseudonymous Public Key Group Signature is (ϵ, t)-anonymous, where $\epsilon \approx \frac{\epsilon'^2}{n \cdot q_{\mathsf{H}} \cdot q_U^2}$, $t \approx t'$, and n, q_U and q_{H} are upper bounds on the number of, respectively, $\mathcal{O}_{\mathsf{CreateUser}}$, $\mathcal{O}_{\mathsf{AddDevice}}$ and hash queries.*

In order to proof anonymity, we describe a sequence of games. We will start with a game where the challenge signature is returned by the device j_0^* (bit $b = 0$). Then in each game we change the protocol execution so that the adversary has only a negligible chance of noticing these changes. Finally, we will end up in a game where the challenge signature is computed for device j_1^* (bit $b = 1$).

The strategy of changing the protocol is as follows. First we need to simulate the signatures for all devices. Then, for the j_0^*th device, under the BDDH assumption, we choose these values independently at random. Next, instead of choosing R_1, R_2, R_3 independently we compute these values as for device j_1^*. Below, we shed some light on the step of changing the values of R_1, R_2, R_3 into random values.

Let (g_2^a, g_2^b, g_1^c) be a BDDH problem instance and let $\mathtt{dom}^*$ denote the domain from the challenge oracle. In all domains $\mathtt{dom} \neq \mathtt{dom}^*$ we choose $r_{\mathtt{dom}}$ at random, program the hash oracle to output $g_2^{r_{\mathtt{dom}}} \leftarrow \mathsf{H}_0(\mathtt{dom})$ and we compute $uRT \leftarrow (g_2^a)^{r_{\mathtt{dom}}}$ and $R_3 \leftarrow e(g_1^{r_{\mathtt{dom}}}, g_2^a)^{r_2}$ for device j_0^*. In domain $\mathtt{dom}^*$, we program the hash oracle to return $g_2^b \leftarrow \mathsf{H}_0(\mathtt{dom}^*)$. Then, we choose $r_2 \xleftarrow{\mathcal{R}} \mathbb{Z}_p$ at random, compute $R_2 \leftarrow g_1^{c \cdot r_2}$ and $R_3 \leftarrow e(g_1, g_2)^{abc \cdot r_2}$(the current game) or R_3 is chosen at random (the next game). Note that if an adversary would be able to distinguish whether $R_3 = e(g_1,g_2)^{abc \cdot r_2}$ or R_3 is random, then it would break the BDDH assumption.

Proof The proof of anonymity the sequence of games described below.

Game 0. This is the game where we run our scheme.

Game 1. From this game we assume that the adversary calls the challenge oracle on the dth domain string, where d is chosen at random from $\{1, \ldots, q_{\mathsf{H}}\}$ and q_{H} is the upper bound of domain strings. It is easy to see, that the solver guesses d with probability $1/q_{\mathsf{H}}$.

Game 2. From now on, we assume that the adversary calls the challenge oracle for the ith user, where i is chosen at random from $\{1, \ldots, n\}$. The probability that the solver guesses this index is $1/n$. We denote as nym the pseudonym of the ith user in the dth domain.

Game 3. We assume that the adversary calls the challenge oracle for the jth and j'th devices, where j and j' are randomly chosen from $\{1, \ldots, q_U\}$. The probability that the solver guesses this indexes is $1/q_U(q_U - 1) \approx 1/q_U^2$.

Game 4. The H_0 oracle calls are handled as follows. First we choose $r \xleftarrow{\mathcal{R}} \mathbb{Z}_p$ at random, and then we program the oracle to return g_2^r. Then we save the value r for a future call with the same input.

Game 5. For devices $j_0 \in \{j, j'\}$, we let the add device oracle compute $uSK_{i,j_0} \leftarrow g_1^{u_{i,j_0}}$. Queries for revocation tokens are handled by restoring $r \in \mathbb{Z}_p$ from the H_0 oracle and computing $uRT_{i,j_0} \leftarrow (g_2^{u_{i,j_0}})^r$. We simulate the signatures for this devices as follows: We choose $(r_1, r_2) \xleftarrow{\mathcal{R}} \mathbb{Z}_p$ at random and compute $R_1 \leftarrow g_1^{r_1}$, $R_2 \leftarrow g_1^{r_2}$ and $R_3 \leftarrow e(g_2^{u_{i,j_0}}, g_2^r)^{r_2}$. Then we choose $(c, s_1, s_2, s_3) \xleftarrow{\mathcal{R}} \mathbb{Z}_p^4$ and compute the values T_1, T_2 and T_3 according to the simulator described in Sect. 6. Simulating, the signature of knowledge may cause abortion due to a collision in the hash oracle H what may happen with probability at most $q_s(q_{\mathsf{H}} + q_s)/p^3$.

Game 6. In domain $\mathtt{dom}^*$ for the jth device, we choose the value $R_3 \xleftarrow{\mathcal{R}} \mathbb{G}_T$ at random. Note, that in all other domains for the jth device we compute $R_3 \leftarrow e(g_2^{u_{i,j}}, g_2^r)^{r_2}$.

Claim *If the BDDH problem is (ϵ', t)-hard, then the adversary has $\epsilon'' \approx \epsilon'$ advantage in distinguishing between Game 5 and Game 6.*

Proof Given $(g_2^a, g_2^b, g_1^c, X) \in \mathbb{G}_2^2 \times \mathbb{G}_1 \times \mathbb{G}_T$, we construct a solver which uses an efficient distinguisher to determine whether $X = e(g_1, g_2)^{abc}$ or X is random.

In all domains $\mathtt{dom} \neq \mathtt{dom}^*$, the solver computes $uRT \leftarrow (g_2^a)^r$, where r is the exponent stored by the H_0 hash oracle, and $R_3 \leftarrow e(g_1^r, g_2^a)^{r_2}$. All other values are computed as in Game 5.

Then, within domain $\mathtt{dom}^*$, the solver programs the H_0 hash oracle to return g_2^b. Finally, for device j the solver chooses $r_2 \xleftarrow{\mathcal{R}} \mathbb{Z}_p$ at random and computes $R_2 \leftarrow g_1^{c \cdot r_2}$ and $R_3 \leftarrow X^{r_2}$.

Now, if the distinguisher outputs that he is in Game 5, the solver outputs that $X = e(g_1, g_2)^{abc}$. Otherwise, the solver outputs that X is random.

Game 7. In domain $\mathtt{dom}^*$ for the j'th user, we choose the value $R_3 \xleftarrow{\mathcal{R}} \mathbb{G}_T$ at random.

Claim *If the BDDH problem is (ϵ', t)-hard, then the adversary has $\epsilon'' \approx \epsilon'$ advantage in distinguishing between Game 6 and Game 7.*

Proof The proof is exactly the same as the proof of claim 5.3. □

Finally, we end up with a system, where the signatures of the devices from the challenge phase are independent of their secret keys. The part of the signature consisting of values R_1, R_2, R_3 are chosen independently at random, and we may write $R_3 = e(g_1^u, g_2^r)^{r_2}$ for each device from the challenge phase. Thus, an adversary may only guess the bit and win with probability $1/2$.

5.4 Domain unlinkability

Theorem 5 *If SXDH is (ϵ', t')-hard in $\mathbb{G}_2$, then the Pseudonymous Public Key Group Signature is (ϵ, t)-domain unlinkable, where $\epsilon \approx \epsilon' \cdot q_{\mathsf{H}}(q_U + n)$, $t \approx t'$, and n, q_U and q_{H} are the upper bounds on the number of, respectively, $\mathcal{O}_{CreateUser}$, $\mathcal{O}_{AddDevice}$ and hash queries.*

Domain unlinkability follows from the fact that we may simulate the signatures for each device and that in each domain we have a distinct base $\hat{g_2} \leftarrow \mathsf{H}_0(\mathtt{dom})$. Note that the device revocation tokens are computed as $uRT_{i,j,\mathtt{dom}} = \hat{g_2}^{u_j}$ for a device secret key $u_j \in \mathbb{Z}_p$. In a given domain we may choose $uRT_{i,j,\mathtt{dom}}$ at random. It is easy to see that if an adversary would recognize this change, he would serve as a distinguisher for the SXDH problem. In the proof, we need to choose revocation tokens of all devices in all domains at random. Finally, we may use the same reasoning to choose pseudonyms $nym = \mathsf{H}_0(\mathtt{dom})$ at random in each domain, finally ending up in an ideal system as defined in Sect. 3.

Proof Below we describe the sequence of games, starting from the Pseudonymous Public Key Group Signature and ending up with a game with the ideal scheme.

Game 0. We run our scheme in the domain anonymity environment.

Game 1. The H_0 oracle calls are handled by choosing $r \xleftarrow{\mathcal{R}} \mathbb{Z}_p$ at random, and then programming the oracle to return g_2^r. Then we save the value r for a future call with the same input.

Game 2. If the adversary requests to sign a message m, by the $j$the device of the ith user with regard to a domain string $\mathtt{dom}$ (domain pseudonym $nym_{i,j}$), we simulate the signature as follows.
First, the simulator obtains the scalar $r \in \mathbb{Z}_p$ from the H_0 hash oracle. Then the simulator chooses $(r_1, r_2) \xleftarrow{\mathcal{R}} \mathbb{Z}_p$ at random and computes $R_1 \leftarrow g_1^{r_1}$, $R_2 \leftarrow g_2^{r_2}$ and $R_3 \leftarrow e(g_1, g_2^{u_{i,j} \cdot r})^{r_2}$. Then the simulator chooses $(c, s_1, s_2, s_3) \xleftarrow{\mathcal{R}} \mathbb{Z}_p^4$, computes the values T_1, T_2 and T_3 according to the simulator described in Sect. 6.

Game $(3, j', d)$: Let us first note that we may assign a unique number to each device, which is upperbounded by the number add device oracle calls q_U. For simplicity, we will denote a device secret key as $u_{j'}$ instead of $u_{i,j}$ having in mind that all $j' \in \{1, \ldots, q_U\}$ represent all devices of all users in the system. Also as the number of domain strings is upperbounded by the number of hash queries q_{H}, we may assign a number to all domain strings in the system. So, we will denote as $d \in \{1, \ldots, q_{\mathsf{H}}\}$ the dth domain string. Denote, as $\mathtt{dom}_d$ the dth domain string. Now, we will incrementally change the game, starting from the first device and the first domain. So instead of computing $uRT \leftarrow g_2^{u_{j' \cdot r_d}}$ and $R_3 \leftarrow e(g_1, g_2^{u_{j' \cdot r_d}})^{r_2}$, for $r_d \in \mathbb{Z}_p$ such that $\mathsf{H}(\mathtt{dom}_d) = g_2^{r_d}$ in Game $(3, j', d)$, we choose a random $u_{j',d}$ and compute $uRT \leftarrow g_2^{u_{j',d}}$ and $R_3 \leftarrow e(g_1, g_2^{u_{j',d}})^{r_2}$ in Game $(3, j', d+1)$. Note, that between Game $(3, j', q_{\mathsf{H}})$ and Game $(3, j'+1, 1)$, there is no change. Finally, we will end up with Game $(3, q_U, q_{\mathsf{H}})$, where all device secret keys are independently chosen for each domain string.

Claim *If SXDH is (ϵ', t)-hard, then the adversary has $\epsilon'' \approx \epsilon'$ advantage in distinguishing between Game $(3, j', d)$ and Game $(3, j', d+1)$.*

Proof Given $(g_2^\alpha, g_2^\beta, g_2^\gamma) \in \mathbb{G}_2^3$, we construct a solver which uses a distinguisher to distinguish whether $\gamma = \alpha \cdot \beta$ or γ is random.

In Game $(3, j', d)$ The solver computes $uRT \leftarrow (g_2^\alpha)^r$ and $R_3 \leftarrow e(g_1, (g_2^\alpha)^r)^{r_2}$ in all domains $d' \geq d$. Then in Game $(3, j', d+1)$ for domain $\mathtt{dom}_d$ the solver programs the H_0 oracle to return g_2^β on input $\mathtt{dom}_d$, For $\mathtt{dom}_d$ and device j' the solver computes $uRT \leftarrow g^\gamma$ and $R_3 \leftarrow e(g_1, g_2^\gamma)^{r_2}$. All other values are computed as in Game $(3, j', d)$.

Now, note that if $\gamma = \alpha \cdot \beta$, we have $uRT \leftarrow (g_2^{\alpha \cdot \beta})$ and $R_3 \leftarrow e(g_1, (g_2^{\alpha \cdot \beta}))^{r_2}$, and the distinguisher will output that he is in game Game $(3, j', d)$ with advantage ϵ'. Otherwise, we may write $uRT \leftarrow g^{\alpha' \cdot \beta}$ and $R_3 \leftarrow e(g_1, g_2^{\alpha' \cdot \beta})^{r_2}$, for

a random α' and the distinguisher will output that he is in Game $(3, j', d+1)$. □

To sum up, we execute Game 3 for all devices of all users in all domains. Having that there may be at most q_U users and q_H domain strings, we have that an adversary may distinguish between Game 2 and Game $(3, q_U, q_\mathsf{H})$ with advantage $\epsilon' \cdot q_U \cdot q_\mathsf{H}$.

Game $(4, i, d)$: Similarly, as in the previous sequence of games, here we will iterate for each user $i \in \{1, \ldots, n\}$ and each domain $d \in \{1, \ldots, q_\mathsf{H}\}$.

In Game $(4, i, d)$ we compute $nym \leftarrow g_2^{z_i \cdot r_d}$, where r_d is the scalar from the H_0 hash oracle for $\mathtt{dom}_d$, and z_i stands for the ith user secret key. In Game $(4, i, d+1)$ we choose $nym \xleftarrow{\mathcal{R}} \mathbb{G}_2$ at random.

Finally, we will end up with Game $(4, n, q_\mathsf{H})$ in which each group manager will have a separate secret key in each domain.

Claim *If the SXDH problem is (ϵ', t')-hard then an adversary has $\epsilon'' \approx \epsilon'$ advantage in distinguishing between Game (4, i, d) and Game (4, i, d + 1).*

Proof Given $(g_2^\alpha, g_2^\beta, g_2^\gamma) \in \mathbb{G}_2^3$, we construct a solver which uses a distinguisher to distinguish whether $\gamma = \alpha \cdot \beta$ or γ is random.

In Game $(4, n, d)$ we compute the pseudonym as $nym \leftarrow (g_2^\alpha)^r$, where r is obtained from the H_0 hash oracle. Then in Game $(4, n, d+1)$ we will program the H_0 hash oracle to return g_2^β for input $\mathtt{dom}_d$, and we set $nym \leftarrow g_2^\gamma$.

Now, note that if $\gamma = \alpha \cdot \beta$, then we have $nym \leftarrow g_2^{\alpha \cdot \beta}$, and the distinguisher will output that he is in game Game $(4, i, d)$ with advantage ϵ'. Otherwise, we may write $nym \leftarrow g_2^{\alpha' \cdot \beta}$ for a random α' and the distinguisher will output that he is in Game $(4, i, d+1)$. □

As noted before, we iterate for all users, which is at most n, in all domain, which is at most q_H. Hence, the advantage of distinguishing between Game $(5, q_U, q_\mathsf{H})$ and Game $(4, n, q_\mathsf{H})$ is $\epsilon \cdot (n \cdot q_\mathsf{H})$.

Finally, we end up with a system where secret key of all users are chosen independently at random with regard to each domain string, and each secret key of all devices are chosen independently at random with regard to each domain string, which is exactly the case of the ideal simulator.

6 The Σ-protocol

Below we describe the Σ-protocol on which our scheme from Sect. 4 is based. The Σ-protocol is a three move zero-knowledge proof of knowledge between a Prover and a Verifier.

Common Input: The common input to the Prover and Verifier CV consists of groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order p, group generators $g_1 \in \mathbb{G}_1$ and $\hat{g_2} \in \mathbb{G}_2$, and a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Furthermore, there is an element $Z \in \mathbb{G}_2$.
Private Input: The private input of the Prover consists of a pair $(A, u) \in \mathbb{G}_1 \times \mathbb{Z}_p$, such that $e(A, \hat{g_2}^u \cdot Z) = e(g_1, \hat{g_2})$.
The protocol:

1. (Prover) The Prover chooses $(r_1, r_2) \xleftarrow{\mathcal{R}} \mathbb{Z}_p^2$ at random and computes $R_1 \leftarrow A^{r_1}$, $R_2 \leftarrow g_1^{r_2}$ and $R_3 \leftarrow e(R_2, \hat{g_2})^u$.
2. (Prover → Verifier) The Prover and Verifier execute the following proof of knowledge:

$$PoK\left\{ (\alpha, \beta, \gamma) : R_1 = g_1^{\beta/(z+\alpha)} \wedge R_2 = g_1^\gamma \wedge R_3 = e\left(g_1, \hat{g_2}\right)^{\alpha \cdot \gamma} \right\}$$

   (a) The Prover chooses $(t_1, t_2, t_3) \xleftarrow{\mathcal{R}} \mathbb{Z}_p^3$ and computes

$$T_1 \leftarrow e\left(A, \hat{g_2}\right)^{-t_1 \cdot r_1} \cdot e\left(g_1, \hat{g_2}\right)^{t_2},$$
$$T_2 \leftarrow g_1^{t_3} \text{ and}$$
$$T_3 \leftarrow e\left(R_2, \hat{g_2}\right)^{t_1}.$$

   Then the Prover sends R_1, R_2, R_3, T_1, T_2 and T_3 to the Verifier.
   (b) (Verifier → Prover) The Verifier chooses $c \xleftarrow{\mathcal{R}} \mathbb{Z}_p$ and sends c to the Prover.
   (c) (Prover → Verifier) The Prover now computes $s_1 \leftarrow t_1 + c \cdot u, s_2 \leftarrow t_2 + c \cdot r_1$ and $s_3 \leftarrow t_3 + c \cdot r_2$.
3. (Verifier) The Verifier accepts if the following equations hold:

$$T_1 = e(R_1, Z)^{-c} \cdot e\left(R_1, \hat{g_2}\right)^{-s_1} \cdot e\left(g_1, \hat{g_2}\right)^{s_2}$$
$$T_2 = g_1^{s_3} \cdot R_2^{-c}$$
$$T_3 = e\left(R_2, \hat{g_2}\right)^{s_1} \cdot R_3^{-c}$$

Theorem 6 *The protocol described above is a Public-Coin Honest Verifier Zero-Knowledge Proof of a pair $(A, u) \in \mathbb{G}_1 \times \mathbb{Z}_p$, such that $e(A, \hat{g_2}^u \cdot Z) = e(g_1, \hat{g_2})$.*

The proof follows from the lemmas below.

Lemma 1 *The protocol is complete.*

Proof Suppose that a prover holds a pair $(u, A) \in \mathbb{Z}_p \times \mathbb{G}_1$ where $A = g_1^{1/(z+u)}$ and follows the protocol. In this case

$$\begin{aligned} & e(R_1, Z)^{-c} \cdot e\left(R_1, \hat{g_2}\right)^{-s_1} \cdot e\left(g_1, \hat{g_2}\right)^{s_2} \\ & = \left(e\left(R_1, \hat{g_2}\right)^{-t_1} \cdot e\left(g_1, \hat{g_2}\right)^{t_2}\right) \\ & \quad \cdot \left(e(R_1, Z)^{-c} \cdot e\left(R_1, \hat{g_2}^{-c \cdot u}\right)\right) \cdot e\left(g_1^{c \cdot r_1}, \hat{g_2}\right) \\ & = T_1 \cdot e\left(A^{r_1}, \hat{g_2}^{u} \cdot Z\right)^{-c} \cdot e\left(g_1^{r_1}, \hat{g_2}\right)^{c} \\ & = T_1 \cdot e\left(g_1, \hat{g_2}\right)^{-r_1 \cdot c} \cdot e\left(g_1, \hat{g_2}\right)^{r_1 \cdot c} = T_1. \end{aligned}$$

Then we have

$$g_1^{s_3} \cdot R_2^{-c} = g_1^{t_3} \cdot g_1^{c \cdot r_2} \cdot g_1^{-c \cdot r_2} = T_2$$

and

$$\begin{aligned} & e\left(R_2, \hat{g_2}\right)^{s_1} \cdot R_3^{-c} \\ & \quad = e\left(g_1^{r_2}, \hat{g_2}\right)^{t_1} \cdot e\left(g_1^{r_2}, \hat{g_2}\right)^{c \cdot u} \cdot e\left(g_1^{r_2}, \hat{g_2}\right)^{-c \cdot u} = T_3 \end{aligned}$$

□

Lemma 2 *The protocol has an extractor.*

Proof Suppose we can rewind the Prover to the moment when he is given the challenge c. The Prover will send R_1, R_2, R_3, T_1, T_2 and T_3 and respond with the challenge c and s_1, s_2 and s_3. Then we rewind to the step when the Prover obtains c and send a different challenge $c' \neq c$. The Prover will answer with s_1', s_2' and s_3' satisfying the verification equations. Let $\Delta s_i = (s_i - s_i')$ for $i = 1, 2, 3$ and $\Delta c = (c - c')$. From the equality

$$\begin{aligned} T_1 &= e(R_1, nym)^{-c} \cdot e\left(R_1, \hat{g_2}\right)^{-s_1} \cdot e\left(g_1, \hat{g_2}\right)^{s_2} \\ &= e(R_1, nym)^{-c'} \cdot e\left(R_1, \hat{g_2}\right)^{-s_1'} \cdot e\left(g_1, \hat{g_2}\right)^{s_2'} \end{aligned}$$

we have that

$$\begin{aligned} & e\left(R_1, \hat{g_2}\right)^{-\Delta s_1} \cdot e\left(g_1, \hat{g_2}\right)^{\Delta s_2} = e(R_1, nym)^{\Delta c} \\ & e\left(R_1, \hat{g_2}\right)^{-\Delta s_1/\Delta c} \cdot e\left(g_1, \hat{g_2}\right)^{\Delta s_2/\Delta c} = e(R_1, nym) \\ & e\left(g_1, \hat{g_2}\right)^{\Delta s_2/\Delta c} = e\left(R, \hat{g_2}^{\Delta s_1/\Delta c} \cdot nym\right) \\ & e\left(g_1, \hat{g_2}\right) = e\left(R_1^{(\Delta s_2/\Delta c)^{-1}}, \hat{g_2}^{\Delta s_1/\Delta c} \cdot nym\right) \end{aligned}$$

Then from $T_2 = g_1^{s_3} \cdot R_2^{-c} = g_1^{s_3'} \cdot R_2^{-c'}$ we have

$$\begin{aligned} g_1^{\Delta s_3} &= R_2^{\Delta c} \\ g_1^{\Delta s_3/\Delta c} &= R_2 \end{aligned}$$

So, we may compute $\tilde{u} = \Delta s_1/\Delta c$, and $\tilde{r_1} = \Delta s_2/\Delta c$, $\tilde{r_2} = \Delta s_3/\Delta c$ and $\tilde{A} = R^{\tilde{r_1}^{-1}}$ such that $g_1^{\tilde{r_2}} = R_2$ and $e(g_1, \hat{g_2}) = e(\tilde{A}, \hat{g_2}^{\tilde{u}} \cdot nym)$.

Finally, from $T_3 = e(R_2, \hat{g_2})^{s_1} \cdot R_3^{-c} = e(R_2, \hat{g_2})^{s_1'} \cdot R_3^{-c'}$ we have

$$\begin{aligned} & e\left(R_2, \hat{g_2}\right)^{\Delta s_1} = e\left(R_3, \hat{g_2}\right)^{\Delta c} \\ & e\left(g_1, \hat{g_2}\right)^{\tilde{r_2} \cdot \tilde{u}} = R_3. \end{aligned}$$

□

Lemma 3 *The protocol is Zero-Knowledge.*

Proof Given the common input $(\mathbb{G}_1, \mathbb{G}_2, e, g_1, \hat{g_2}, nym)$, where $\hat{g_2} = \mathsf{H}_0(\mathtt{dom})$ and $nym = \hat{g_2}^z$ for some $z \in \mathbb{Z}_p$, the simulator works as follows. Choose $R_3 \overset{\mathcal{R}}{\leftarrow} \mathbb{G}_1$ and $R_2 \overset{\mathcal{R}}{\leftarrow} \mathbb{G}_1$. Note that now the value of A_i is fixed by the choice of R_3 and R_2. In order to highlight this we may denote $R_2 = g_1^{r_2}$ and $R_3 = e(R_2, \hat{g_2})^{u_i}$ for some $u_i \in \mathbb{Z}_p$, hence we have $A = g_1^{1/(u_i+z)}$. Choose $R_1 \overset{\mathcal{R}}{\leftarrow} \mathbb{G}_1$ at random. See that $R_1 = A^{r_1/(u+z)}$ for some r_1, thus the values R_1, R_2, R_3 are distributed as in a real protocol. Now, the simulator chooses $(c, s_1, s_2, s_3) \overset{\mathcal{R}}{\leftarrow} \mathbb{Z}_p^4$ and computes

$$\begin{aligned} T_1 \leftarrow e(R_1, nym)^{-c} \cdot e\left(R_1, \hat{g_2}\right)^{-s_1} \cdot e\left(g_1, \hat{g_2}\right)^{s_2}, \\ T_2 \leftarrow g_1^{s_3} \cdot R_2^{-c} \text{ and } T_3 \leftarrow e\left(R_2, \hat{g_2}\right)^{s_1} \cdot R_3^{-c}. \end{aligned}$$

Obviously, T_1, T_2 and T_3 along with the values c, s_1, s_2, s_3 satisfy the verification equations. Moreover, R_1, R_2, R_3, c, s_1, s_2, s_3 are uniformly distributed as in the real executions, so the simulation is perfect. □

7 Conclusions

Beyond the concrete application case of delegating the rights by a user to multiple own devices, we have introduced a novel notion for group signature schemes. It expands the functionality of group signatures by adding the feature that group public keys may be pseudonyms derived ad hoc.

We have introduced a security framework for our scheme supporting strong privacy protection on one hand, and revocation capabilities on the other hand.

Finally, we have designed a scheme based on bilinear groups which implements such a system. Even if it uses bilinear groups and pairings, it is relatively simple and implementable. As our tests and efficiency comparison have shown, our solution is comparable with the most efficient schemes which offer only a limited functionality. Note that the user's root of trust may be a relatively weak device, since no procedure executed by it requires computation of pairings. They are needed for signature creation (this can be done by smart phones) and verification (on strong servers).

Acknowledgements This research is funded by National Science Centre, Poland, grant PRELUDIUM 8 under registration number 2014/15/N/ST6/04655 and Polish-Chinese cooperation venture of Xidian University and Wrocaw University of Science and Technology on Secure Data Outsourcing in Cloud Computing.

References

1. Ali, S., Amberker, B.: Dynamic attribute based group signature with attribute anonymity and tracing in the standard model. In: Gierlichs, B., Guilley, S., Mukhopadhyay, D. (eds.) Security, Privacy, and Applied Cryptography Engineering. Lecture Notes in Computer Science, vol. 8204, pp. 147–171. Springer, Berlin (2013)
2. Barbulescu, R., Duquesne, S.: Updating key size estimations for pairings. Cryptology ePrint Archive, Report 2017/334. http://eprint.iacr.org/2017/334 (2017). Accessed 14 June 2017
3. Bellare, M., Micciancio, D., Warinschi, B.: Foundations of group signatures: formal definitions, simplified requirements, and a construction based on general assumptions. In: Biham, E. (ed.) EUROCRYPT'03, Lecture Notes in Computer Science, vol. 2656, pp. 614–629. Springer, Berlin (2003)
4. Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS '06, pp. 390–399. ACM, New York (2006)
5. Bellare, M., Shi, H., Zhang, C.: Foundations of group signatures: the case of dynamic groups. In: Menezes, A. (ed.) CT-RSA'05, Lecture Notes in Computer Science, vol. 3376, pp. 136–153. Springer, Berlin (2005)
6. Bichsel, P., Camenisch, J., Neven, G., Smart, N., Warinschi, B.: Get shorty via group signatures without encryption. In: Garay, J., De Prisco, R. (eds.) Security and Cryptography for Networks, Lecture Notes in Computer Science, vol. 6280, pp. 381–398. Springer, Berlin (2010)
7. Boneh, D., Boyen, X.: Short signatures without random oracles and the SDH assumption in bilinear groups. J. Cryptol. **21**(2), 149–177 (2008)
8. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: Franklin, M. (ed.) CRYPTO'04, Lecture Notes in Computer Science, vol. 3027, pp. 41–55. Springer, Berlin (2004)
9. Boneh, D., Shacham, H.: Group signatures with verifier-local revocation. In: Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS '04, pp. 168–177. ACM (2004)
10. Bonneau, J., Herley, C., Van Oorschot, P.C., Stajano, F.: The quest to replace passwords: a framework for comparative evaluation of web authentication schemes. In: 2012 IEEE Symposium on Security and Privacy, pp. 553–567 (2012)
11. Boyen, X., Waters, B.: Full-domain subgroup hiding and constant-size group signatures. In: Okamoto, T., Wang, X. (eds.) PKC'07, Lecture Notes in Computer Science, vol. 4450, pp. 1–15. Springer, Berlin (2007)
12. Brickell, E., Camenisch, J., Chen, L.: Direct anonymous attestation. In: Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS '04, pp. 132–145. ACM (2004)
13. Bringer, J., Chabanne, H., Lescuyer, R., Patey, A.: Efficient and strongly secure dynamic domain-specific pseudonymous signatures for ID documents. In: Christin, N., Safavi-Naini, R. (eds.) Financial Cryptography and Data Security, Lecture Notes in Computer Science, vol. 8437, pp. 255–272. Springer, Berlin (2014)
14. Camenisch, J., Mdersheim, S., Sommer, D.: A formal model of identity mixer. In: Kowalewski, S., Roveri, M. (eds.) Formal Methods for Industrial Critical Systems, Lecture Notes in Computer Science, vol. 6371, pp. 198–214. Springer, Berlin (2010)
15. Caro, A.D., Iovino, V.: JPBC: java pairing based cryptography. In: 2011 IEEE Symposium on Computers and Communications (ISCC), pp. 850–855 (2011)
16. Damgård, I.: On Σ-protocols. Lecture notes for CPT, v.2. http://www.cs.au.dk/~ivan/Sigma.pdf (2010). Accessed 5 Feb 2017
17. Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for cryptographers. Discrete Appl. Math. **156**(16), 3113–3121 (2008)
18. Guillevic, A., Vergnaud, D.: Algorithms for outsourcing pairing computation. In: Joye, M., Moradi, A. (eds.) Smart Card Research and Advanced Applications, CARDIS'14, pp. 193–211. Springer, Cham (2015)
19. Han, S., Wang, J., Liu, W.: An efficient identity-based group signature scheme over elliptic curves. In: Freire, M., Chemouil, P., Lorenz, P., Gravey, A. (eds.) Universal Multiservice Networks, Lecture Notes in Computer Science, vol. 3262, pp. 417–429. Springer, Berlin (2004)
20. ISO/EIC: 15946-5:2009, cryptographic techniques based on elliptic curves—part 5: elliptic curve generation. https://www.iso.org/standard/46541.html (2009). Accessed 11 Aug 2017
21. Kato, A., Scott, M., Kobayashi, T., Kawahara, Y.: Barreto–Naehrig curves. Internet-Draft draft-kasamatsu-bncurves-02, IETF Secretariat. https://tools.ietf.org/html/draft-kasamatsu-bncurves-02 (2016). Accessed 17 Apr 2018
22. Kiayias, A., Tsiounis, Y., Yung, M.: Traceable signatures. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT'04, Lecture Notes in Computer Science, vol. 3027, pp. 571–589. Springer, Berlin (2004)
23. Kluczniak, K., Wang, J., Chen, X., Kutyłowski, M.: Multi-device anonymous authentication. In: Chen, J., Piuri, V., Su, C., Yung, M. (eds.) NSS'16, pp. 21–36. Springer, Cham (2016)
24. Neuman, C., Yu, T., Hartman, S., Raeburn, K.: The Kerberos Network Authentication Service (V5). RFC Editor. http://www.rfc-editor.org/rfc/rfc4120.txt (2005). Accessed 17 Apr 2018
25. Lynn, B.: On the implementation of pairing-based cryptosystems. https://crypto.stanford.edu/pbc/thesis.pdf (2007). Accessed 7 Mar 2017
26. Nakanishi, T., Funabiki, N.: Verifier-local revocation group signature schemes with backward unlinkability from bilinear maps. In: ASIACRYPT'05, pp. 533–548. Springer, Berlin (2005)
27. Needham, R.M., Schroeder, M.D.: Using encryption for authentication in large networks of computers. Commun. ACM **21**(12), 993–999 (1978)
28. O'Gorman, L.: Comparing passwords, tokens, and biometrics for user authentication. Proc. IEEE **91**(12), 2021–2040 (2003)
29. Parno, B., Kuo, C., Perrig, A.: Phoolproof phishing prevention. In: Di Crescenzo, G., Rubin, A. (eds.) Financial Cryptography and Data Security, pp. 1–19. Springer, Berlin (2006)
30. PBC: Pairing based cryptography library. https://crypto.stanford.edu/pbc/. Accessed 26 Aug 2017
31. Recordon, D., Reed, D.: Openid 2.0: a platform for user-centric identity management. In: Proceedings of the Second ACM Workshop on Digital Identity Management, DIM '06, pp. 11–16. ACM (2006)
32. Renaud, K.: Quantifying the quality of web authentication mechanisms: a usability perspective. J. Web Eng. **3**(2), 95–123 (2004)
33. Sun, S.T., Boshmaf, Y., Hawkey, K., Beznosov, K.: A billion keys, but few locks: the crisis of web single sign-on. In: Proceedings

of the 2010 New Security Paradigms Workshop, NSPW '10, pp. 61–72. ACM (2010)
34. Trolin, M., Wikström, D.: Hierarchical group signatures. In: Caires, L., Italiano, G., Monteiro, L., Palamidessi, C., Yung,M. (eds.) Automata, Languages and Programming, Lecture Notes in Computer Science, vol. 3580, pp. 446–458. Springer, Berlin (2005)