EPID with Efficient Proof of Non-Revocation

Olivier Sanders

EPID with Efficient Proof of Non-Revocation

Olivier Sanders
olivier.sanders@orange.com
Orange Labs
Cesson-Sévigné, France

ABSTRACT

EPID systems are anonymous authentication mechanisms which are standardized by ISO/IEC and massively deployed in Intel processors. They are related to the large family of privacy-preserving signatures but differ in that they provide a very pragmatic way of revoking members. Concretely, a member $\mathcal{P}$ can be revoked by simply placing one of its signatures in a so-called signature revocation list SRL. Once this is done, every signer will have to include in its future signatures a proof that it has not generated any element of SRL, which implicitly revokes $\mathcal{P}$.

This proof of non-revocation generated by each signer is thus the core component of EPID systems and largely dominates the overall complexity. Yet, it appears that it has been a secondary concern for existing constructions that usually implement it using some costly modular zero-knowledge proofs.

In this paper, we reconsider this problem by proposing a new EPID system with a much more efficient proof of non-revocation. The latter is no longer zero-knowledge but its combination with the other components of the EPID system still results in an anonymous signature. Proving the latter point is actually quite complex and requires to tweak these other components but in the end it leads to EPID signatures that are up to three times smaller than previous ones and that can be generated and verified with two times less computations.

CCS CONCEPTS

• **Security and privacy** → *Public key (asymmetric) techniques.*

KEYWORDS

applied cryptography, anonymous authentication, EPID

ACM Reference Format:
Olivier Sanders. 2022. EPID with Efficient Proof of Non-Revocation. In *Proceedings of the 2022 ACM Asia Conference on Computer and Communications Security (ASIA CCS '22), May 30-June 3, 2022, Nagasaki, Japan.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3488932.3517392

1 INTRODUCTION

Related Works. Introduced by Brickell and Li in 2007 [7], Enhanced Privacy ID (EPID) is an anonymous authentication mechanism that quickly gained traction in the real-world. It is today embedded in billion of devices [17] and included in the ISO/IEC 20008-2 standard [18].

Technically speaking, it is a spin-off of group signature [12] and, as such, is a centralized system managed by a so-called *issuer* which can enrol *platforms*. Once a platform has been accepted by the issuer, it can generate EPID signatures that anyone can verify using the issuer's public key but that cannot be traced back to the platform. So far, this is exactly a group signature. The fundamental difference between these two primitives lies in the safeguards they implement to prevent bad behaviour.

In group signatures, an *opener* (that may be merged with the issuer) has the ability to lift anonymity of any signature using some secret knowledge. This is a very powerful feature, which places heavy responsibilities on this entity that must be trusted by all the actors involved in the system. Moreover, once the issuer of a given group signature has been identified, it is not very clear what the next step should be as revocation is usually not considered by group signature models (*e.g.* [2, 3]) and is quite hard to add in practice.

In an EPID system, there is no trusted entity able to lift anonymity. This feature is replaced by two revocation mechanisms that are quite different. The first one is a key revocation mechanism that allows to trace all EPID signatures generated with a given secret key. It is very powerful but assumes knowledge of the secret key of the platform to revoke, which is rather unlikely, unless in very specific scenarios where this secret key would leak. The second one is a signature revocation mechanism which constitutes the main specificity of EPID. Concretely, a signature revocation list SRL is generated from a given set of EPID signatures $\{\mu_i\}_{i=1}^n$ and can then be used as an input for future EPID signatures. When generating the latter, the platforms have to additionally provide a proof that they have *not* generated any of the signatures μ_i. This excludes all the platforms that have generated at least one of the μ_i which are *de facto* revoked. This calls for several comments.

Firstly, signature revocation is an inherently decentralized system in the sense that every entity can *technically*[1] generate a valid signature revocation list SRL from a set of signatures. In other words, the generation of SRL is a public process that does not require knowledge of a secret value. Secondly, the status (active or revoked) of each platform is linked to a given SRL. Concretely, a platform implicitly revoked by SRL may still be able to generate EPID signatures for a different list SRL′. Finally, signature revocation cannot be done *a posteriori*: once an EPID signature has been generated for some list SRL it is not possible to check if it is still valid for another list SRL′. This is one of the main differences with the related notion of verifier-local revocation [5] for group signatures.

ASIA CCS '22, May 30-June 3, 2022, Nagasaki, Japan

ACM ISBN 978-1-4503-9140-5/22/05…$15.00
https://doi.org/10.1145/3488932.3517392

[1] We emphasize this word as some models require SRL to be generated by a trusted entity but do not implement countermeasures to prevent another entity from generating a valid SRL (see discussion in [22]).

In practice, all EPID systems use the same framework, yielding signature that can be split into two parts. The first part is inherited from group signature and consists in proving knowledge of a certificate generated by the issuer on the platform secret key s. The second part is dedicated to the proof of non-revocation and essentially consists in a proof that s was not used to generate any element of the signature revocation list SRL. In all existing systems (*e.g.* [7, 8, 19, 22]) this latter part is by far the largest as its size grows linearly with the number n of revoked signatures. For example, as illustrated in Section 5, in both [8] and [22] the first part amounts to roughly 2000 bits whereas the latter represents $894n$ bits, which quickly dominates the overall size.

Yet, we note that the literature on EPID has always been focused on the first part, trying to leverage advances in the group signature area. In particular, the proof of non-revocation (the second part) clearly appears as a secondary concern that is usually addressed by using (or just mentioning) some modular zero-knowledge proof (e.g., the one from [11]) without trying to optimize it for the EPID context.

Our Contributions. In this paper, we depart from previous works by focusing on the second part of EPID systems, namely the proof of non-revocation, and explain how we can significantly improve it by using a very different strategy.

But first, let us recall some elements on existing constructions in cyclic groups. As all EPID signatures can potentially be used in a revocation list, any signature μ generated with a secret s contains some pair $(h, h^s) \in \mathbb{G}_1^2$ whose only purpose is to facilitate future proofs of non-revocation. This approach, which is reminiscent of Direct Anonymous Attestation [10], does not harm privacy as the platform uses a new h for each signature. Thanks to this random-looking pair, any platform can prove that it has not generated μ by essentially proving that its own secret key s^* would yield a different element h^{s^*}.

However, this is easier said than done as the most straightforward solutions (such as providing h^{s^*} in clear) would clearly break anonymity (the same element h^{s^*} will appear in all signatures generated by this platform for signature revocation lists containing the pair (h, h^s)). This is where zero-knowledge proof proves convenient. By using a modular zero-knowledge proof that s^* is different from all the secret keys used to generate the pairs $(h_i, h_i^{s_i})$ contained in revoked signatures, one ensures that anonymity is preserved (thanks to the zero-knowledge property) and that revocation is enforced (thanks to the soundness property) without having to write a new security proof. Typically, in the anonymity analysis, one simply runs the zero-knowledge simulator and then has no longer to care for this part of the signature. Unfortunately, this modularity has a cost. Even the efficient zero-knowledge proof system from [11] requires to send 1 element of $\mathbb{G}_1$ and two scalars per revoked signature. This also affects the computational cost of both the signature and the verification algorithms which are augmented by 6 group exponentiations per revoked signature.

Our goal in this paper is to propose a middle-way which is cheaper than a full-fledged zero-knowledge proof but which still retains anonymity of the construction. Let us consider a platform $\mathcal{P}$ owning some secret scalar s^* which needs to prove that it has not generated any of the pairs $(h_i, h_i^{s_i})$ contained in some revocation list SRL. Logically, for each i, this proof will be a function of s^*, h_i and $h_i^{s_i}$. However, it must also depend on other values. Otherwise, if $\mathcal{P}$ produced a new signature for a list SRL$'$ that would contain one these pairs, then it would generate the same proof and could be traced. More generally, this proof of non-revocation must look random to prove anonymity of the resulting construction.

In zero-knowledge proofs used by previous works this problem is dealt with by adding random values that essentially re-randomizes all the group elements. Unfortunately, this requires to prove knowledge of these values so as to ensure soundness, which increases the proof size. Moreover, one cannot use the same random values for two different pairs $(h_i, h_i^{s_i})$ and $(h_j, h_j^{s_j})$ contained in SRL as the adversary may have generated the latter and thus know some relations between these elements, which it could use to distinguish valid proofs from simulated ones. Therefore, fresh random values must be used for each pair, hence the linear number of scalars included in all existing EPID systems.

In our construction, we then need to avoid the use of random scalars but still need to generate random looking proofs of non-revocation. In this regard, pseudo-random function seems to be the right tool to use but we need to evaluate it on the inputs mentioned above (in particular $(h_i, h_i^{s_i})$), which is not usual. Fortunately, we note that the Dodis-Yampolskiy PRF [13] is very well-suited to our case. Indeed, for any scalar a_i, the element $C_i = (h_i^{a_i} \cdot h_i^{s_i})^{\frac{1}{s^*+a_i}}$ is h_i if, and only if, $s_i = s^*$, which allows to quickly detect revoked platform. Moreover, in the case where $s_i \neq s^*$, C_i looks random even when a_i is known as it is essentially the output of a PRF function. The scalars a_i are not even required to be random, which allows us to generate them deterministically using a hash function. Our proof of non-revocation thus contains only one element (C_i) per revoked signature which is roughly a three-fold decrease in size compared to the state-of-the-art. This also affects the computational cost of the signature and the verification algorithms which is roughly halved.

Surprisingly, this is not the end of the story. The proof of non-revocation we obtain this way is indeed correct and looks random but formally proving this fact is quite difficult and requires some additional modifications of the signature. This is where we pay the price of using a proof that is not zero-knowledge. As there is no simulator that we can run to generate the proof of non-revocation in the anonymity analysis, we need to construct each of these elements C_i *without knowing* s^*.

Here again, someone familiar with the result in [13] might underestimate the problem. Dodis and Yampolskyi indeed showed in their paper how to generate $g^{\frac{1}{s^*+a_i}}$ without knowing s^* but for a *fixed* basis g, generated by the reduction. Here, we need to evaluate this function over random basis $(h_i, h_i^{s_i})$ that can be chosen by the adversary. Of course, we can (and do) leverage some common techniques such as setting h_i as a hash output that will be programmed in the security proof but we have no way of controlling the secret s_i.

We deal with this issue in two complementary ways. Firstly, we enable the reduction to extract the value s_i from the signature itself by relying on the techniques introduced by Fischlin [15]. This adds some constant overhead to the signature but it is fortunately quickly amortized by what we gain with our new proof of non-revocation. We provide more technical details in Section 4. Secondly, we need

to revisit (slightly) the requirements placed on signature revocation lists by the model. Sanders and Traoré [22] recently pointed out that this is the weak point of previous EPID models and then proposed a new model where the adversary has full control of the signature revocation list SRL that can even contain fully random elements. This was a significant step forward compared to previous models that require SRL to be generated by a trusted entity which decided which signature can be added. However we cannot prove security in the strong model of [22]. Instead, we rely on an intermediate model, stronger than the one of all EPID systems before [22], where the adversary is free to decide which signatures must be added to SRL as long as the signatures are valid, which can be publicly verified. We discuss this point in Section 3 but, intuitively, we need to deal with valid signatures so as to be sure that our reduction will be able to extract all the elements necessary to compute C_i.

Thanks to these slight modifications, we manage to prove security of our scheme without significantly changing the performance. We indeed show in Section 5 that our construction overperforms the most efficient ones from the state-of-the-art as soon as the number of revoked signatures is larger than 5, a threshold that should easily be met in practice.

2 PRELIMINARIES

Bilinear groups. Our construction requires bilinear groups which are constituted of a set of three groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ of order p along with a map, called pairing, $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ that is

(1) bilinear: for any $g \in \mathbb{G}_1, \widetilde{g} \in \mathbb{G}_2$, and $a, b \in \mathbb{Z}_p$, $e(g^a, \widetilde{g}^b) = e(g, \widetilde{g})^{ab}$;
(2) non-degenerate: for any $g \in \mathbb{G}_1^*$ and $\widetilde{g} \in \mathbb{G}_2^*$, $e(g, \widetilde{g}) \neq 1_{\mathbb{G}_T}$;
(3) efficient: for any $g \in \mathbb{G}_1$ and $\widetilde{g} \in \mathbb{G}_2$, $e(g, \widetilde{g})$ can be efficiently computed.

As most recent cryptographic papers, we only consider bilinear groups of prime order with *type 3* pairings [16], meaning that no efficiently computable homomorphism is known between $\mathbb{G}_1$ and $\mathbb{G}_2$, in either direction.

Computational assumptions. The security analysis of our protocol will make use of the following assumptions.

- q-DL *assumption: Given* $(g, g^x, \ldots, g^{x^q}) \in \mathbb{G}_1^{q+1}$ *and* $(\widetilde{g}, \widetilde{g}^x) \in \mathbb{G}_1^2$*, it is hard to recover* x.
  It has been used in several papers (*e.g.* [1, 20]) and is implied by many other assumptions such as, for example, the q-SDH one [4]. In the case $q = 1$, we simply say the DL assumption.
- q-DHI *assumption: Given* $(g, g^x, \ldots, g^{x^q}) \in \mathbb{G}_1^{q+1}$ *in a type 3 bilinear group, it is hard to distinguish* $g^{\frac{1}{x}}$ *from a random element in* $\mathbb{G}_1$.
  This assumption is adapted from the q-DBDHI in [13] that states the hardness of distinguishing $e(g, g)^{\frac{1}{x}}$ from a random element in $\mathbb{G}_T$, given the same input but in a *type 1* bilinear group. In the latter case, the element $g^{\frac{1}{x}} \in \mathbb{G}_1$ could indeed be trivially distinguished by testing whether $e(g^{\frac{1}{x}}, g^x) = e(g, g)$. Fortunately, this is no longer possible in type 3 bilinear group (because no element involving x is provided in $\mathbb{G}_2$) and we can then simplify this assumption by defining a challenge element in $\mathbb{G}_1$ instead of $\mathbb{G}_T$.
- DDH *assumption: Given* $(g, g^x, g^y, g^z) \in \mathbb{G}^4$*, it is hard to decide whether* $z = x \cdot y$ *or* z *is random.*

3 SPECIFICATION OF EPID

In a recent paper [22], Sanders and Traoré propose a new security model for EPID which fixes some issues of previous definitions but also allows the adversary to request signatures generated with malicious revocation lists, contrarily to all previous models which assume that revocation lists are generated by a trusted entity. Although this is a very desirable feature, our scheme cannot be proven secure in this strong model as our reduction can only handle well-formed revocation lists. To generalise the model from [22], we will then simply define a new oracle $O\mathtt{SigRevoke}$ that the adversary can query on any set of signatures to generate the corresponding signature revocation list. The only task of this oracle is to check validity of the revoked signatures, which makes our model stronger than previous ones, except the one from [22].

Indeed, in the first security models for EPID (*e.g.* [8]) the signature revocation list SRL had to be generated by a trusted entity (the *revocation manager*) to ensure that SRL only contains signatures generated by the challenger during experiments (see, *e.g.*, [8]). Concretely, this means a very centralised system where the revocation manager has full control on the revocation list. We do not need such an entity in our case and we allow the adversary to revoke *any* signature μ_i as long as μ_i is valid. In practice, this means that the signer should check validity of the revoked signatures or trust some entities to perform these verifications. This is thus an intermediate situation between initial EPID security models and the one from [22]. To sum up:

- In existing security models (except [22]), a trusted entity controls the revocation list. In particular, the adversary cannot place signatures it has generated on the latter.
- In [22], the adversary has full control of the revocation list that may even contain random elements.
- In our model, the adversary chooses the set of signatures that constitutes the signature revocation list as long as the signatures are valid. In particular, it can place its own signatures on this list.

3.1 Syntax

An EPID system is defined by the following algorithms that involve three types of entities: an issuer $\mathcal{I}$, platforms $\mathcal{P}$ and verifiers $\mathcal{V}$.

- $\mathtt{Setup}(1^k)$: on input a security parameter 1^k, this algorithm returns the public parameters pp of the system.
- $\mathtt{GKeygen}(pp)$: on input the public parameters pp, this algorithm generates the issuer's key pair $(\mathsf{isk}, \mathsf{ipk})$. We assume that ipk contains pp and so we remove pp from the inputs of all following algorithms.
- Join: this is an interactive protocol between a platform $\mathcal{P}$, taking as inputs ipk, and the issuer $\mathcal{I}$ owning isk. At the end of the protocol, the platform returns either $\perp$ or a signing key sk whereas the issuer does not return anything.
- $\mathtt{KeyRevoke}(\{\mathsf{sk}_i\}_{i=1}^m)$: this algorithm takes as input a set of m platform secret keys sk_i and returns a corresponding key revocation list KRL containing m elements.

- $\mathtt{SigRevoke}(\mathsf{ipk}, \{(\mu_i, \mathsf{m}_i)\}_{i=1}^n)$: this algorithm takes as input ipk and a set of n EPID (signature,message) pairs $\{(\mu_i, \mathsf{m}_i)\}_{i=1}^n$ and returns a corresponding signature revocation list SRL containing n elements.
- $\mathtt{Sign}(\mathsf{ipk}, \mathsf{sk}, \mathsf{m}, \mathsf{SRL})$: this algorithm takes as input the issuer's public key ipk, a platform secret key sk a message m and a signature revocation list SRL and returns an EPID signature μ.
- $\mathtt{Identify}(\mathsf{sk}, t)$: given a platform secret key and an element t from a revocation list SRL (*i.e.* there exists some i such that $t = \mathsf{SRL}[i]$), this algorithm returns either 1 (t was generated using sk) or 0.
- $\mathtt{Verify}(\mathsf{ipk}, \mathsf{KRL}, \mathsf{SRL}, \mu, \mathsf{m})$: given an issuer public key ipk, a key revocation list KRL, a signature revocation list SRL, a signature μ and a message m, this algorithms returns 1 (the signature is valid on m for the corresponding revocation lists) or 0.

3.2 Security Model

As all previous works, we expect an EPID system to be correct, unforgeable and anonymous. However, the formal definitions of the latter two properties differ according to the papers. In our paper, we will make use of the security properties defined in [22] that we slightly adapt, as discussed at the beginning of this section.

Correctness. An EPID system is *correct* if, for all signing key sk_i generated using Join, KRL generated using KeyRevoke and SRL generated using SigRevoke:

$$\mathtt{Verify}(\mathsf{ipk}, \mathsf{KRL}, \mathsf{SRL}, \mathtt{Sign}(\mathsf{ipk}, \mathsf{sk}_i, m, \mathsf{SRL}), m) = 1$$
$$\Leftrightarrow \mathsf{sk}_i \notin \mathsf{KRL} \wedge \forall j : \mathtt{Identify}(\mathsf{sk}_i, \mathsf{SRL}[j]) = 0$$

Honest Revocation List. The security properties from [22] do not place any restriction on the signature revocation lists SRL. The adversary is indeed free to construct them as it wants, for example by placing random group elements in SRL, hence the term of *malicious revocation lists* used in the original paper. As discussed above, this is not possible for our construction and we then define a new oracle that the adversary can query to revoke signatures of its choice.

- $O\mathtt{SigRevoke}(\{(\mu_i, \mathsf{m}_i)\}_{i=1}^n)$ is an oracle used by $\mathcal{A}$ to revoke a set of signatures $\{\mu_i\}$ on messages m_i. The oracle checks the validity of each signature by running the Verify algorithm and outputs $\perp$ if one of the signatures is not valid. Else, it runs $\mathtt{SigRevoke}(\mathsf{ipk}, \{(\mu_i, \mathsf{m}_i)\}_{i=1}^n)$ and outputs SRL.

Unforgeability. The unforgeability experiment is defined in Figure 1, where c (resp. d) is a counter indicating the number of corrupt users created by $\mathcal{A}$ (resp. of signatures issued by the adversary $\mathcal{A}$) at the current time and where $\mathcal{A}$ may query the following oracles:

- $O\mathtt{Add}(k)$ is an oracle that is used by the adversary to add a new honest platform k. A signing key sk_k is then generated for this platform using the Join protocol but nothing is returned to $\mathcal{A}$.
- $O\mathtt{Join}_c()$ is an oracle playing the issuer's side of the Join protocol. It is used by $\mathcal{A}$ to add a new corrupt platform. Each call to this oracle increases by 1 the current value of c (*i.e.* $c = c + 1$).
- $O\mathtt{Cor}(k)$ is an oracle that returns the signing key sk_k of an honest platform k and also adds it to a list $\mathcal{K}$ that is initially set as empty.
- $O\mathtt{Sign}(k, \mathsf{SRL}, m)$ is an oracle that is used by $\mathcal{A}$ to query a signature from the platform k on a message m with a signature revocation list SRL. If SRL has not been output by $O\mathtt{SigRevoke}$ then this oracle returns $\perp$. We define $\mathcal{S}$ as the set of all signatures returned by this oracle.

$\mathsf{Exp}_{\mathcal{A}}^{unf}(1^k)$ – Unforgeability Security Game
(1) $c, d \leftarrow 0$
(2) $\mathcal{S}_{\mathcal{A}} \leftarrow \emptyset$
(3) $pp \leftarrow \mathtt{Setup}(1^k)$
(4) $(\mathsf{isk}, \mathsf{ipk}) \leftarrow \mathtt{GKeygen}(pp)$
(5) while $d \leq c$:
- $\mathsf{SRL} \leftarrow \mathtt{SigRevoke}(\mathcal{S}_A)$
- $(\mu, m) \leftarrow \mathcal{A}^{O\mathtt{Add}, O\mathtt{Join}_c, O\mathtt{Cor}, O\mathtt{SigRevoke}, O\mathtt{Sign}}(\mathsf{SRL}, \mathsf{ipk})$
- $\mathsf{KRL} \leftarrow \mathtt{KeyRevoke}(\mathcal{K})$
- if $1 = \mathtt{Verify}(\mathsf{ipk}, \mathsf{KRL}, \mathsf{SRL}, \mu, m) \wedge \mu \notin \mathcal{S}_{\mathcal{A}} \cup \mathcal{S}$ then $d = d + 1 \wedge \mathcal{S}_{\mathcal{A}} \leftarrow \mathcal{S}_{\mathcal{A}} \cup \{\mu\}$

(6) Return 1

Figure 1: Unforgeability Game for EPID Signature

An EPID system is unforgeable if $\mathsf{Adv}^{unf}(\mathcal{A}) = \Pr[\mathsf{Exp}_{\mathcal{A}}^{unf}(1^k) = 1]$ is negligible for any $\mathcal{A}$. The core idea of this unforgeability notion is that revocation of a signature generated using some secret key sk must force the owner of sk to use a different secret key to produce a new signature. Therefore, an adversary owning $d - 1$ keys should not be able to produce d valid and distinct signatures $\{\mu_i\}_{i=1}^d$ if μ_j was generated for a revocation list SRL constructed from $\{\mu_i\}_{i=1}^{j-1}$, for all $j \in [1, d]$.

Anonymity. The anonymity property introduced in [22] is presented in Figure 2 and makes use of the following oracles.

- $O\mathtt{Join}_h()$ is an oracle playing the user's side of the Join protocol and is then used by $\mathcal{A}$, playing the issuer, to add a new honest platform. Each call generates a platform secret key sk that is kept secret by the challenger.
- $O\mathtt{Sign}^*(\mathsf{SRL}, m)$ is an oracle used by $\mathcal{A}$ to query a signature on m from an honest platform that is not implicitly revoked by SRL. If SRL has not been output by $O\mathtt{SigRevoke}$, then this oracle returns $\perp$. Else, the challenger of the experiment randomly selects a signing key sk among those that are not revoked by SRL (that is, the secret keys sk_i such that $\mathtt{Identify}(\mathsf{sk}_i, \mathsf{SRL}[j]) = 0$ for every element $\mathsf{SRL}[j]$ of SRL) and then returns the output of $\mathtt{Sign}(\mathsf{ipk}, \mathsf{sk}, m, \mathsf{SRL})$. We define $\mathcal{S}$ as the set of all signatures returned by this oracle concatenated with the key sk used to generate them.
- $O\mathtt{Cor}^*(\mu)$ is an oracle that returns the signing key sk_k used to generate the signature μ if there is some pair $(\mu||\mathsf{sk}_k)$ in $\mathcal{S}$. Else, it returns $\perp$. Once the adversary has returned the two challenge signatures μ_0 and μ_1 (step 3 of the anonymity game), we slightly modify the behaviour of this oracle to prevent unintentional failure of the adversary. Indeed, the adversary could inadvertently query $O\mathtt{Cor}^*$ on a signature μ

generated using the same key as μ_0 or μ_1, making it lose the game at step 8. After step 3, this oracle therefore returns $\perp$ if queried on a signature μ generated using sk_i, for $i \in \{0,1\}$. We note that we nevertheless still need the success condition of step 8 as the adversary could have queried $O\mathsf{Cor}^*$ on such a signature μ before outputting μ_0 and μ_1. However, in such a case, the adversary knows that μ_0 and μ_1 are illicit choices before returning them and its failure will then no longer be unintended.

$\mathsf{Exp}^{an-b}_{\mathcal{A}}(1^k)$ – Anonymity Security Game
(1) $pp \leftarrow \mathsf{Setup}(1^k)$
(2) $(\mathsf{isk}, \mathsf{ipk}) \leftarrow \mathsf{GKeygen}(pp)$
(3) $(\mu_0, \mu_1, m, \mathsf{SRL}) \leftarrow \mathcal{A}^{O\mathsf{Join}_h, O\mathsf{Cor}^*, O\mathsf{Sign}^*}(\mathsf{isk})$
(4) if SRL is not an output of $O\mathsf{SigRevoke}$, then return 0
(5) if no entry (μ_i, sk_i) in $\mathcal{S}$ for $i \in \{0,1\}$, then return 0
(6) if $\exists i, j \in \{0,1\}$: $\mathsf{Identify}(\mathsf{sk}_i, \mathsf{SRL}[j]) = 1$, then return 0
(7) $\mu^* \leftarrow \mathsf{Sign}(\mathsf{ipk}, \mathsf{sk}_b, m, \mathsf{SRL})$
(8) $b' \leftarrow \mathcal{A}^{O\mathsf{Join}_h, O\mathsf{Cor}^*, O\mathsf{Sign}^*}(\mathsf{isk}, \mu^*)$
(9) if $\exists i \in \{0,1\}$: sk_i leaked by $O\mathsf{Cor}^*$, then return 0
(10) Return $(b = b')$

Figure 2: Anonymity Game for EPID Signature

An EPID system is anonymous if $\mathsf{Adv}^{an}(\mathcal{A}) = |\Pr[\mathsf{Exp}^{an-1}_{\mathcal{A}}(1^k) = 1] - \Pr[\mathsf{Exp}^{an-0}_{\mathcal{A}}(1^k) = 1]|$ is negligible for any $\mathcal{A}$.

4 OUR CONSTRUCTION

4.1 Intuition

As explained in Section 1, EPID signatures of all previous papers (*e.g.* [7, 8, 22]) can be divided into two parts.

In the first part, the signer anonymously proves that it is a legitimate platform that has been enrolled by the issuer $\mathcal{I}$. In practice, this is done by proving possession of a certificate issued by $\mathcal{I}$ on the platform secret key, which roughly corresponds to a group signature. In this paper, we instantiate this part using PS signatures [21] that have proved very suitable for this kind of proofs. This is a very common approach and we do not claim any novelty here.

In the second part, the platform proves that it is not implicitly revoked by the signature revocation list SRL. To make this proof easier, each signature generated with a secret key s contains a pair $(h, h^s) \in \mathbb{G}_1^2$. This way, a platform with a key s^* can prove that it is not revoked by essentially showing that raising h to the power s^* would not yield h^s. In current systems, this is done by using zero-knowledge proofs that require to send several elements per revoked signatures, as explained in Section 1.

The novelty of our EPID scheme is the way it implements this second part. It makes use of a new proof of non-revocation which is rather generic and could actually be used in previous schemes provided that (1) h is generated as a hash output of some string str, that is, $h = H(str)$ for some hash function H and (2) the revocation lists SRL are well-formed, that is, they only contain pairs (h, h^s) originating from valid signatures. The first requirement (met by our scheme) is quite classical as it allows to reduce the size of the EPID signature. The second requirement was actually implicit for all previous constructions until the very recent work by Sanders and Traoré [22].

The difference with all previous works is that we no longer prove non-revocation by relying on a modular zero-knowledge proof. Instead, we use a proof of non-revocation that is not zero-knowledge but whose combination with the other part of the EPID signature still yields an anonymous system under a reasonable computational assumption. The first rationale behind our construction is that we want to avoid the re-randomization approach common to previous strategies where the elements h^{s_i} from SRL are raised to some random powers r_i so as to be compared to $(h^{r_i})^{s^*}$. This approach indeed allows to easily retain anonymity but requires to provide these re-randomized elements along with a proof of well-formedness, which is rather costly as the random scalars r_i must obviously remain secret.

To reduce the proof size, we need to use much less secret elements while still generating random-looking proofs of non-revocation. To this end, we note that the Dodis-Yampolskiy pseudo-random function [13] has an interesting feature: it roughly acts as the inverse function of the PRF: $(x, str) \mapsto H(str)^s$ that is used to generate the pair $(h = H(str), h^s)$ included in each EPID signature. More specifically, for any scalar $a \in \mathbb{Z}_p$, we note that $C = (h^a \cdot h^s)^{1/(s^*+a)}$ is either h if $s = s^*$ or some pseudo-random element otherwise. In particular, we stress that, in the latter case, C looks random even to an entity that knows a and s and that has access to other elements C generated with the same s^* but a different (still public) a'. This means that we can safely include C in the signature and use it as a proof of non-revocation. Actually, this property would hold even in the case where h would not be generated as a hash output but this would prevent us from relying on a reasonable computational assumption.

In all cases, this has two consequences. Firstly, each element a_i used to construct C_i (one per pair $(h_1^{(i)}, h_2^{(i)} = (h_1^{(i)})^{s_i})$ in SRL) can be revealed. Better still, it can be derived deterministically from the signature itself, for example by setting $a_i = H_p(\sigma||i)$ where H_p is some hash function mapping bitstrings to $\mathbb{Z}_p$ and σ is the first part of the EPID signature (namely, a randomized PS signature). The only condition on a_i is indeed that this scalar must be fresh, that is, a platform should never use the same a_i twice. This way, the scalars a_i have no impact on the proof size. Secondly, it means that the elements C_i along with a proof of well-formedness are sufficient to prove non-revocation: a platform is revoked if, and only if, $C_i = h_1^{(i)}$ for some i. Regarding the proof of well-formedness, we note that the signer must prove that $C_i^s = (h_1^{(i)}/C_i)^{a_i} \cdot h_2^{(i)}$, which only requires to send a constant number of elements if one uses the Schnorr's proof of knowledge [23]. Actually, this proof of well-formedness does not even increase the size of the signature as the platform already proves knowledge of s in the first part of the signature.

At this stage, our EPID signature generated with s^* contains a re-randomized PS certificate on s^* (proving that the platform has been enrolled), a pair (h, h^{s^*}) that can be used to revoke s^* and a list of elements (one per revoked signature) $C_i = (h_i^{a_i} \cdot h_i^{s_i})^{\frac{1}{s^*+a_i}}$ (where $\{s_i\}_i$ are the secret keys used to generate the revoked signatures) attesting non-revocation along with a zero-knowledge proof π_1 that all the elements are well-formed. At first sight, we are done:

by verifying π_1 and testing whether $C_i = h_i$ anyone can decide if the signature is valid and if it has been generated with a revoked key. Unfortunately, this is not true for a purely technical reason.

Indeed, let us recall that every existing EPID system makes use of a zero-knowledge proof of non-revocation. Generating the latter, even when one uses an unknown secret value s^*, is then not a problem as one can simply generate a simulated proof. In our case, we do not have such a zero-knowledge property. This is the reason behind efficiency of our system but it forces us to find a way to generate the elements C_i above even when our reduction does not know the value s^*. As we are essentially evaluating the Dodis-Yampolskyi PRF [13] with an unknown seed s^*, we would like to use the original strategy from [13] which astutely combines elements $g^{s^*}, \ldots, g^{(s^*)^q}$ from a q-DHI instance to handle further PRF evaluation queries. But this strategy does not readily work here because we evaluate the PRF on pairs $(h_i, h_i^{s_i})$ potentially generated by the adversary and not on some basis controlled by the security reduction. Of course, we can leverage the random oracle model to retain some control on these pairs, because h_i is some output of a hash function, but this is not sufficient as the values s_i can be unknown to our reduction. We indeed recall that the security model allows the adversary to control the issuer and some users in the anonymity game so nothing prevents the adversary from using a fresh s_i that the reduction has never seen.

There are essentially three ways of dealing with this problem. The first one is to fall back on the original EPID model where revocation lists only contain signatures generated by the reduction itself. This way, every pair $(h_i, h_i^{s_i})$ placed on SRL involves a scalar s_i known to the reduction and we can easily adapt the strategy from [13]. The price for this simple solution is a weaker security model. The second one is to alter the model so as to ensure that the reduction will know all secret keys, even corrupt ones. For example, one could assume a honest but curious issuer that would honestly perform registration, which would allow to extract all platforms secret keys. Technically, this would be a simple solution as it would not require any change in our EPID signature. However, the resulting model would be, at best, contrived so we favour the third solution where the reduction will extract the secret key s_i directly from the revoked signature. As s_i is a scalar involved in proofs of knowledge of discrete logarithms, encrypting it would not be convenient. We therefore prefer to extract it directly from the proof of knowledge, which has practical consequences: we now need a proof of knowledge supporting online extraction [15], which excludes the sole use of the classical (and very efficient) combination of Schnorr's protocol [23] with Fiat-Shamir methodology [14].

Concretely, our signature will still contain the proof π_1 of well-formedness, generated using [23] and [14] as usual, but will additionally[2] contain a Fischlin's proof of knowledge π_2 of s_i whose only purpose is to allow extraction of the latter value during the anonymity proof. This looks somewhat artificial, in particular because the statement proven by π_2 is already included in the ones proven by π_1, but it seems to be the simplest solution in our setting to avoid ad-hoc interactive assumptions. Regarding efficiency, this increases the complexity of our signature but the overhead is constant and so quickly amortized by what we gain with the proof of non-revocation itself. Indeed, our protocol essentially requires to send one element of $\mathbb{G}_1$ per revoked signature whereas the most efficient alternatives, based on the zero-knowledge proof in [11], require 1 element of $\mathbb{G}_1$ and 2 scalars per revoked signature.

[2]Actually, we could have used the Fischlin's protocol for all the statements proved in the signature but this would have significantly increased the computational cost of the signing process.

4.2 Construction.

- $\mathtt{Setup}(1^k)$: this algorithm returns the public parameters pp containing the description of a type-3 bilinear group $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ and of two hash functions $H_p : \{0,1\} \to \mathbb{Z}_p^*$ and $H_1 : \{0,1\} \to \mathbb{G}_1$ along with two generators $g \in \mathbb{G}_1$ and $\widetilde{g} \in \mathbb{G}_2$.
- $\mathtt{GKeygen}(pp)$: this algorithm generates a key pair $(\mathsf{isk}, \mathsf{ipk})$ for the PS signature scheme by setting $\mathsf{isk} = (x, y) \xleftarrow{\$} \mathbb{Z}_p^2$ and $\mathsf{ipk} = (pp, \widetilde{X} \leftarrow \widetilde{g}^x, \widetilde{Y} \leftarrow \widetilde{g}^y)$.
- Join: this protocol starts when a platform $\mathcal{P}$, taking as inputs ipk, contacts the issuer $\mathcal{I}$ for enrolment. It first generates a random $s \xleftarrow{\$} \mathbb{Z}_p$ and sends g^s to $\mathcal{I}$. $\mathcal{P}$ then engages in an interactive proof of knowledge of s with $\mathcal{I}$, using for example the Schnorr's protocol [23]. Once the latter is complete, $\mathcal{I}$ selects a random $r \xleftarrow{\$} \mathbb{Z}_p$ and computes a PS signature $\sigma = (\sigma_1, \sigma_2) \leftarrow (g^r, g^{r \cdot x} \cdot (g^s)^{r \cdot y})$ on s that it returns to $\mathcal{P}$. The platform then verifies this signature and stores (s, σ) as its secret key sk.
- $\mathtt{KeyRevoke}(\{\mathsf{sk}_i\}_{i=1}^m)$: this algorithm takes as input a set of m platform secret keys $\mathsf{sk}_i = (s^{(i)}, \sigma^{(i)})$ and returns a corresponding key revocation list KRL with $\mathtt{KRL}[i] = \mathsf{sk}_i$, for $i \in [1, m]$.
- $\mathtt{SigRevoke}(\{(\mu_i)\}_{i=1}^n)$: this algorithm takes as input a set of n EPID signatures $\{(\mu_i)\}_{i=1}^n$ and parses each of them as $((\sigma_1^{(i)}, \sigma_2^{(i)}), h_2^{(i)}, \{C_j\}_{j=1}^{n^{(i)}}, \Pi^{(i)})$. It then returns a signature revocation list SRL such that $\mathtt{SRL}[i] = (\sigma_1^{(i)}, h_2^{(i)})$, for $i \in [1, n]$.
- $\mathtt{Sign}(\mathsf{ipk}, \mathtt{SRL}, \mathsf{sk}, \mathsf{m})$: To sign a message m while proving that it has not been implicitly revoked by SRL, a platform $\mathcal{P}$ owning $\mathsf{sk} = (s, (\sigma_1, \sigma_2))$ generates a random $r \xleftarrow{\$} \mathbb{Z}_p^*$ and
  (1) re-randomizes the PS signature $(\sigma_1', \sigma_2') \leftarrow (\sigma_1^r, \sigma_2^r)$;
  (2) computes $(h_1, h_2) \leftarrow (H_1(\sigma_1'), h_1^s)$;
  (3) for all $i \in [1, n]$,
    - it parses $\mathtt{SRL}[i]$ as $(\sigma_1^{(i)}, h_2^{(i)})$ and computes $h_1^{(i)} \leftarrow H_1(\sigma_1^{(i)})$;
    - it computes $a_i \leftarrow H_p(\sigma_1' || i)$;
    - it computes $C_i = ((h_1^{(i)})^{a_i} \cdot h_2^{(i)})^{\frac{1}{s+a_i}}$;
  (4) it produces, using the Schnorr's protocol [23], a non-interactive zero-knowledge proof π_1 that (σ_1', σ_2') is a valid PS signature on s and that the elements h_2 and C_i are well-formed. More specifically,
    - it selects a random $k \xleftarrow{\$} \mathbb{Z}_p$;
    - it computes $K = h_1^k$
    - it computes $K' = e((\sigma_1')^k, \widetilde{Y})$;
    - it computes, for all $i \in [1, n]$, $K_i \leftarrow C_i^k$;
    - it computes $c = H_p(\sigma_1', \sigma_2', h_1, h_2, \{C_i, K_i\}_{i=1}^n, K, K', \mathsf{m})$;

– it computes $z = k + c \cdot s$;
– it sets $\pi_1 = \{c, z\}$

(5) it produces an extractable proof of knowledge π_2 of s such that $h_2 = h_1^s$ using Fischlin's protocol [15] (see Section 5).
(6) it returns the signature $\mu = ((\sigma_1', \sigma_2'), h_2, \{C_i\}_{i=1}^n, \Pi)$ where $\Pi = \{\pi_1, \pi_2\}$.

- $\texttt{Identify}(\mathsf{sk}, t)$: this algorithm parses sk as $(s, (\sigma_1, \sigma_2))$ and t as (σ_1, h_2), and returns 1 if $h_2 = H'(\sigma_1)^s$ and 0 otherwise.
- $\texttt{Verify}(\mathsf{ipk}, \mathsf{SRL}, \mathsf{KRL}, \mu, \mathsf{m})$: to verify an EPID signature μ, the verifier parses it as $((\sigma_1, \sigma_2), h_2, \{C_i\}_{i=1}^n, \Pi)$ and each $\mathsf{SRL}[i]$ as $(\sigma_1^{(i)}, h_2^{(i)})$, for $i \in [1, n]$. It then computes $a_i \leftarrow H_p(\sigma_1 || i)$ $\forall i \in [1, n]$ and returns 1 if all the following conditions hold and 0 otherwise.

(1) $\sigma_1 \neq 1_{\mathbb{G}_1}$;
(2) $\forall i \in [1, m]$, $\texttt{Identify}(\mathsf{KRL}[i], (\sigma_1, h_2)) = 0$;
(3) $\forall i \in [1, n]$, $C_i \neq H_1(\sigma_1^{(i)})$;
(4) $c = H_p(\sigma_1, \sigma_2, h_2, \{C_i, K_i\}_{i=1}^n, K, K', \mathsf{m})$, where $h_1 \leftarrow H_1(\sigma_1)$, $K \leftarrow h_1^z \cdot h_2^{-c}$, $K' \leftarrow e(\sigma_1, \widetilde{Y})^z \cdot [e(\sigma_1^{-1}, \widetilde{X}) \cdot e(\sigma_2, \widetilde{g})]^{-c}$ and $K_i \leftarrow C_i^z \cdot [(H_1(\sigma_1^{(i)})/C_i)^{a_i} \cdot h_2^{(i)}]^{-c}$.
(5) π_2 is valid.

Correctness. The second step of the verification process checks that μ has not been generated with one of the keys $\mathsf{sk}_i = (s^{(i)}, (\overline{\sigma}_1^{(i)}, \overline{\sigma}_2^{(i)}))$ placed on KRL. If this is true, then $s \neq s^{(i)}$ and we have $h_2 \neq H'(\sigma_1)^{s^{(i)}}$ for all $i \in [1, m]$. Similarly, the third step checks that μ has not been generated with a value s implicitly revoked by SRL. For a non-revoked s, we have $C_i = ((h_1^{(i)})^{a_i} \cdot h_2^{(i)})^{\frac{1}{s+a_i}} = ((h_1^{(i)})^{a_i} \cdot (h_1^{(i)})^{s^{(i)}})^{\frac{1}{s+a_i}}$ with $s^{(i)} \neq s$ for all $i \in [1, n]$. Hence, $C_i \neq h_1^{(i)} = H_1(\sigma_1^{(i)})$ and a valid signature passes this test. Finally, step 4 checks the validity of the Schnorr's proof. For a honestly generated signature, we have $z = k + c \cdot s$ and $\sigma_2 = \sigma_1^{x+y \cdot s}$, which means that:

- $K = h_1^z \cdot h_2^{-c} = h_i^k$
- $K' = e(\sigma_1, \widetilde{Y})^z \cdot [e(\sigma_1^{-1}, \widetilde{X}) \cdot e(\sigma_2, \widetilde{g})]^{-c} = e(\sigma_1, \widetilde{Y}^{k+c \cdot s}) \cdot e(\sigma_1, \widetilde{Y}^s)^{-c} = e(\sigma_1, \widetilde{Y}^k)$
- $K_i = C_i^z \cdot [(H_1(\sigma_1^{(i)})/C_i)^{a_i} \cdot h_2^{(i)}]^{-c} = C_i^{k+c \cdot s} \cdot [(H_1(\sigma_1^{(i)})/C_i)^{a_i} \cdot h_2^{(i)}]^{-c} = C_i^{a_i+s}]^c \cdot (H_1(\sigma_1^{(i)}))^{a_i} \cdot h_2^{(i)})^{-c} = C_i^k \cdot ((h_1^{(i)})^{a_i} \cdot h_2^{(i)})^c \cdot (H_1(\sigma_1^{(i)}))^{a_i} \cdot h_2^{(i)})^{-c} = C_i^k$

Therefore, the hash value generated during the last step of the verification process is exactly c.

Remark. In our proof of correctness, we have implicitly used the fact that the revocation list SRL is well-formed when we have proved that a valid signature always passes the third step of the verification process. We indeed assumed that $h_2^{(i)} = (h_1^{(i)})^{s^{(i)}}$ for some revoked secret $s^{(i)}$, which might not be true for a malicious revocation list. However, even in the latter case, we could still prove correctness by showing that an adversary is unlikely to revoke an honest user with an ill-formed SRL, as it is done in [22].

Our point here is that our scheme still works with malicious revocation lists. The only reason why we rule out the latter is that we cannot formally prove anonymity under a non-interactive assumption otherwise.

Theorem 4.1. *Let q be a bound on the number of $O\texttt{Sign}$ or $O\texttt{Sign}^*$ queries. In the random oracle model, our EPID system is*

- *unforgeable under the $(q+1)$-DL assumption and the EUF-CMA security of the PS signature if π_1 and π_2 are sound zero-knowledge proof systems;*
- *anonymous under the DDH and $(q+1)$-DHI assumptions if π_1 and π_2 are zero-knowledge proof systems and π_2 supports online extraction.*

4.3 Security Proofs

Unforgeability. An adversary $\mathcal{A}$ succeeding in the unforgeability game is able to produce $c+1$ valid signatures $\{\mu_i\}_{i=1}^{c+1}$ with only c corrupt keys, despite systematic revocation of its signature. We distinguish two cases.

- (type 1) Among all the signatures output by $\mathcal{A}$ during the game (that is, among the set $\{\mu_i\}_{i=1}^{c+1}$ extended with the signatures submitted to $O\texttt{SigRevoke}$), there is one signature $\mu = ((\sigma_1, \sigma_2), h_2, \{C_i\}_{i=1}^n, \Pi)$ such that $\texttt{Identify}(\mathsf{sk}, (\sigma_1, h_2)) = 0$ for all keys sk (honest or corrupt) generated during the experiment.
- (type 2) the previous situation does not occur.

In the first case, we will show that the adversary must have forged a certificate on a new secret key, which would imply an attack against PS signatures as stated by the following lemma.

Lemma 4.2. *Any type 1 adversary succeeding with probability ϵ can be converted into an adversary against the EUF-CMA security of PS signature.*

Proof. Let C be the challenger of the EUF-CMA security game for PS signatures, we construct a reduction $\mathcal{R}$ using $\mathcal{A}$ to win the latter game. At the beginning of the game, $\mathcal{R}$ receives a key pair for the PS signature scheme that it sets as the issuer's key pair $(\mathsf{isk}, \mathsf{ipk})$. Thanks to the signing oracle provided by C, it can trivially handle $O\texttt{Add}$ query. It can also deals with $O\texttt{Join}_c()$ by first extracting the secret value s from the interactive proof of knowledge and then querying s to its signing oracle. As it knows every honest platform secret keys, it can answer any $O\texttt{Cor}$ or $O\texttt{Sign}$ queries. Actually, $\mathcal{R}$ knows the secret value s of all platforms (honest or corrupt), which means that it can detect (using $\texttt{Identify}$) if the signatures output by $\mathcal{A}$ are legitimate (*i.e.* they have been produced by enrolled platforms) or not.

As we here consider type 1 adversary, we know that, at some point, $\mathcal{A}$ will output, either in a $O\texttt{SigRevoke}$ query or in the list $\{\mu_i\}_{i=1}^{c+1}$, a signature μ that cannot be associated with any platform secret key. This means that $\mu = ((\sigma_1, \sigma_2), h_2, \{C_i\}_{i=1}^n, \Pi)$ is such that $\texttt{Identify}(\mathsf{sk}, (\sigma_1, h_2)) = 0$ for every platform secret key sk. Let s be the secret value contained in sk. The latter equation means that $h_2 \neq H_1(\sigma_1)^s$ for every s submitted to the PS signing oracle. Thanks to the soundness of π_1, this means that there is some $s^* \neq s$ $\forall s$ such that

$$e(\sigma_1, \widetilde{Y})^{s^*} = e(\sigma_1^{-1}, \widetilde{X}) \cdot e(\sigma_2, \widetilde{g}).$$

Therefore, (σ_1, σ_2) is a valid PS signature on s^*, which has never been queried to C. $\mathcal{R}$ can then extract s^* from Π and return s^* and (σ_1, σ_2) as a valid forgery, which concludes the proof. □

We now show that a type 2 adversary must have produced a signature on behalf of some honest platform and can thus be used to perform discrete logarithm computations.

Lemma 4.3. *Let q_a be a bound on the number of $O\mathsf{Add}$ queries. Any type 2 adversary succeeding with probability ϵ can be converted into an adversary against the $(q+1)$-DL assumption succeeding with probability at least $\frac{\epsilon}{q_a}$.*

Proof. Let $(g, g^x, \ldots, g^{x^{q+1}}) \in \mathbb{G}_1^{q+2}$ and $(\widetilde{g}, \widetilde{g}^x) \in \mathbb{G}_1^2$ be a $(q+1)$-DL instance. As we here consider a type-2 adversary, any signature output by the adversary must be associated with some secret keys sk, either honest or corrupt. If we more specifically consider the set $\{\mu_i\}_{i=1}^{c+1}$, we note that one of them must have been generated on behalf of some honest platform. Indeed, let $\{sk_i\}_{i=1}^{c+1}$ be the secret keys associated with the signatures $\{\mu_i\}_{i=1}^{c+1}$. As the adversary only owns c secret keys, there is at least two indices $i \neq j$ such that $\mathsf{sk}_i = \mathsf{sk}_j$ or $\exists i$ such that sk_i is honest (since we consider type 2 adversary). In the former case, let us assume without loss of generality that $i < j$. Then, μ_j must contain an element C_i proving that sk_j is not implicitly revoked by $(\sigma_1^{(i)}, h_2^{(i)})$. However, this can work only if $h_2^{(i)}$ or C_i is ill-formed, which in both case implies an attack against the soundness of the Schnorr's protocol. We can then assume that only the latter case occurs. The reduction $\mathcal{R}$ therefore makes a guess on the identifier k^* of the corresponding honest platform and then generates the issuer key pair (isk, ipk). It can then answer the oracle queries as follows.

- H_p: Let str be the string queried to H_p. $\mathcal{R}$ returns a random $a \xleftarrow{\$} \mathbb{Z}_p$ if this is the first query on str or returns the original answer otherwise.
- H_1: Let str be the string queried to H_1. If this is the first time that str is requested, then $\mathcal{R}$ generates q random scalars $a_1, \ldots, a_q$ along with some $u \xleftarrow{\$} \mathbb{Z}_p$ and computes $h = g^u \prod_{i=1}^q (x+a_i)$ using the elements from the $q+1$-DL instance. The tuple $(h, u, a_1, \ldots, a_q)$ is then stored by $\mathcal{R}$ whereas h is returned to $\mathcal{A}$. For any further query on str, $\mathcal{R}$ returns the same element h.
- $O\mathsf{Add}$: $\mathcal{R}$ proceeds normally for any query on $k \neq k^*$ thanks to its knowledge of isk. For k^*, it proceeds as if the platform secret key was x. This is possible since g^x is sufficient to issue a PS signature on x.
- $O\mathsf{Join}_c()$: $\mathcal{R}$ proceeds normally thanks to its knowledge of isk but extracts the corrupt secret value s from the interactive proof of knowledge.
- $O\mathsf{Cor}()$: $\mathcal{R}$ hands over the requested platform secret key except if this oracle is queried on k^*, in which case $\mathcal{R}$ aborts. The latter case never occurs if the guess on k^* was right.
- $O\mathsf{SigRevoke}$: given a list of signatures $\{\mu_i\}_{i=1}^n$, $\mathcal{R}$ first checks their validity and rejects this list if one of these signatures does not pass verification. Else, it parses μ_i as $((\sigma_1^{(i)}, \sigma_2^{(i)}), h_2^{(i)}, \Pi^{(i)})$ and uses $\widetilde{g}^x$ to test if $\mathsf{Identify}(\mathsf{sk}_{k^*}, (\sigma_1^{(i)}, h_2^{(i)})) = 1$ for all $i \in [1, n]$. Indeed:

$$\begin{aligned}&\mathsf{Identify}(\mathsf{sk}_{k^*}, (\sigma_1^{(i)}, h_2^{(i)})) = 1\\ \Leftrightarrow &H_1(\sigma_1^{(i)})^x = h_2^{(i)}\\ \Leftrightarrow &e(H_1(\sigma_1^{(i)}), \widetilde{g}^x) = e(h_2^{(i)}, \widetilde{g}),\end{aligned}$$

  which means that the output of $\mathsf{Identify}(\mathsf{sk}_{k^*}, .)$ can be computed without the knowledge of x. In the case where $\mathsf{Identify}$ returns 1 on some signature μ and yet μ has never been produced by $\mathcal{R}$, then one can extract x from the proof of knowledge Π contained in μ. Else, $\mathcal{R}$ outputs $\mathsf{SRL} = \{(\sigma_1^{(i)}, h_2^{(i)})\}_{i=1}^n$.
- $O\mathsf{Sign}$: if this oracle is queried on $k \neq k^*$, then $\mathcal{R}$ can generate a signature as usual as it knows the corresponding secret key sk. Else, It proceeds as follows to generate the signature:
  (1) it re-randomizes the PS signature on x, yielding (σ_1', σ_2')
  (2) it queries σ_1' to H_1 and thus receives $h = g^u \prod_{i=1}^q (x+a_i)$ for some random u and $\{a_i\}_{i=1}^q$ that it knows.
  (3) it computes $h_2 = h^x = g^{u \cdot x} \prod_{i=1}^q (x+a_i)$, which is possible from the $q+1$-DL instance as $x \prod_{i=1}^q (x+a_i)$ is of degree $q+1$.
  (4) For every $i \in [1, n]$, we have $\mathsf{SRL}[i] = (\sigma_1^{(i)}, h_2^{(i)})$ where the latter pair has been extracted from a valid signature during a previous $O\mathsf{SigRevoke}$ query. Let $j \leq q$ be the number of times that this pair has been involved in a signature query so far and $a_1^{(i)}, \ldots, a_q^{(i)}$ be the scalars used to generate $H_1(\sigma_1^{(i)})$. The reduction programs H_p to return $a_j^{(i)}$ on input $(\sigma_1' || i)$. In the very unlikely case where $(\sigma_1' || i)$ has already been queried to H_p, $\mathcal{R}$ simply returns to step 1. Now we note that
$$C_i = (H_1(\sigma_1^{(i)}))^{a_j^{(i)}} \cdot (H_1(\sigma_1^{(i)}))^s)^{\frac{1}{x+a_j^{(i)}}}$$
  for some secret s. In the case where $s = x$, then $C_i = H_1(\sigma_1^{(i)})$ and we can move to the next step. In the other cases, we know that s is some value certified by the issuer during a $O\mathsf{Join}_c()$ or a $O\mathsf{Add}$ query as we here consider type 2 adversary. In both cases, $\mathcal{R}$ knows s so it can compute:
$$C_i = (g^{u^{(i)} \prod_{\ell=1}^q (x+a_\ell^{(i)})})^{\frac{a_j^{(i)}+s}{a_j^{(i)}+x}} = g^{u^{(i)}(a_j^{(i)}+s) \prod_{\ell=1, \ell \neq j}^q (x+a_\ell^{(i)})}$$
  (5) $\mathcal{R}$ simulates the proof of knowledge of x to sign m, thus completing Π.
  (6) Finally, it returns $\mu = ((\sigma_1', \sigma_2'), h_2, \{C_i\}_{i=1}^n, \Pi)$, which is a valid signature on m on behalf of the platform k^*.

  At some point, assuming that $\mathcal{R}$ has not already succeeded in $O\mathsf{SigRevoke}$, $\mathcal{A}$ outputs a set of signature $\{\mu_i\}_{i=1}^{c+1}$ such that there is $i^* \in [1, c+1]$ and an honest sk_k satisfying $\mathsf{Identify}(\mathsf{sk}_k, (\sigma_1^{(i)}, h_2^{(i)})) = 1$, as we have explained at the beginning of this proof. If $k = k^*$, which occurs with probability at least $\frac{1}{q_a}$, then $\mathcal{R}$ can extract x from the proof of knowledge contained in μ_{i^*}, which concludes the proof.

□

Anonymity. Let $\mathcal{A}$ be an adversary succeeding against anonymity with advantage ϵ, q be a bound on the number of $O\mathtt{Sign}^*$ queries, q' be a bound on the size of the revocations lists SRL and q_a be a bound on the number of $O\mathtt{Join}_h$ queries. We prove anonymity by using a sequence of games where Game 1,b is exactly the experiment $\mathtt{Exp}^{an-b}_{\mathcal{A}}$. For each i, we define $\mathtt{Adv}_i$ as the advantage of $\mathcal{A}$ playing Game i, 0 and Game i, 1. In the following, we will omit the parameter b as both games are identical (except the parameter itself).

Game 1. By definition, this is exactly experiment $\mathtt{Exp}^{an-b}_{\mathcal{A}}$ and we thus have $\mathtt{Adv}_1 = \epsilon$.

Game 2. Here, the reduction $\mathcal{R}$ proceeds as in the previous game except that it randomly selects $k_0, k_1 \in [1, q_a]$ and aborts if the two signatures μ_0 and μ_1 returned by $\mathcal{A}$ in the challenge phase were not generated by the platforms k_0 and k_1. For valid guesses on k_0 and k_1, this does not affect the behaviour of $\mathcal{R}$, which means that $\mathtt{Adv}_2 \geq \frac{\mathtt{Adv}_1}{q_a^2}$.

Game 3. In this game, $\mathcal{R}$ proceeds as previously except that it aborts if $\mathcal{A}$ queries $O\mathtt{SigRevoke}$ with a list containing a signature $\mu = ((\sigma_1, \sigma_2), h_2, \{C_i\}_{i=1}^n, (\pi_1, \pi_2))$ such that $\mathcal{R}$ cannot extract the secret value s from the proof π_2. We note that Games 2 and 3 are indistinguishable unless $\mathcal{A}$ manages to break extractability of π_2. We thus have $\mathtt{Adv}_3 \geq \mathtt{Adv}_2 - \mathtt{Adv}_{\mathsf{ext}}(\mathcal{A})$.

Game 4. In this game, $\mathcal{R}$ proceeds as in Game 3 except that each enrolled platform simulates the proof of knowledge during registration. We then have $\mathtt{Adv}_4 \geq \mathtt{Adv}_3 - \mathtt{Adv}_{ZK'}$ where $\mathtt{Adv}_{ZK'}$ is the advantage of an adversary against the zero-knowledge property of the proof system used in Join.

Game 5. In this game, $\mathcal{R}$ proceeds as previously except that it simulates the zero-knowledge proofs for all the signatures it generates. We then have $\mathtt{Adv}_5 \geq \mathtt{Adv}_4 - \mathtt{Adv}_{ZK_1} - \mathtt{Adv}_{ZK_2}$ where $\mathtt{Adv}_{ZK_i}$ is the advantage of an adversary against the zero-knowledge property of the system used to generate π_i, for $i \in \{1, 2\}$.

Game 6, j, i. In this game, defined for all $j \in [1, q]$ and $i \in [1, q']$, the reduction proceeds as in Game 5 except that:

- the first $j-1$ signatures generated by k_b during a $O\mathtt{Sign}^*$ query are answered by returning a signature μ containing $\{C_\ell\}_{\ell=1}^n$ where all the elements C_ℓ are replaced by random elements of $\mathbb{G}_1$.
- the j-th signature generated by k_b during a $O\mathtt{Sign}^*$ query is answered by returning a signature μ whose i first elements C_ℓ are replaced by random elements of $\mathbb{G}_1$.

We prove below that $\forall j \in [1, q-1]$ and $i \in [1, q'-1]$, $\mathtt{Adv}_{6,j,i+1} \geq \mathtt{Adv}_{6,j,i} - \mathtt{Adv}_{(q+1)-\mathsf{DHI}}$ and that $\mathtt{Adv}_{6,j+1,1} \geq \mathtt{Adv}_{6,j,q'} - \mathtt{Adv}_{(q+1)-\mathsf{DHI}}$.

Game 7, i. In this game, defined for all $i \in [1, q']$, the reduction proceeds as in Game 6, q, q' except that the i first elements C_ℓ contained in μ^* are now replaced by random elements of $\mathbb{G}_1$. We prove below that $\mathtt{Adv}_{7,i+1} \geq \mathtt{Adv}_{7,i} - \mathtt{Adv}_{(q+1)-\mathsf{DHI}}$

Game 8. In this game, $\mathcal{R}$ proceeds as in Game 7, q' except that the element h_2 from μ^* is replaced by a random element of $\mathbb{G}_1$. We prove below that $\mathtt{Adv}_8 \geq \mathtt{Adv}_{7,q'} - \mathtt{Adv}_{\mathsf{DDH}}$.

Game 9. In this last game, $\mathcal{R}$ proceeds as in Game 8 except that the PS signature (σ_1, σ_2) from μ^* is replaced by a PS signature on some random secret key. We prove below that $\mathtt{Adv}_9 \geq \mathtt{Adv}_8 - \mathtt{Adv}_{\mathsf{DDH}}$.

In the end, we thus have:

$$\mathtt{Adv}_9 \geq \frac{\epsilon}{q_a^2} - (q+1)q'\mathtt{Adv}_{(q+1)-\mathsf{DHI}} - 2\mathtt{Adv}_{\mathsf{DDH}} - \mathtt{Adv}_{ZK_1} - \mathtt{Adv}_{ZK_2} - \mathtt{Adv}_{ZK'} - \mathtt{Adv}_{\mathsf{ext}}(\mathcal{A})$$

In the last game, we note that μ^* only contains random elements, a signature on a random key and a simulated proof of knowledge. The advantage $\mathtt{Adv}_9$ is then at most negligible, which concludes the proof.

Proof related to Game 6 and Game 7. We note that the proofs that $\mathtt{Adv}_{6,j,i+1} \geq \mathtt{Adv}_{6,j,i} - \mathtt{Adv}_{(q+1)-\mathsf{DHI}}$, $\mathtt{Adv}_{6,j+1,1} \geq \mathtt{Adv}_{6,j,q'} - \mathtt{Adv}_{(q+1)-\mathsf{DHI}}$ and $\mathtt{Adv}_{7,i+1} \geq \mathtt{Adv}_{7,i} - \mathtt{Adv}_{(q+1)-\mathsf{DHI}}$ are exactly the same as they all consist in replacing one element C_i by a random one. We here specifically describe the proof that $\mathtt{Adv}_{7,i+1} \geq \mathtt{Adv}_{7,i} - \mathtt{Adv}_{(q+1)-\mathsf{DHI}}$, which readily adapts to the other games.

Let $(g, g^x, \ldots, g^{x^q}) \in \mathbb{G}_1^{q+1}$ along with $g^z \in \mathbb{G}_1$ be a $(q+1)$-DHI instance, the goal is to decide whether $z = \frac{1}{x}$ or not. We select some random $a \xleftarrow{\$} \mathbb{Z}_p$ and set $y = x - a$. We stress that any element $g^{f(y)}$ can be computed from the $q+1$-DHI instance for every polynomial f of degree smaller than $q+1$. The reduction then generates the issuer key pair (isk, ipk) and answers the oracle queries as follows.

- H_p: Let str be the string queried to H_p. $\mathcal{R}$ returns a random $a' \xleftarrow{\$} \mathbb{Z}_p$ if this is the first query on str or returns the original answer otherwise.
- H_1: Let str be the string queried to H_1. If this is the first time that str is requested, then $\mathcal{R}$ generates q random scalars $a_1, \ldots, a_q$ along with some $u \xleftarrow{\$} \mathbb{Z}_p$ and computes $h = g^{u\prod_{j=1}^q (y+a_j)}$ using the elements from the $q+1$-DL instance. The tuple $(str, h, u, a_1, \ldots, a_q)$ is then stored by $\mathcal{R}$ whereas h is returned to $\mathcal{A}$. For any further query on str, $\mathcal{R}$ returns the same element h.
- $O\mathtt{Add}$: $\mathcal{R}$ proceeds normally for any query on $k \neq k_b$ thanks to its knowledge of the platform secret key. For k_b, it proceeds as if the platform secret key was y. This is possible since $g^y = g^x \cdot g^{-a}$ is sufficient to issue a PS signature on y and since $\mathcal{R}$ simulates the zero-knowledge proof during registration (Game 4).
- $O\mathtt{Cor}()$: $\mathcal{R}$ hands over the requested platform secret key. Since Game 2, we are indeed ensured that $\mathcal{A}$ will not query this oracle on k_0 or k_1.
- $O\mathtt{SigRevoke}$: given a list of signatures $\{\mu_j\}_{j=1}^n$ and corresponding messages $\{\mathsf{m}_j\}_{j=1}^n$, $\mathcal{R}$ first checks their validity and rejects this list if one of these signatures does not pass verification. Since Game 3, $\mathcal{R}$ knows the secret values s_j used to generate μ_j and so is able to determine the set of honest users that are not revoked by $\{\mu_j\}_{j=1}^n$, which will be useful to handle the following queries. $\mathcal{R}$ then returns $\mathtt{SRL} \leftarrow \mathtt{SigRevoke}(\mathsf{ipk}, (\mu_j, \mathsf{m}_j))$ to $\mathcal{A}$.
- $O\mathtt{Sign}^*$: As we here consider honest revocation lists, we are ensured that:

(1) each element $(\sigma_1^{(j)}, h_2^{(j)})$ in $\mathsf{SRL}[j]$ is such that $h_2^{(j)} = H_1(\sigma_1^{(j)})^s$ for some scalar s that is known since Game 3.
(2) $\mathcal{R}$ knows the exact set of honest users that are not revoked by SRL, as explained above. It can then select some random identifier k among them.

If $k \neq k_b$, then $\mathcal{R}$ can generate a signature as usual as it knows the corresponding secret key sk. Else, It proceeds as follows to generate the signature.

(1) It re-randomizes the PS signature on y, yielding (σ_1', σ_2').
(2) it queries σ_1' to H_1 and thus receives $h = g^{u \prod_{j=1}^{q}(y+a_j)}$ for some random u and $\{a_j\}_{j=1}^{q}$ that it knows.
(3) it computes $h_2 = h^y = g^{u \cdot y \prod_{j=1}^{q}(y+a_j)}$, which is possible from the $q+1$-DL instance as $y \prod_{j=1}^{q}(y+a_j)$ is of degree $q+1$.
(4) As we are here considering Games 7, i and 7, $i+1$ the elements C_ℓ are random elements of $\mathbb{G}_1$. In the case of proofs of Games 6, j, i, one can use the strategy described below for μ^* to generate non-random C_ℓ.
(5) $\mathcal{R}$ simulates the proof of knowledge of y to sign m, thus completing Π.
(6) Finally, it returns $\mu = ((\sigma_1', \sigma_2'), h_2, \{C_\ell\}_{\ell=1}^{n}, \Pi)$, as a signature on m on behalf of the platform k^*.

Eventually, $\mathcal{A}$ outputs during the challenge phase two signatures μ_0 and μ_1 along with a message and a signature revocation list SRL that has been output by $\mathcal{R}$ during a previous $O\mathsf{SigRevoke}$ query. At this stage, we know since Game 2 that $\mathcal{R}$ must produce a signature on behalf of k_b. We then proceed exactly as in $O\mathsf{Sign}^*$ except that:

(1) the i first elements C_j are replaced by random elements of $\mathbb{G}_1$ as specified in Game 7, i.
(2) the elements $\{C_j\}_{j=i+2}^{n}$ are generated as follows. First we recall that, for every $j \in [1, n]$, we have $\mathsf{SRL}[j] = (\sigma_1^{(j)}, H_1(\sigma_1^{(j)})^s)$ for some s that has been extracted by $\mathcal{R}$, as explained above. Let $\ell \leq q$ be the number of times that this pair has been involved in a signature query so far and $a_1^{(j)}, \ldots, a_q^{(j)}$ be the scalars used to generate $H_1(\sigma_1^{(j)})$. The reduction programs H_p to return $a_\ell^{(j)}$ on input $(\sigma_1' || j)$. In the very unlikely case where $(\sigma_1' || j)$ has already been queried to H_p, $\mathcal{R}$ simply returns to step 1. Now we note that

$$C_j = ((H_1(\sigma_1^{(j)}))^{a_\ell^{(j)}} \cdot (H_1(\sigma_1^{(j)}))^s)^{\frac{1}{y+a_\ell^{(j)}}}.$$

As $\mathcal{R}$ knows s, it can compute:

$$C_j = (g^{u^{(j)} \prod_{t=1}^{q}(y+a_t^{(j)})})^{\frac{a_\ell^{(j)}+s}{a_\ell^{(j)}+y}} = (g^{u^{(j)} \prod_{t=1, t\neq\ell}^{q}(y+a_t^{(j)})})^{a_\ell^{(j)}+s}$$

(3) To generate C_{i+1}, the reduction programs H_p to return a on input $(\sigma_1' || i+1)$ and must then compute

$$\begin{aligned} C_{i+1} &= ((H_1(\sigma_1^{(i+1)}))^a \cdot (H_1(\sigma_1^{(i+1)}))^s)^{\frac{1}{y+a}} \\ &= ((H_1(\sigma_1^{(i+1)}))^a \cdot (H_1(\sigma_1^{(i+1)}))^s)^{\frac{1}{x}} \end{aligned}$$

where $H_1(\sigma_1^{(i+1)}) = g^{u^{(i+1)} \prod_{t=1}^{q}(y+a_t^{(i+1)})}$. We note that

$$\prod_{t=1}^{q}(y + a_t^{(i+1)}) = x \cdot f(x) + \alpha$$

where $f(x)$ is some (known) polynomial in x and α is a known scalar. Therefore C_{i+1} is supposed to be equal to:

$$(g^{f(x)} \cdot g^{\frac{\alpha}{x}})^{u(a+s)}.$$

$\mathcal{R}$ then sets $C_{i+1} = (g^{f(x)} \cdot (g^z)^\alpha)^{u(a+s)}$ and includes it in the signature. If $z = \frac{1}{x}$, then this is exactly Game 7, i. Else C_{i+1} is a random element of $\mathbb{G}_1$ and we are playing Game 7, $i+1$. Any adversary able to distinguish these two games can then succeed against the $q+1$-DHI assumption.

Proof related to Game 8. Let $(g, g^x, g^y, g^z) \in \mathbb{G}^4$ be a DDH instance. Our reduction $\mathcal{R}$ proceeds as follows to answer oracle queries. We recall that, since Game 7, q', every element C_i involved in a signature generated by k_b has been replaced by a random element.

- H_p: in this proof, we do not need to program H_p and so can treat it as a standard hash function.
- H_1: Let str be the string queried to H_1. If this is the first time that str is requested, then $\mathcal{R}$ generates a random scalar a and computes $h = g^a$. The pair (str, a) is then stored by $\mathcal{R}$ whereas h is returned to $\mathcal{A}$. For any further query on str, $\mathcal{R}$ returns the same element h.
- $O\mathsf{Add}$: $\mathcal{R}$ proceeds normally for any query on $k \neq k_b$ thanks to its knowledge of the platform secret key. For k_b, it proceeds as if the platform secret key was x. This is possible since g^x is sufficient to issue a PS signature on x and since ZK proofs are simulated during registration (Game 4).
- $O\mathsf{Cor}()$: $\mathcal{R}$ hands over the requested platform secret key. Since Game 2, we are indeed ensured that $\mathcal{A}$ will not query this oracle on k_0 or k_1.
- $O\mathsf{SigRevoke}$: given a list of signatures $\{\mu_j\}_{j=1}^{n}$ and corresponding messages $\{\mathsf{m}_j\}_{j=1}^{n}$, $\mathcal{R}$ first checks their validity and rejects this list if one of these signatures does not pass verification. Since Game 3, $\mathcal{R}$ knows the secret values s_j used to generate μ_j and so is able to determine the set of honest users that are not revoked by $\{\mu_j\}_{j=1}^{n}$, which will be useful to handle the following queries. $\mathcal{R}$ then returns $\mathsf{SRL} \leftarrow \mathsf{SigRevoke}(\mathsf{ipk}, (\mu_j, \mathsf{m}_j))$ to $\mathcal{A}$.
- $O\mathsf{Sign}^*$: As we here consider honest revocation lists, we are ensured that $\mathcal{R}$ knows the exact set of honest users that are not revoked by SRL, as explained above. It can then select some random identifier k among them.

  If $k \neq k_b$, then $\mathcal{R}$ can generate a signature as usual as it knows the corresponding secret key sk. Else, It proceeds as follows to generate the signature.

  (1) It re-randomizes the PS signature on x, yielding (σ_1', σ_2').
  (2) it queries σ_1' to H_1 and thus receives $h = g^a$ for some random a that it knows.
  (3) it computes $h_2 = h^x = (g^x)^a$ using g^x and a.
  (4) As we are here considering Game 8, the elements C_ℓ are random elements of $\mathbb{G}_1$.

(5) $\mathcal{R}$ simulates the proofs of knowledge of y to sign m, thus completing Π.
(6) Finally, it returns $\mu = ((\sigma'_1, \sigma'_2), h_2, \{C_\ell\}_{\ell=1}^n, \Pi)$, as a signature on m on behalf of the platform k^*.

Eventually, $\mathcal{A}$ outputs during the challenge phase two signatures μ_0 and μ_1 along with a message and a signature revocation list SRL that has been output by $\mathcal{R}$ during a previous OSigRevoke query. At this stage, we know since Game 2 that $\mathcal{R}$ must produce a signature on behalf of k_b. We then proceed exactly as in OSign* except that:

- $\mathcal{R}$ programs H_1 to return g^y on input σ'_1. In the very unlikely case where σ'_1 has already been queried to H_1, $\mathcal{R}$ re-randomizes differently the PS signature on x.
- $\mathcal{R}$ sets $h_2 = g^z$.

If $z = x \cdot y$, then this is exactly Game 7, q'. Conversely, a random z corresponds to Game 8. Any adversary able to distinguish these two games is thus able to break the DDH assumption.

Proof related to Game 9. Let $(g, g^x, g^y, g^z) \in \mathbb{G}^4$ be a DDH instance. Our reduction $\mathcal{R}$ proceeds as follows to answer oracle queries.

- H_p: in this proof, we do not need to program H_p and so can treat it as a standard hash function.
- H_1: Let str be the string queried to H_1. If this is the first time that str is requested, then $\mathcal{R}$ generates a random scalar a and computes $h = g^a$. The pair (str, a) is then stored by $\mathcal{R}$ whereas h is returned to $\mathcal{A}$. For any further query on str, $\mathcal{R}$ returns the same element h.
- OAdd: $\mathcal{R}$ proceeds normally for any query on $k \neq k_b$ thanks to its knowledge of the platform secret key. For k_b, it proceeds as if the platform secret key was x. This is possible since g^x is sufficient to issue a PS signature on x and since ZK proofs are simulated during registration (Game 4).
- OCor(): $\mathcal{R}$ hands over the requested platform secret key. Since Game 2, we are indeed ensured that $\mathcal{A}$ will not query this oracle on k_0 or k_1.
- OSigRevoke: given a list of signatures $\{\mu_j\}_{j=1}^n$ and corresponding messages $\{\mathsf{m}_j\}_{j=1}^n$, $\mathcal{R}$ first checks their validity and rejects this list if one of these signatures does not pass verification. Since Game 3, $\mathcal{R}$ knows the secret values s_j used to generate μ_j and so is able to determine the set of honest users that are not revoked by $\{\mu_j\}_{j=1}^n$, which will be useful to handle the following queries. $\mathcal{R}$ then returns SRL $\leftarrow$ SigRevoke(ipk, (μ_j, m_j)) to $\mathcal{A}$.
- OSign*: As we here consider honest revocation lists, we are ensured that $\mathcal{R}$ knows the exact set of honest users that are not revoked by SRL, as explained above. It can then select some random identifier k among them.
  If $k \neq k_b$, then $\mathcal{R}$ can generate a signature as usual as it knows the corresponding secret key sk. Else, It proceeds as follows to generate the signature.
  (1) It re-randomizes the PS signature on x, yielding (σ'_1, σ'_2).
  (2) it queries σ'_1 to H_1 and thus receives $h = g^a$ for some random a that it knows.
  (3) it computes $h_2 = h^x = (g^x)^a$ using g^x and a.
  (4) As we are here considering Game 9, the elements C_ℓ are random elements of $\mathbb{G}_1$.
  (5) $\mathcal{R}$ simulates the proofs of knowledge of y to sign m, thus completing Π.
  (6) Finally, it returns $\mu = ((\sigma'_1, \sigma'_2), h_2, \{C_\ell\}_{\ell=1}^n, \Pi)$, as a signature on m on behalf of the platform k^*.

Eventually, $\mathcal{A}$ outputs during the challenge phase two signatures μ_0 and μ_1 along with a message and a signature revocation list SRL that has been output by $\mathcal{R}$ during a previous OSigRevoke query. At this stage, we know since Game 2 that $\mathcal{R}$ must produce a signature on behalf of k_b. $\mathcal{R}$ then proceeds as follows.

- it sets $\sigma'_1 = g^y$ and $\sigma'_2 = (g^y)^\alpha \cdot (g^z)^\beta$ where (α, β) is the issuer's secret key isk.
- it selects a random h_2, which is possible since Game 8.
- As we are here considering Game 9, the elements C_ℓ are random elements of $\mathbb{G}_1$.
- $\mathcal{R}$ simulates the proofs of knowledge of y to sign m, thus completing Π.
- Finally, it returns $\mu = ((\sigma'_1, \sigma'_2), h_2, \{C_\ell\}_{\ell=1}^n, \Pi)$, as a signature on m on behalf of the platform k^*.

If $z = x \cdot y$, then (σ'_1, σ'_2) is a valid PS signature on x and we are playing Game 8. Conversely, a random z implies that (σ'_1, σ'_2) is a certificate on some random key and we are playing Game 9. Any adversary able to distinguish these two games is thus able to break the DDH assumption.

5 EFFICIENCY

Before comparing the efficiency of our scheme with the state-of-the-art, we first need to recall some elements on the Fischlin's proof of knowledge with online extractor [15] that we use to instantiate π_2 in Section 4.

Fischlin's proof of knowledge. In our construction, each signature contains an extractable proof π_2 that $h_2 = h_1^s$, where s is the platform secret key. We use the protocol from [15] that depends on some parameters r, b, t, S and that makes use of a hash function H outputting b-bits strings seen as integers in $[0, 2^b - 1]$. It essentially works as follows.

[Prover]
(1) compute r commitments $h_1^{k_i}$ for random scalars $k_i \in \mathbb{Z}_p$
(2) for each $i \in [1, r]$:
  - for each $c_i \in [0, 2^t - 1]$, compute $z_i = k_i + c_i \cdot s$ and $H(\{h_1^{k_i}\}_{i=1}^r, i, c_i, z_i)$
  - select the value c_i^* yielding the smallest hash value among the 2^t hash values above
(3) output $\pi = \{h_1^{k_i}, c_i^*, z_i^*\}_{i=1}^r$

[Verifier]
(1) parse π as $\{h_1^{k_i}, c_i, z_i\}_{i=1}^r$
(2) if $\sum_{i=1}^r H(\{h_1^{k_i}\}_{i=1}^r, i, c_i, z_i) > S$, then return 0.
(3) for each $i \in [1, r]$:
  - if $h_1^{k_i} \cdot h_2^{c_i} \neq h_1^{z_i}$, then return 0
(4) return 1

Intuitively, the core idea of this protocol is that the prover must try several values (c_i, z_i) to get a sufficiently small hash output. Otherwise, the verifier will reject the proof at step 2 with overwhelming probability. Since these (challenge,response) pairs are

	σ	Sign	Verify
[8]	$(2+n)\mathbb{G}_1 + (2n+5)\mathbb{Z}_p$ $= 2044 + 894n$ bits	$(6n{+}7)\mathrm{e}_1{+}1\mathrm{p}_4{+}(n{+}2)\mathsf{H}$	$(m+n+2)\mathsf{H} + (m+$ $6n+8)\mathrm{e}_1 + 1\mathrm{p}_6$
[22], Sec. 5	$(4+n)\mathbb{G}_1 + (2n+2)\mathbb{Z}_p$ $= 2040 + 894n$ bits	$(6n+5)\mathrm{e}_1 + (n+2)\mathsf{H}$	$(m+n+2)\mathsf{H} + (m+$ $6n+4)\mathrm{e}_1 + 1\mathrm{p}_3$
Our construction	$(3{+}n)\mathbb{G}_1{+}(2{+}r)\mathbb{Z}_p{+}r{\cdot}t$ $= 4338 + 382n$ bits	$(5+3n+r)\mathrm{e}_1 + 1\mathrm{p}_1 +$ $(2+2n+r\cdot 2^t)\mathsf{H}$	$(2+2n+m+r)\mathsf{H} + (5+$ $m+2r+3n)\mathrm{e}_1 + 1\mathrm{p}_3$

Table 1: Complexity of our construction and the ones from [8] and [22]. We consider a signature revocation list SRL (resp. a key revocation list KRL) containing n elements (resp. m elements). Here, H denotes the evaluation of a hash function, e_i denotes an exponentiation in $\mathbb{G}_i$, for $i \in \{1,2\}$, and p_k denotes the batch computation of k pairings.

generated and submitted to the random oracle for the *same* commitment $h_1^{k_i}$, one can extract the secret value s without rewinding by computing $\frac{z_i - z_i'}{c_i - c_i'}$. However, this only works if:

(1) the protocol remains complete: an honest prover should be able to output a convincing proof with high probability;
(2) no adversary querying the random oracle H only once for each $i \in [1, r]$ is able to output a valid proof with non-negligible probability.

By carefully selecting the parameters r, b, t, S, one can ensure that both conditions are satisfied. We here follow the recommendations from [15] and thus define $r = S = 10$, $b = 9$ and $t = 12$ which ensures a completeness error of only 2^{-83} while assuring that no adversary can foil the extraction procedure with probability greater than 2^{-78} for each proof. By slightly increasing r or b, one can significantly reduce the latter probability, if necessary. However, this would require to also increase t (in particular in the case where b is increased) to retain completeness.

As noted in Section 4 of [15], there is actually no need to include the elements $\{h_1^{k_i}\}_{i=1}^r$ in the proof since they can be recovered by computing $h_1^{z_i} \cdot h_2^{-c_i}$ so the proof size is $r(t + |p|)$. In our case where $|p| = 256$, this leads to a proof of size 2680 bits, which is quickly amortized for sufficiently large revocation list SRL.

Complexity Comparison. We compare in Table 1 our construction with [8] and [22] on the most relevant metrics, namely the signature size and the computational cost for both the signer and the verifier. Obviously, this comparison has some limitations as the constructions do not target the same security model but the table at least shows that our scheme is the most efficient one even for quite small n. We note that [9] uses a different strategy for proving non-revocation but it is much less efficient than the one (based on [11]) used in [22] and [8] so there is no point in including this scheme in our table. To provide a fair comparison, we assume that the element B in the EPID signature from [8] is now generated as a hash output, which allows to remove this element from the signature and hence decreases the size of the latter. For the signature size, the bit size is provided by implementing the bilinear groups with the BLS12-381 curve used by ZCash [6]. It yields 256-bit elements of $\mathbb{Z}_p$ and 382-bit elements of $\mathbb{G}_1$. We follow the Fischlin's recommendations above for the parameters r and t related to the extractable proof π_2, namely $r = 10$ and $t = 12$.

Table 1 highlights the main difference between our work and previous ones. Whereas the latter have focused on improving the efficiency of the first, constant-size, part of the EPID signature, ours improves the latter at the cost of a slight overhead due to the use of online extractor. Our signature size grows roughly three times slower than those from [8] and [22], which means that our signature is the shortest one as soon as $n \geq 5$. Similarly, the computational cost of both Sign and Verify is dominated by $3n$ e_1 in our case, against $6n$ e_1 for previous works, meaning that our algorithms are roughly two times faster.

6 CONCLUSION

In this paper, we have proposed a new way for proving non-revocation in an EPID system which significantly improves performance. Concretely, we have replaced the zero-knowledge proof commonly used in previous works by a new proof implicitly based on a pseudo-random function, thereby reducing the number of secret scalars to hide. Although not fully generic, our techniques are designed to work with all existing EPID systems in cyclic groups and could then also be used to improve the latter.

Our strategy, which leads to a rather simple protocol, however gives rise to several issues when it comes to formally proving security. The second contribution of this paper is then to show how to deal with such issues while relying on common computational assumptions. In the end, our work thus shows that we can significantly improve efficiency of EPID systems without compromising security.

ACKNOWLEDGMENTS

The authors are grateful for the support of the ANR through project ANR-18-CE-39-0019-02 MobiS5.

REFERENCES

[1] Balthazar Bauer, Georg Fuchsbauer, and Julian Loss. 2020. A Classification of Computational Assumptions in the Algebraic Group Model. In *CRYPTO 2020, Part II (LNCS, Vol. 12171)*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer, Heidelberg, 121–151. https://doi.org/10.1007/978-3-030-56880-1_5
[2] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. 2003. Foundations of Group Signatures: Formal Definitions, Simplified Requirements, and a Construction Based on General Assumptions. In *EUROCRYPT 2003 (LNCS, Vol. 2656)*, Eli Biham (Ed.). Springer, Heidelberg, 614–629. https://doi.org/10.1007/3-540-39200-9_38
[3] Mihir Bellare, Haixia Shi, and Chong Zhang. 2005. Foundations of Group Signatures: The Case of Dynamic Groups. In *CT-RSA 2005 (LNCS, Vol. 3376)*, Alfred Menezes (Ed.). Springer, Heidelberg, 136–153. https://doi.org/10.1007/978-3-540-30574-3_11
[4] Dan Boneh and Xavier Boyen. 2008. Short Signatures Without Random Oracles and the SDH Assumption in Bilinear Groups. *Journal of Cryptology* 21, 2 (April 2008), 149–177. https://doi.org/10.1007/s00145-007-9005-7
[5] Dan Boneh and Hovav Shacham. 2004. Group Signatures With Verifier-Local Revocation. In *ACM CCS 2004*, Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick McDaniel (Eds.). ACM Press, 168–177. https://doi.org/10.1145/1030083.1030106
[6] Sean Bowe. 2017. BLS12-381: New zk-SNARK Elliptic Curve Construction. https://electriccoin.co/blog/new-snark-curve/.

[7] Ernie Brickell and Jiangtao Li. 2007. Enhanced privacy id: a direct anonymous attestation scheme with enhanced revocation capabilities. In *WPES 2007*.

[8] Ernie Brickell and Jiangtao Li. 2010. Enhanced Privacy ID from Bilinear Pairing for Hardware Authentication and Attestation. In *IEEE Conference on Social Computing, SocialCom*, Ahmed K. Elmagarmid and Divyakant Agrawal (Eds.).

[9] Ernie Brickell and Jiangtao Li. 2012. Enhanced Privacy ID: A Direct Anonymous Attestation Scheme with Enhanced Revocation Capabilities. *IEEE Trans. Dependable Secur. Comput.* (2012).

[10] Ernest F. Brickell, Jan Camenisch, and Liqun Chen. 2004. Direct Anonymous Attestation. In *ACM CCS 2004*, Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick McDaniel (Eds.). ACM Press, 132–145. https://doi.org/10.1145/1030083.1030103

[11] Jan Camenisch and Victor Shoup. 2003. Practical Verifiable Encryption and Decryption of Discrete Logarithms. In *CRYPTO 2003 (LNCS, Vol. 2729)*, Dan Boneh (Ed.). Springer, Heidelberg, 126–144. https://doi.org/10.1007/978-3-540-45146-4_8

[12] David Chaum and Eugène van Heyst. 1991. Group Signatures. In *EUROCRYPT'91 (LNCS, Vol. 547)*, Donald W. Davies (Ed.). Springer, Heidelberg, 257–265. https://doi.org/10.1007/3-540-46416-6_22

[13] Yevgeniy Dodis and Aleksandr Yampolskiy. 2005. A Verifiable Random Function with Short Proofs and Keys. In *PKC 2005 (LNCS, Vol. 3386)*, Serge Vaudenay (Ed.). Springer, Heidelberg, 416–431. https://doi.org/10.1007/978-3-540-30580-4_28

[14] Amos Fiat and Adi Shamir. 1987. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO'86 (LNCS, Vol. 263)*, Andrew M. Odlyzko (Ed.). Springer, Heidelberg, 186–194. https://doi.org/10.1007/3-540-47721-7_12

[15] Marc Fischlin. 2005. Communication-Efficient Non-interactive Proofs of Knowledge with Online Extractors. In *CRYPTO 2005 (LNCS, Vol. 3621)*, Victor Shoup (Ed.). Springer, Heidelberg, 152–168. https://doi.org/10.1007/11535218_10

[16] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. 2008. Pairings for cryptographers. *Discret. Appl. Math.* (2008).

[17] Intel. 2016. A Cost-Effective Foundation for End-to-End IoT Security, White Paper. https://www.intel.ca/content/www/ca/en/internet-of-things/white-papers/iot-identity-intel-epid-iot-security-white-paper.html.

[18] ISO/IEC. 2013. ISO/IEC 20008-2:2013 Information technology — Security techniques — Anonymous digital signatures — Part 2: Mechanisms using a group public key. https://www.iso.org/standard/56916.html.

[19] Nada El Kassem, Luís Fiolhais, Paulo Martins, Liqun Chen, and Leonel Sousa. 2019. A Lattice-Based Enhanced Privacy ID. In *WISTP 2019*.

[20] Helger Lipmaa. 2012. Progression-Free Sets and Sublinear Pairing-Based Non-Interactive Zero-Knowledge Arguments. In *TCC 2012 (LNCS, Vol. 7194)*, Ronald Cramer (Ed.). Springer, Heidelberg, 169–189. https://doi.org/10.1007/978-3-642-28914-9_10

[21] David Pointcheval and Olivier Sanders. 2016. Short Randomizable Signatures. In *CT-RSA 2016 (LNCS, Vol. 9610)*, Kazue Sako (Ed.). Springer, Heidelberg, 111–126. https://doi.org/10.1007/978-3-319-29485-8_7

[22] Olivier Sanders and Jacques Traoré. 2021. EPID with Malicious Revocation. In *CT-RSA 2021 (LNCS, Vol. 12704)*, Kenneth G. Paterson (Ed.). Springer, Heidelberg, 177–200. https://doi.org/10.1007/978-3-030-75539-3_8

[23] Claus-Peter Schnorr. 1990. Efficient Identification and Signatures for Smart Cards. In *CRYPTO'89 (LNCS, Vol. 435)*, Gilles Brassard (Ed.). Springer, Heidelberg, 239–252. https://doi.org/10.1007/0-387-34805-0_22