Subversion-Resilient Enhanced Privacy ID

Antonio Faonio[1(✉)], Dario Fiore[2], Luca Nizzardo[4], and Claudio Soriente[3]

[1] EURECOM, Sophia Antipolis, France
faonio@eurecom.fr
[2] IMDEA Software Institute, Madrid, Spain
dario.fiore@imdea.org
[3] NEC Labs Europe, Madrid, Spain
claudio.soriente@neclab.eu
[4] Protocol Labs, Madrid, Spain
luca@proto.ai

Abstract. Anonymous attestation for secure hardware platforms leverages tailored group signature schemes and assumes the hardware to be trusted. Yet, there is an increasing concern on the trustworthiness of hardware components and embedded systems. A subverted hardware may, for example, use its signatures to exfiltrate identifying information or even the signing key. We focus on Enhanced Privacy ID (EPID)—a popular anonymous attestation scheme used in commodity secure hardware platforms like Intel SGX. We define and instantiate a *subversion resilient* EPID scheme (or SR-EPID). In a nutshell, SR-EPID provides the same functionality and security guarantees of the original EPID, despite potentially subverted hardware. In our design, a "sanitizer" ensures no covert channel between the hardware and the outside world both during enrollment and during attestation (i.e., when signatures are produced). We design a practical SR-EPID scheme secure against adaptive corruptions and based on a novel combination of malleable NIZKs and hash functions modeled as random oracles. Our approach has a number of advantages over alternative designs. Namely, the sanitizer bears no secret information—hence, a memory leak does not erode security. Also, we keep the signing protocol non-interactive, thereby minimizing latency during signature generation.

1 Introduction

Anonymous attestation is a key feature of secure hardware platforms, such as Intel SGX[1] or the Trusted Computing Group's Trusted Platform Module[2]. It allows a verifier to authenticate a party as member of a trusted set, while keeping the party itself anonymous (within that set). This functionality is realized by

[1] https://www.intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions.html.
[2] https://trustedcomputinggroup.org/resource/tpm-library-specification/.

S. D. Galbraith (Ed.): CT-RSA 2022, LNCS 13161, pp. 562–588, 2022.
https://doi.org/10.1007/978-3-030-95312-6_23

using a privacy-enhanced flavor of group signatures in which signatures cannot be traced, not even by the group manager.

Given such realization paradigm, the security of anonymous attestation schemes is grounded on the trustworthiness of the signer. In particular, anonymity and unforgeability definitions assume that the signer is trusted and does not exfiltrate any information via its signatures. Yet, in most applications, the signer is a small piece of hardware with closed-source firmware (e.g., a smart card) to which a user has only black-box access. In such a scenario, trusting the hardware to behave honestly may be too strong of an assumption for at least two reasons. First, having only black-box access to a piece of hardware makes it virtually impossible to verify whether the hardware provides the claimed guarantees of security and privacy. Second, recent news on state-level adversaries corrupting security services[3] have shown that subverted hardware is a realistic threat. In the context of anonymous attestation, if the hardware gets subverted (e.g., via firmware bugs or backdoors), it may output valid, innocent-looking signatures that, in reality, covertly encode identifying information (e.g., using special nonces). Such signatures may allow a remote adversary to trace the signer, thereby breaking anonymity. Using a similar channel, a subverted signer could also exfiltrate its secret key, and this would enable an external adversary to frame an honest signer, for example by signing bogus messages on its behalf.

1.1 Our Contribution

We continue the study of subversion-resilient anonymous attestation and we focus on Enhanced Privacy ID (EPID) [9,10], an anonymous attestation scheme that is currently deployed on commodity trusted execution environments like Intel SGX. Our contribution is twofold: we first formalize the notion of *Subversion-Resilient EPID* (SR-EPID), and we propose a realization in bilinear groups.

The Model of Subversion-Resilient EPID. Enhanced Privacy ID is essentially a privacy-enhanced group signature where the group manager cannot trace a signature but signers can be revoked. In the context of remote attestation, a group member is instantiated by its signing component (the "signer"), which is typically a piece of hardware.

To counter subverted signers, our idea is to enhance the model by adding a "sanitizer" party whose goal is to ensure that no covert channel is established between a potentially subverted signer and external adversaries.[4] In practical application scenarios, the sanitizer could run on the same host of the signer (e.g., on a phone to sanitize signatures issued by the SIM card), or on a separate one (e.g., on a corporate firewall to sanitize signatures issued by local machines).

Compared to a subversion-resilient anonymous attestation scheme that uses split-signatures [11], our approach comes with multiple benefits. First, signature

[3] https://snowdenarchive.cjfe.org/.

[4] Adding a party between the potentially subverted signer and the outside world is necessary, as the signer could exfiltrate arbitrary information otherwise [11].

generation is non-interactive and the communication flow is unidirectional from the signer to the sanitizer, on to the verifier. Thus, our design decreases signing latency and provides more flexibility as the sanitization of a signature does not need to be done online. Another benefit of our design is the fact that the sanitizer holds no secret. This means that if a memory leak occurs on the sanitizer, one has nothing to recover but public information. Differently, in a split signature approach, security properties no longer hold if the TPM is subverted and the key share of its host is leaked.

The idea of adding a sanitizer to mitigate subversion attacks in anonymous attestation is inspired by that of using a cryptographic reverse firewall of Mironov and Stephens-Davidowitz [29]. Besides subversion-resilient unforgeability (as in Ateniese, Magri and Venturi [3]), in an EPID scheme we have to guarantee additional properties such as anonymity and non-frameability, as well as to deal with the complications of supporting revocation. Formalizing all these properties in rigorous definitions is a significant contribution of this paper.

We acknowledge that adding a sanitizer in this unidirectional communication channel also comes with the drawback that, when the signer is honest (i.e., not subverted), then for anonymity to hold we still need the sanitizer to be trusted. This is an inherent limitation of our model since the sanitizer is the last to speak in the protocol and can always establish a channel with the adversary[5]. Since our sanitizer does not hold a secret, a potential way to reduce trust on it can be to distribute its re-randomization procedure across multiple parties. Designing such a protocol does not seem straightforward as one should consider, for example, rushing adversaries and therefore it is left as future work.

As a byproduct of our new definitions of SR-EPID, we obtain a careful formalization of the notion of unforgeability for (non-subversion-resilient) EPID schemes, or more broadly, for group signature schemes with both key-revocation mechanisms and a blind join protocol. Compared to the previous definition of [10], ours formalizes several technical aspects that in [10] were essentially expressed only in words and left to the reader's interpretation. See the full version of this paper [20] for more details.

Our SR-EPID in Bilinear Groups. Our next contribution is an efficient construction of a SR-EPID based on bilinear pairings. Our starting point is the classical blueprint of group signature schemes where: (I) the group manager holds the secret key of a signature scheme; (II) during the join protocol the group manager creates a blind signature σ_y on a value y private to the prospective group member, and both σ and y are the group member's secret key; (III) a signature σ_M on message M is a signature of knowledge for M of a σ_y that verifies for y and the group public key. (IV) Finally, to support revocation and linkability, a signature σ_M is bound to an arbitrary basename B and contains a pseudorandom token $R_B = f_y(B)$. Without knowing y the token looks random (and thus hides the signer's identity) but, at the same time, can be efficiently checked

[5] This is not the case for the unforgeability and non-frameability properties for which we do consider the case of malicious sanitizers for honest signers.

against a revoked key y^*. That is, the verifier checks if $R_B = f_{y^*}(B)$ for all the y^* in the revocation list. Similarly, a signer that allows for linkability of its signatures may accept to produce multiple signatures on the same basename; such signatures could be easily linked as they carry the same token.

Our first idea to contrast subversion attacks is to let the sanitizer re-randomize every signature σ_M produced by the signer, sanitizing in this way the (possibly malicious) randomness chosen by the signer encodes a covert channel. We achieve this by employing re-randomizable NIZKs in step (III).

This is not the only possible attack vector between a subverted signer and an external adversary. For example, the signer may come with an hardcoded value y known to the external adversary so that all the (valid) signatures produced by the signer can be easily traced. To counter this class of attacks we let the sanitizer contribute with its randomness to the choice of y during the join protocol. Even further, we require the sanitizer to re-randomize any message sent to the group manager during the join protocol. Finally, another potential attack is that at any moment after the join protocol, the signer may switch to creating signatures by using a hardcoded secret $y', \sigma_{y'}$. As above, an external party equipped with y' could track those signatures. To contrast this class of attacks, we require the signer to produce, along with every signature σ_M, a proof π_σ that is verified by the sanitizer using a dedicated verification token and that ensures that the signer is using the same secret y used in the join protocol; if the check passes, the sanitizer strips off π_σ and returns a re-randomization of σ_M. Our model diverges from the cryptographic reverse firewall framework of [29] because of the verification token mechanism. Looking ahead, this is not simply a limitation of our scheme but more generally we can show that EPID schemes that admit secret-key based revocation cannot have a cryptographic reverse firewall, we give more details in Sect. 2.2.

The description above gives an high-level overview of the main ideas that we introduced in the protocol to counter subversion attacks. A significant technical contribution in the design of our construction is a set of techniques that we introduced to reconcile our extensive use of malleable NIZKs (and in particular, Groth-Sahai proofs [26]) with the goal of obtaining an efficient SR-EPID scheme. The main problem to prove security of our scheme is that we need the NIZK to be malleable and to have a flavor of simulation-extractable soundness.[6] In the EPID of [10], simulation-extractable soundness is also needed, but it is obtained for free by using Fiat-Shamir (Faust *et al.* [21]). In our case, this approach is not viable because the Fiat-Shamir compiler breaks any chance for re-randomizability. One could use a re-randomizable and (controlled) simulation-extractable NIZK (Chase *et al.* [16]), but in practice these tools are very expensive—they would require hundreds of pairings for verification and hundreds of group elements for the proofs. To overcome this problem, we propose a combination of (plain) GS proofs with the random oracle model. Briefly speaking, we use the random oracle

[6] In fact, on one hand we have to extract the witness from the adversary's forgery, while on the other hand we rely on zero-knowledge to disable any covert channel from subverted signers.

to generate the common reference string that will be used by the GS proof system and use the property that, in perfectly-hiding mode, this CRS can be created from a uniform random string[7]. In this way we can program the random oracle to produce extractable common reference strings for the forged signature made by the adversary and for the messages in the join protocol with corrupted members, and program the random oracle to have perfectly-hiding common reference strings for all the material that the reduction needs to simulate. Our technique is a reminiscence of techniques based on programmable hash functions [14] and linearly homomorphic signatures [27]. However, our ROM-based technique enables for more efficient schemes with *unbounded* simulation soundness. The resulting scheme provides the same functionality of EPID, tolerates subverted signers, and features signatures that are shorter than the ones in [10] for reasonable sizes of the revocation list: ours have $28 + 2n$ group elements whereas EPID signatures have $8+5n$, where n is the size of the revocation list (i.e., ours are shorter already for $n \geq 7$).

1.2 Related Work

Subversion-resilient signatures and Cryptographic Reverse Firewalls. Ateniese et al. [3] study subversion-resilient signature schemes and show that unique signatures as well as the use of a cryptographic reverse firewall (RF) of [29] ensure unforgeability despite a subverted signing algorithm. Chakraborty et al. [15] show how to use RF in multi-party settings where the adversary can fully corrupt all but one party, while the remaining parties are corrupt in a functionality-preserving way. Ganesh, Magri and Venturi [24] study the security properties of RF for the concrete case of interactive proof systems. Chen et al. [17] introduce malleable smooth projective hash functions and show how to use them in a modular way to construct RF for several cryptographic protocols.

Our scheme could be interpreted as a new EPID scheme equipped with a cryptographic reverse firewall for the *join protocol* that allows a new party to join the group, and a cryptograhic reverse firewall that protects the signatures sent by the signer. However, as mentioned in Sect. 1.1, there are some technical details that differentiate our model to the cryptographic reverse firewall framework.

Subversion-resilient anonymous attestation. Camenisch et al. [12] provide a Universally Composable (UC) definition for Direct Anonymous Attestation (DAA) that guarantees privacy despite a subverted TPM. The DAA scheme presented in [12] leverages dual-mode signatures of Camenisch and Lehmann [13] and builds upon the ideas of Bellare and Sandhu [7] to provide a signature scheme where the signing key is split between the host and the TPM. Later on, Camenisch *et al.* [11] build on the same idea of [12] and show a UC-secure DAA scheme that requires only minor changes to the TPM 2.0 interface and tolerates a subverted TPM by splitting the signing key between the host and the TPM.

[7] In particular, we need cryptographic hash functions that allow to hash directly on $\mathbb{G}_1$ and on $\mathbb{G}_2$, see Galbraith *et al.* [23].

We argue that splitting the signing key between the potentially subverted hardware (e.g., the TPM) and the host to achieve resilience to subversions is viable in scenarios where (i) the channel between the two parties has low latency—because of the interactive nature of the signing protocol—and (ii) the user can trust the host. Both conditions holds for TPM scenarios. In particular, a TPM is soldered to the motherboard of the host and has a high-speed bus to the main processor. Also, the TPM manufacturer is usually different from the one of the main processor—hence, the user may trust the latter but not the former. In case of TEEs such as Intel SGX, we note that there is no real separation between the TEE and the main processor. Thus, it would be hard to justify an untrusted TEE and a trusted processor since, in reality, they lie on the same die and are shipped by the same manufacturer. As such, the entity in charge of preventing the TEE from exfiltrating information (i.e., the one holding a share of the signing key) must be placed elsewhere along the channel between the TEE and the verifier, thereby paying a latency penalty to generate signatures.

We think that our solution is more suitable for TEE platforms like Intel SGX. In particular, the non-interactive nature of the signing protocol allows us to place the sanitizer "away" from the signer, without impact on performance. Thus, the sanitizer may be instantiated by a co-processor next to the TEE, or it may run on a company gateway that sanitizes attestations produced by hosts within the company network before they are sent out. As the sanitizer and the potentially subverted hardware may run on different platforms, they may come from different manufacturers. For example, one could pick an AMD or Risc-V processor to sanitize an Intel-based TEE such as SGX. A sanitizer may even be built by combining different COTS hardware as [28].

Other models for subversion resilience. Fischlin and Mazaheri introduce "self-guarding" cryptographic protocols [22] based on the assumption of a secure initial phase where the algorithm was genuine. Kleptograpic attacks, introduced by Young and Yung [32], assume subverted implementations of standard cryptographic primitives. Later on, Bellare et al. [6] and Russell et al. [30] studied subverted symmetric encryption schemes and subverted key-generation routines, respectively. Russell et al. [31] propose for the first time an IND-CPA-secure encryption scheme that remains secure even in case of a subversion-capable adversary. Ateniese et al. [2] introduce an "immunizer" that takes as input a cryptographic primitive and augments it with subversion resilience. They introduce an immunizer in the plain model for a number of deterministic primitives (with a randomized key generation routine). Chow et al. [18], construct secure digital signature schemes in the presence of kleptographic attacks, by leveraging an offline phase to test the potentially subverted implementation in a black-box manner.

2 Subversion-Resilient Enhanced Privacy ID

Background on EPID. Enhanced Privacy ID is essentially a privacy-enhanced group signature scheme. Compared to classic group signatures (see Bellare *et al.* [5]), EPID drops the ability of the group manager to trace signatures, and adds novel revocation mechanisms. In particular, EPID allows to revoke a group member by adding its private key to a revocation list named PrivRL; while verifying a signature σ, the verification algorithm checks that none of the private keys in PrivRL may have produced σ. In case the secret key of a misbehaving group member did not leak, EPID can still revoke that member by using one of its signatures. That is, EPID accounts for an additional revocation list, named SigRL, containing signatures of revoked members.

Security notions for EPID include anonymity and unforgeability. Informally, anonymity ensures that signatures are not traceable by any party, including the group manager. Unforgeability ensures that only non-revoked group members can generate valid signatures. We note that EPID does not account for pseudonymous signatures. The latter allow for a sort of controlled linkability as each signature is bound to a "basename", and one can easily tell—via a Link algorithm— whether two signatures on the same basename where produced by the same group member. This signature mode is actually available in DAA and in the version of EPID used by Intel SGX. Further, DAA defines a security property tailored to pseudonymous signatures called *non-frameability.* Informally, non-frameability ensures that an adversary that corrupts the group manager cannot create a signature on a message M and basename bsn, that links to a signature of an honest group member. Given the usefulness of pseudonymous signatures in real-world deployments, we decide to include them—along with a definition of non-frameability—in our definition of subversion-resilient EPID. The study of subversion-resilient non-frameability can be found in the full version [20].

2.1 Subversion-Resilient EPID

For simplicity, we assume each signer to be paired with a sanitizer and we denote a pair of signer-sanitizer as a "platform".[8] In the security experiments we denote with $\mathcal{I}$ the issuer, with $\mathcal{S}$ the sanitizer, with $\mathcal{M}$ the signer, and with $\mathcal{P}$ the platform. Very often we refer to the signer as the "hardware" or the "machine" (thus the letter $\mathcal{M}$ for our notation). We assume group members to be platforms and gear security definition towards them.[9] Our notion of SR-EPID is designed so that (i) the sanitizer participates to the join protocol obtaining a verification token as output, and (ii) the sanitizer sanitizes signatures to avoid covert channel based on maliciously-sampled randomness using such verification token. The resulting syntax is a generalization of EPID that adds a $\mathsf{Sanitize}$ algorithm and modifies the original Join and Sig algorithms.

Syntax of Subversion-Resilient EPID. We denote by $\langle d, e, f\rangle \leftarrow P_{\mathcal{A},\mathcal{B},\mathcal{C}}\langle a, b, c\rangle$ an interactive protocol P between parties $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ where a, b, c (resp. d, e, f) are

[8] In practical deployments a sanitizer may sanitize signatures of multiple signers and a single signer may have multiple sanitizers.

[9] For example, the anonymity definition focuses on an adversary that must tell which, out of two platforms, outputs the challenge signature.

the local inputs (resp. outputs) of $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$, respectively. An SR-EPID consists of an interactive protocol Join and algorithms: Init, Setup, Sig, Ver, $\mathsf{Sanitize}$. All the algorithms (and the protocol) but Init take as input public parameters (generated by Init); for readability reasons, we keep this input implicit.

$\mathsf{Init}(1^\lambda) \rightarrow \mathsf{pub}$. This algorithm takes as input the security parameter λ and outputs public parameters pub.

$\mathsf{Setup}(\mathsf{pub}) \rightarrow (\mathsf{gpk}, \mathsf{isk})$. This algorithm takes the public parameters pub and outputs a group public key gpk and an issuing secret key isk for the issuer $\mathcal{I}$.

$\mathsf{Join}_{\mathcal{I},\mathcal{S}_i,\mathcal{M}_i}\langle(\mathsf{gpk}, \mathsf{isk}), \mathsf{gpk}, \mathsf{gpk})\rangle \rightarrow \langle b, (b, \mathsf{svt}_i), \mathsf{sk}_i\rangle$. This is a protocol between the issuer $\mathcal{I}$, a sanitizer $\mathcal{S}_i$ and a signer $\mathcal{M}_i$. The issuer inputs $(\mathsf{gpk}, \mathsf{isk})$, while the other parties only input gpk. At the end of the protocol, $\mathcal{I}$ obtains a bit b indicating if the protocol terminated successfully, $\mathcal{M}_i$ obtains private key sk_i, and $\mathcal{S}_i$ obtains a sanitizer verification token svt_i and the same bit b of $\mathcal{I}$.

$\mathsf{Sig}(\mathsf{gpk}, \mathsf{sk}_i, \mathsf{bsn}, M, \mathsf{SigRL}) \rightarrow \perp/(\sigma, \pi_\sigma)$. The signing algorithm takes as input gpk, sk_i, a basename bsn, a message M, and a signature based revocation list SigRL. It outputs a signature σ and a proof π_σ, or an error $\perp$.

$\mathsf{Ver}(\mathsf{gpk}, \mathsf{bsn}, M, \sigma, \mathsf{SigRL}, \mathsf{PrivRL}) \rightarrow 0/1$. The verification algorithm takes as input gpk, bsn, M, σ, a signature based revocation list SigRL, and a private key based revocation list PrivRL. It outputs a bit.

$\mathsf{Sanitize}(\mathsf{gpk}, \mathsf{bsn}, M, (\sigma, \pi_\sigma), \mathsf{SigRL}, \mathsf{svt}_i) \rightarrow \perp/\sigma'$. The sanitization algorithm takes as input gpk, a basename bsn, a message M, a signature σ with corresponding proof π_σ, a signature based revocation list SigRL, and a sanitizer verification token svt_i. It outputs either $\perp$ or a sanitized signature σ'.

$\mathsf{Link}(\mathsf{gpk}, \mathsf{bsn}, M_1, \sigma_1, M_2, \sigma_2) \rightarrow 0/1$. The linking algorithm takes as input gpk, a basename bsn, and two tuples M_1, σ_1 and M_2, σ_2 and output a bit.

The last algorithm outputs 1 if the two signatures were generated by the same signer using the same basename (we call such property *linking correctness*).

In our syntax, we assume PrivRL to be a set of private keys $\{\mathsf{sk}_i\}_i$, and SigRL to be a set of triples $\{(\mathsf{bsn}_i, M_i, \sigma_i)\}_i$, each consisting of a basename, a message and a signature. We define two forms of correctness. To keep the syntax more light we let $\mathsf{Sig}(\mathsf{gpk}, \mathsf{sk}, \mathsf{bsn}, M)$ be equal to $\mathsf{Sig}(\mathsf{gpk}, \mathsf{sk}, \mathsf{bsn}, M, \emptyset)$, and $\mathsf{Ver}(\mathsf{gpk}, \mathsf{bsn}, M, \sigma)$ be equal to $\mathsf{Ver}(\mathsf{gpk}, \mathsf{bsn}, M, \sigma, \emptyset, \emptyset)$.

Definition 1 (Correctness, without revocation lists). *We say that an SR-EPID scheme satisfies standard correctness if for any* $\mathsf{pub} \leftarrow \mathsf{Init}(1^\lambda)$*, any* $(\mathsf{gpk}, \mathsf{gsk}) \leftarrow \mathsf{Setup}(\mathsf{pub})$*, any* $\langle b, (b, \mathsf{svt}), \mathsf{sk}\rangle \leftarrow \mathsf{Join}\langle(\mathsf{gpk}, \mathsf{gsk}), \mathsf{gpk}, \mathsf{gpk})\rangle$ *such that* $b = 1$*, and for any* bsn*,* M*,* $(\sigma, \pi_\sigma) \leftarrow \mathsf{Sig}(\mathsf{gpk}, \mathsf{sk}, \mathsf{bsn}, M)$*, and any* $\sigma' \leftarrow \mathsf{Sanitize}(\mathsf{gpk}, \mathsf{bsn}, M, (\sigma, \pi_\sigma), \mathsf{svt})$ *we have that both* $\mathsf{Ver}(\mathsf{gpk}, \mathsf{bsn}, M, \sigma) = 1$ *and* $\mathsf{Ver}(\mathsf{gpk}, \mathsf{bsn}, M, \sigma') = 1$*.*

Let $\Sigma_{\mathsf{gpk},\mathsf{sk}}$ be the set of triples for any basename, message and signature where the signature algorithm can produce the signature with input gpk and the secret key sk (thus enumerating over all possible basenames and messages).

Definition 2 (Correctness, with revocation lists). *We say that an SR-EPID scheme satisfies correctness if for any* $\mathsf{pub} \leftarrow \mathsf{Init}(1^\lambda)$*, any* $(\mathsf{gpk}, \mathsf{gsk}) \leftarrow \mathsf{Setup}(\mathsf{pub})$*, any* $\langle b, (b, \mathsf{svt}), \mathsf{sk}\rangle \leftarrow \mathsf{Join}\langle(\mathsf{gpk}, \mathsf{gsk}), \mathsf{gpk}, \mathsf{gpk})\rangle$ *such that* $b = 1$*, and for any* bsn, M*, for any* PrivRL *and* SigRL*, any* $(\sigma_0, \pi_\sigma) \leftarrow \mathsf{Sig}(\mathsf{gpk}, \mathsf{sk}, \mathsf{bsn}, M, \mathsf{SigRL})$ *and any* $\sigma_1 \leftarrow \mathsf{Sanitize}(\mathsf{gpk}, \mathsf{bsn}, M, (\sigma_0, \pi_\sigma), \mathsf{SigRL}, \mathsf{svt})$ *we have:*

$$\Sigma_{\mathsf{gpk},\mathsf{sk}} \cap \mathsf{SigRL} = \emptyset \Rightarrow \sigma_0 \neq \bot \tag{1}$$

$$\mathsf{sk} \notin \mathsf{PrivRL} \Rightarrow \forall b' \in \{0,1\} : \mathsf{Ver}(\mathsf{gpk}, \mathsf{bsn}, M, \sigma_{b'}, \mathsf{SigRL}, \mathsf{PrivRL}) = 1 \tag{2}$$

2.2 Subversion-Resilient Security

The security of an SR-EPID scheme is defined by three main properties, namely anonymity, unforgeability, and non-frameability. We consider subverted signers that can arbitrarily behave during the join protocol and, in particular, abort the execution of the protocol. However, once the join protocol is completed we assume that signers, although subverted, maintain a correct "input-output behavior". That is, a subverted signer produces a valid signature to a message and basename, namely a signature that verifies if the signer were not revoked, but that could be arbitrarily (and maliciously) distributed over the set of all valid signatures. We formalize this idea in the following assumption.

Assumption 1. Let Π be a SR-EPID. We assume that for any public parameter pub, any adversary $\mathcal{A}$, any gpk and auxiliary information aux, and any (possibly adaptively chosen) sequence of tuples $(\mathsf{bsn}_1, M_1), \ldots, (\mathsf{bsn}_q, M_q)$, let $\langle b, (b', \mathsf{svt}), \mathsf{state}_1\rangle$ be a possible output of the join protocol $\mathsf{Join}_{\mathcal{A},\mathcal{S},\mathcal{M}}\langle(\mathsf{gpk}, aux), \mathsf{gpk}, (\mathsf{gpk}, aux)\rangle$ conditioned on $b' = 1$ or a possible output of the join protocol $\mathsf{Join}_{\mathcal{I},\mathcal{A},\mathcal{M}}\langle(\mathsf{gpk}, aux), \mathsf{gpk}, (\mathsf{gpk}, aux)\rangle$ conditioned on $b = 1$ and let $\sigma_i, \mathsf{state}_i \leftarrow \mathcal{M}_i(\mathsf{state}_{i-1}, M_i, \mathsf{bsn}_i)$ for $i = 1, \ldots, q$ then $\forall i \in [q] : \mathsf{Ver}(\mathsf{gpk}, M_1, \mathsf{bsn}_1, \sigma_i) = 1$.

The assumption models the fact that, if signers can be subverted, a signer should be considered safe as long as it does not return errors when it comes to generating signatures. The occurrence of such an error should alert a sanitizer anyway. First, such an error can occur if one of the signatures produced by the signer was included in the signature based revocation list: if the list was honestly created, it means that the signer has been revoked; if the list was maliciously crafted, then the signature request may constitute an attempt to deanonymize the signer. Second, if the errors are arbitrary then they inevitably enable to signal any kind of information from the signer.

Macros for the Join Protocol and Signature generation. As mentioned, the join protocol is a three-party protocol with the sanitizer being in the middle. To simplify the already heavy notation, we define the macro $\mathsf{Join}(\mathcal{M}, \mathsf{state}_\mathcal{S}, \mathsf{state}_\mathcal{M}, \gamma_\mathcal{I})$ which identifies one full round of the join protocol from the issuer point of view with an honest sanitizer and a machine $\mathcal{M}$. In more detail, the macro takes as input the description of the (possibly subverted) machine $\mathcal{M}$, the state of the sanitizer $\mathsf{state}_{\mathcal{S}}$, the state of the machine $\mathsf{state}_{\mathcal{M}}$ and the message sent by the issuer $\gamma_{\mathcal{I}}$, and it identifies the following set of actions:

```
$\underline{\mathsf{Join}(\mathcal{M}, \mathsf{state}_{\mathcal{S}}, \mathsf{state}_{\mathcal{M}}, \gamma_{\mathcal{I}})}$:
 1. $(\gamma'_{\mathcal{S}}, \mathsf{state}'_{\mathcal{S}}) \leftarrow \mathcal{S}.\mathsf{Join}(\mathsf{gpk}, \mathsf{state}_{\mathcal{S}}, \gamma_{\mathcal{I}})$;
 2. $(\gamma_{\mathcal{M}}, \mathsf{state}'_{\mathcal{M}}) \leftarrow \mathcal{M}.\mathsf{Join}(\mathsf{gpk}, \mathsf{state}_{\mathcal{M}}, \gamma'_{\mathcal{S}})$;
 3. $(\gamma_{\mathcal{S}}, \mathsf{state}''_{\mathcal{S}}) \leftarrow \mathcal{S}.\mathsf{Join}(\mathsf{gpk}, \mathsf{state}'_{\mathcal{M}}, \gamma_{\mathcal{M}})$;
 4. Output $(\mathsf{state}''_{\mathcal{S}}, \mathsf{state}'_{\mathcal{M}}, \gamma_{\mathcal{S}})$.
```

Notice the procedures additionally take as input the group public key gpk, which we keep implicit. Similarly, the signature procedure is a two-phase protocol between the signer and the sanitizer for which we define the macro:

```
$\underline{\mathsf{Sig}(\mathcal{M}, \mathsf{state}_{\mathcal{M}}, \mathsf{svt}, \mathsf{bsn}, M, \mathsf{SigRL})}$:
 1. $(\sigma', \pi'_{\sigma}, \mathsf{state}'_{\mathcal{M}}) \leftarrow \mathcal{M}.\mathsf{Sig}(\mathsf{state}_{\mathcal{M}}, \mathsf{bsn}, M, \mathsf{SigRL})$;
 2. if $\mathsf{svt} \neq \perp$ then $\sigma \leftarrow \mathsf{Sanitize}(\mathsf{gpk}, \mathsf{bsn}, M^*, (\sigma', \pi'_{\sigma}), \mathsf{SigRL}, \mathsf{svt})$;
 3. else $\sigma \leftarrow \sigma'$;
 4. Output $(\mathsf{state}'_{\mathcal{M}}, \sigma)$.
```

The macro additionally checks in step 2 that svt is a valid string. We use this check to discriminate the case when the sanitizer is corrupted.

Subversion-Resilient Anonymity. This notion formalizes the idea that an adversarial issuer cannot identify a group member through the signatures it produces. Recall that we assume a signer $\mathcal{M}_i$ to be paired with a sanitizer $\mathcal{S}_i$; we denote the platform constituted by $\mathcal{M}_i$ and $\mathcal{S}_i$ with $\mathcal{P}_i$. We assume $\mathcal{M}_i$ to be subverted, i.e., it runs an adversarially specified program, while $\mathcal{S}_i$ is honest. The case when both $\mathcal{M}_i$ and $\mathcal{S}_i$ are corrupted is meaningless for anonymity since the adversary controls all the relevant parties. The remaining case in which $\mathcal{M}_i$ is honest but $\mathcal{S}_i$ is corrupted is also hopeless for anonymity since a corrupted sanitizer could always maul the outputs of the signer in order to reveal its identity.

We formalize subversion-resilient anonymity for SR-EPID in a security experiment that appears in Fig. 1, and we formally define anonymity as follows.

Definition 3. *Consider the experiment described in Fig. 1. We say that an SR-EPID Π is anonymous if and only if for any PPT adversary $\mathcal{A}$:*

$$\mathbf{Adv}^{\mathsf{anon}}_{\mathcal{A},\Pi}(\lambda) := \left| \Pr\left[\mathbf{Exp}^{\mathsf{anon}}_{\mathcal{A},\Pi}(\lambda, 0) = 1\right] - \Pr\left[\mathbf{Exp}^{\mathsf{anon}}_{\mathcal{A},\Pi}(\lambda, 1) = 1\right] \right| \in \mathsf{negl}(\lambda).$$

Here we provide an intuitive explanation of the anonymity experiment. The idea is that the adversary plays the role of the issuer, i.e., it selects the group public key, and it can do the following: (1) ask platforms with subverted signers to join the system; (2) ask platforms with subverted signers to sign messages; (3) corrupt platforms. For (1), it means that the adversary specifies the code of a signer $\mathcal{M}_i$ that, together with an honest sanitizer $\mathcal{S}_i$, run the Join protocol with the adversary playing the role of the issuer. For (2), a subverted signer $\mathcal{M}_i$ produced a signature that is sanitized by $\mathcal{S}_i$ and then delivered to the adversary. Finally, (3) models a full corruption of the platform in which the adversary learns the secret key sk_i obtained by $\mathcal{M}_i$ at the end of its Join protocol.

The adversary can choose two platforms $(\mathcal{P}_{i_0}, \mathcal{P}_{i_1})$, a basename bsn^*, and a message M^* and it receives a *sanitized* signature on M^*, bsn^* produced by one of the two platforms. The goal of the adversary is to figure out which platform produced the signature. In order to avoid trivial attacks the two "challenge" platforms must be non-corrupted and none of their signatures can be included in the SigRL used to produce the challenge signature. Further, if the adversary has previously requested a signature with bsn^* form either platform, the challenger aborts. Similarly, after seeing the challenge signature, the adversary may not ask for a signature by any of the challenge platforms on basename bsn^*.

Technical details. The structure is the one depicted earlier: the adversary chooses the group public key on input the public parameters and then starts interacting with the oracle $\mathcal{C}$. The experiment maintains lists $L_{join}, L_{usr}, L_{corr}$ to bookkeep information on the state of the Join protocol sessions, and the list of non-corrupted and corrupted platforms, respectively. Also, it maintains a flag Bad, initialized to false, which is turned to true whenever the adversary violates the rules of the experiments (see below). At some point the adversary outputs a message M^*, a basename bsn^*, and two indices i_0, i_1, along with a signature revocation list SigRL^*; it receives a sanitized signature generated using the subverted signer $\mathcal{M}_{i_b}$. In line 8 of $\mathbf{Exp}^{\mathsf{anon}}_{\mathcal{A},\Pi}(\lambda, b)$ we ensure that the adversary did not previously query for a signature with basename bsn^* by one of the challenge platforms; if that is the case, the adversary could trivially win by using the Link algorithm. In line 11 of $\mathbf{Exp}^{\mathsf{anon}}_{\mathcal{A},\Pi}(\lambda, b)$ we ensure that both challenge platforms generate valid signatures, after sanitization. Indeed if a difference would occur (e.g., one of them is $\perp$), the adversary could trivially win the game. For example, this would be the case if the SigRL chosen by $\mathcal{A}$ would contain a signature from, e.g., $\mathcal{M}_{i_0}$. Similar checks are done in lines 16–20 of the $\mathcal{C}$ oracle upon a signing query that involves one of the challenge platforms, say $i_{1-\beta}$. The code of those lines essentially ensure that the queried basename is not the challenge one, and that the other challenge platform i_β would generate a signature on the same message M that is valid iff so is the one generated by $i_{1-\beta}$. Again if such a difference would occur the adversary could trivially distinguish and win the experiment. Similarly to the other case, this could occur if the queried SigRL contains a signature of (only) one of the challenge platforms.

We stress that the mechanism that uses the verification tokens is necessary[10]. Indeed, consider the definition above where the svt and the proof π_σ are missing: An attacker can first performs two join protocols one with a subverted machine $\tilde{\mathcal{M}}$ with hardcoded a secret key $\tilde{\mathsf{sk}}$ that during joining time acts honestly, thus

[10] Here is where our model diverges from the cryptographic reverse firewall framework of [29].

```
Experiment $\mathbf{Exp}^{\mathsf{anon}}_{\mathcal{A},\Pi}(\lambda, b)$

1 :  $L_{join}, L_{usr}, L_{corr} \leftarrow \emptyset$; $post \leftarrow 0$; $\mathsf{Bad} \leftarrow \mathsf{true}$
2 :  $\mathsf{pub} \leftarrow \mathsf{Init}(1^{\lambda})$; $\mathsf{gpk} \leftarrow \mathcal{A}(\mathsf{pub})$;
3 :  $(\mathsf{bsn}^*, M^*, i_0, i_1, \mathsf{SigRL}^*) \leftarrow \mathcal{A}(\mathsf{gpk})^{\mathcal{C}(\mathsf{gpk},\cdot)}$; $post \leftarrow 1$;
4 :  if $(i_0, *, *, *, *) \notin L_{usr} \vee (i_1, *, *, *) \notin L_{usr}$ then $\mathsf{Bad} \leftarrow \mathsf{true}$;
5 :  if $\mathsf{VerSigRL}(\mathsf{gpk}, \mathsf{SigRL}^*) = 0$ then $\mathsf{Bad} \leftarrow \mathsf{true}$
6 :  for $j = 0, 1$ do :
7 :    Retrieve $(i_j, \mathcal{M}_{i_j}, \mathsf{state}_{i_j}, \mathsf{svt}_{i_j}, B_{i_j})$ from $L_{usr}$;
8 :    if $\mathsf{bsn}^* \in B_{i_j}$ then $\mathsf{Bad} \leftarrow \mathsf{true}$
9 :    $(\mathsf{state}'_{i_j}, \sigma_j) \leftarrow \mathsf{Sig}(\mathcal{M}_{i_j}, \mathsf{state}_{i_j}, \mathsf{svt}_{i_j}, \mathsf{bsn}, M, \mathsf{SigRL})$;
10 :    Update $(i_j, \mathcal{M}_{i_j}, \mathsf{state}'_{i_j}, \mathsf{svt}_{i_j}, B_{i_j} \cup \{\mathsf{bsn}^*\})$ in $L_{usr}$;
11 :  if $\perp \in \{\sigma_0, \sigma_1\}$ then $\mathsf{Bad} \leftarrow \mathsf{true}$ else $\sigma^* \leftarrow \sigma_b$;
12 :  $b' \leftarrow \mathcal{A}(\sigma^*)^{\mathcal{C}(\mathsf{gpk},\cdot)}$;
13 :  if $\mathsf{Bad} = \mathsf{false}$ return $b'$; else return $\tilde{b} \leftarrow_{\$} \{0, 1\}$.

Oracle $\mathcal{C}(\mathsf{gsk}, \cdot)$

1 :  Upon query $(\mathtt{join}, i, \gamma_{\mathcal{I}})$ :
2 :    Retrieve $(i, \mathcal{M}_i, \mathsf{state}_{\mathcal{S}}, \mathsf{state}_{\mathcal{M}})$ from $L_{join}$, ;
3 :    If not find parse $\gamma_{\mathcal{I}} = \mathcal{M}_i$ and add $(i, \mathcal{M}_i, \perp, \perp)$ in $L_{join}$ and return ;
4 :    $(\mathsf{state}''_{\mathcal{S}}, \mathsf{state}'_{\mathcal{M}}, \gamma_{\mathcal{S}}) \leftarrow \mathsf{Join}(\mathcal{M}_i, \mathsf{state}_{\mathcal{S}}, \mathsf{state}_{\mathcal{M}}, \gamma_{\mathcal{I}})$;
5 :    Store $(i, \mathcal{M}_i, \mathsf{state}''_{\mathcal{S}}, \mathsf{state}'_{\mathcal{M}})$ in $L_{join}$;
6 :    if $\gamma_{\mathcal{S}} = \mathtt{concluded}$ then
7 :      $\mathsf{svt}_i \leftarrow \mathsf{state}''_{\mathcal{S}}$, store $(i, \mathcal{M}_i, \mathsf{state}'_{\mathcal{M}}, \mathsf{svt}_i, \emptyset)$ in $L_{usr}$; return $(\gamma_{\mathcal{S}}, \mathsf{svt}_i)$;
8 :    else return $\gamma_{\mathcal{S}}$.
9 :  Upon query $(\mathtt{sign}, i, \mathsf{bsn}, M, \mathsf{SigRL})$ :
10 :    if $(i, *, *, *, *) \notin L_{usr}$ then $\mathsf{Bad} \leftarrow \mathsf{true}$;
11 :    else retrieve $(i, \mathcal{M}_i, \mathsf{state}_i, \mathsf{svt}_i, B_i) \in L_{usr}$;
12 :    $(\mathsf{state}'_i, \sigma) \leftarrow \mathsf{Sig}(\mathcal{M}_i, \mathsf{state}_i, \mathsf{svt}_i, \mathsf{bsn}, M, \mathsf{SigRL})$;
13 :    Update $(i, \mathcal{M}_i, \mathsf{state}'_i, \mathsf{svt}_i, B_i \cup \{\mathsf{bsn}\})$;
14 :    if $post = 0$ or $i \notin \{i_0, i_1\}$ then return $\sigma$
15 :    else let $i = i_\beta$ and $\beta \in \{0, 1\}$;
16 :    if $\mathsf{bsn} = \mathsf{bsn}^*$ then $\mathsf{Bad} \leftarrow \mathsf{true}$;
17 :    Let $i = i_{1-\beta}$, retrieve $(i_\beta, \mathcal{M}_{i_\beta}, \mathsf{state}_{i_\beta}, \mathsf{svt}_{i_\beta}, B_{i_\beta}) \in L_{usr}$;
18 :    $(\mathsf{state}'_{i_\beta}, \tilde{\sigma}) \leftarrow \mathsf{Sig}(\mathcal{M}_{i_\beta}, \mathsf{state}_{i_\beta}, \mathsf{svt}_{i_\beta}, \mathsf{bsn}, M, \mathsf{SigRL})$;
19 :    Update $(i_\beta, \mathcal{M}_{i_\beta}, \mathsf{state}'_{i_\beta}, \mathsf{svt}_{i_\beta}, B_{i_\beta} \cup \{\mathsf{bsn}\})$ in $L_{usr}$;
20 :    if $\perp \in \{\sigma, \tilde{\sigma}\}$ then $\mathsf{Bad} \leftarrow \mathsf{true}$; endif return $\sigma$;
21 :  Upon query $(\mathtt{corrupt}, i)$ :
22 :    if $post = 1 \wedge i \in \{i_0, i_1\}$ then $\mathsf{Bad} \leftarrow \mathsf{true}$;
23 :    else retrieve $(i, \mathcal{M}_i, \mathsf{state}_i, \mathsf{svt}_i)$ from $L_{usr}$;
24 :      move the tuple from $L_{usr}$ to $L_{cor}$;
25 :      return $(\mathsf{state}_i, \mathsf{svt}_i)$.
```

Fig. 1. Subversion-resilient anonymity experiment.

obtaining a new fresh secret key, but that computes valid signature using the hardcoded secret key. Suppose the scheme has a secret-key based revocation mechanism, then the adversary that knows $\tilde{\mathsf{sk}}$ can easily distinguish the subverted machine from an honest machine. In particular, it could verify the challenge signature using the revocation list $\{\tilde{\mathsf{sk}}\}$. The sanitizer, which only posses public information, has no way to identify that a different secret key has been used and avoid this attack. We formalize the above intuition in the full version of this paper. Another aspect of the anonymity experiment that we would like to point out is that the adversary receives the verification token immediately after the Join protocol is over. This models the fact the adversary could have access to the internal state of an honest sanitizer (except for its random tape), and this does not break anonymity.

Subversion-Resilient Unforgeability. This notion formalizes the idea that an adversary who does not control the issuer cannot generate signatures on new messages on behalf of non-corrupted platforms. To model subversion attacks, we let the platform signer $\mathcal{M}_i$ be an adversarially specified program. The sanitizer $\mathcal{S}_i$ is instead honest (unless the platform is fully corrupted).

Here we provide an intuition of the notion. The adversary receives the group public key, and it can do the following: (1) ask platforms with subverted signers to join the system; (2) ask corrupted platforms to join the system; (3) ask platforms with subverted signer to sign messages; (4) corrupt platforms. For (1), it means that the adversary specifies the code of a signer $\mathcal{M}_i$ and that signer together with sanitizer $\mathcal{S}_i$, run the Join protocol where both the issuer and $\mathcal{S}_i$ are controlled by the challenger. For (2), the adversary runs the Join protocol with the challenger playing the role of the issuer, whereas both the signer $\mathcal{M}_i$ and the sanitizer $\mathcal{S}_i$ are fully controlled by the adversary. For (3), the adversary asks a platform that joined the system to create a signature using the subverted signing algorithm (specified in $\mathcal{M}_i$ at Join time), this signature is sanitized by $\mathcal{S}_i$ and given to the adversary. Finally, (4) simply models a full corruption of the platform in which the adversary learns the secret key sk_i obtained by $\mathcal{M}_i$ at the end of its Join protocol[11].

The adversary's goal is to produce a valid signature on a basename-message tuple bsn^*, M^*. On the one hand, we cannot require the tuple bsn^*, M^* to be fresh, since it is reasonable to assume that multiple platforms may sign the same bsn^*, M^*. On the other hand, strong unforgeability is impossible, as we require that the signatures must be valid before and after sanitization. To satisfy these two apparently contrasting requirements simultaneously, we instead require that the adversary's forgery does not link to any of the other queried signatures on the same basename-message tuple. This essentially guarantees that the forgery is not a trivial rerandomization of signature obtained through a signing query.

[11] Here the corruption is *adaptive* in the sense that the platform first joined honestly and later can be corrupted by the adversary but we assume *secure erasure* of the previous states of the sanitizer.

Since an SR-EPID is a (kind of) group signature and in the above game the adversary may have learnt the secret keys of some group members, we add some additional checks to formalize what is a forgery, so to avoid trivial unavoidable attacks. Intuitively, we want that the signature must verify with respect to a private-key revocation list PrivRL^* (resp. signature-based revocation list SigRL^*) that includes the secret keys of (resp. a signature from) all corrupted group members. These corrupted group members include both the ones that honestly joined the system and were later corrupted, and those that were already corrupted (i.e., adversarially controlled) at join time. Modeling which keys should be revoked is not straightforward though. The first issue is that in case of a corrupted platform joining the group, the challenger does not know what is the key obtained by the adversary. Essentially, unless we revoke exactly that key or a signature produced with that key, the adversary is able to create valid signatures on any message of its choice. The second issue is similar and involves cases when a platform with a subverted signer joins the group: the challenger obtains a secret key sk_i from the signer $\mathcal{M}_i$ at the end of the Join protocol, but $\mathcal{M}_i$ is subverted and thus we have no guarantee that sk_i is the "real" secret key.[12] To define forgeries, we solve these issues by assuming the existence of an extractor that, by knowing a trapdoor and seeing the transcript of the Join protocol between the issuer and the sanitizer, can extract a token uniquely linkable (via an efficient procedure) to the secret key that is supposed to correspond to such transcript. This definition is close to the notion of uniquely identifiable transcripts used by [8] for DAA schemes. We stress that the extractor does not exist in the real world and is only an artifact of the security definition.[13] A practical interpretation of our definition is that unforgeability is guaranteed under the assumption that the revocation system is "perfect", namely that one revokes all the secret keys, or signatures produced by those secret keys, that an adversary obtained by interacting with the issuer in the Join protocol.

Definition 4. *Consider the experiment described in Fig. 2. We say that an SR-EPID Π is unforgeable if there exist PPT algorithms* $\mathsf{CheckTK}$*,* $\mathsf{CheckSig}$*, and a PPT extractor $\mathcal{E} = (\mathcal{E}_0, \mathcal{E}_1)$ such that the following properties hold:*

1. *For any pair of keys* $(\mathsf{gpk}, \mathsf{isk})$ *in the support of* $\mathsf{Setup}(\mathsf{pub})$ *and for any (even adversarial)* $\mathsf{tk}, \mathsf{sk}_1, \mathsf{sk}_2$ *it holds* $(\mathsf{CheckTK}(\mathsf{gpk}, \mathsf{sk}_1, \mathsf{tk}) = 1 \wedge \mathsf{CheckTK}(\mathsf{gpk}, \mathsf{sk}_2, \mathsf{tk}) = 1) \Rightarrow \mathsf{sk}_1 = \mathsf{sk}_2$. *(Namely, any* tk *is associated to one and only one* sk.*)*
2. *For any pair of keys* $(\mathsf{gpk}, \mathsf{isk})$ *in the support of* $\mathsf{Setup}(\mathsf{pub})$ *and for any (even adversarial)* $\mathsf{tk}, \mathsf{sk}, M, \mathsf{bsn}, \sigma, \mathsf{SigRL}, \mathsf{PrivRL}$ *such that* $\mathsf{Ver}(\mathsf{gpk}, \mathsf{bsn}, M, \sigma, \mathsf{SigRL}, \mathsf{PrivRL}) = 1$ *and* $\mathsf{Ver}(\mathsf{gpk}, \mathsf{bsn}, M, \sigma, \mathsf{SigRL}, \mathsf{PrivRL} \cup \{\mathsf{sk}\}) = 0$*, it is always the case that* $\mathsf{CheckTK}(\mathsf{gpk}, \mathsf{sk}, \mathsf{tk}) = 0 \vee \mathsf{CheckSig}(\mathsf{gpk}, \mathsf{tk}, \sigma) = 1$. *(Namely, the token* tk *and the algorithm* $\mathsf{CheckSig}$ *allow to verify if a signature comes from a specific secret key.)*

[12] For instance, $\mathcal{M}_i$ may store locally only an obfuscated or encrypted version of the secret key.

[13] More precisely, an extractor does not exist if in the real world the Init algorithm is realized in a trusted manner, akin to CRS generation in NIZK proof systems.

3. *For any PPT adversary $\mathcal{A}$,* $\mathbf{Adv}^{\mathsf{unf}}_{\mathcal{A},\mathcal{E},\Pi}(\lambda) := \Pr\left[\mathbf{Exp}^{\mathsf{unf}}_{\mathcal{A},\mathcal{E},\Pi}(\lambda) = 1\right] \in \mathsf{negl}(\lambda)$.
4. *The distribution* $\{\mathsf{pub} \overset{\$}{\leftarrow} \mathsf{Init}(1^\lambda)\}_{\lambda\in\mathbb{N}}$ *and* $\{\mathsf{pub}|\mathsf{pub},\mathsf{tp} \overset{\$}{\leftarrow} \mathcal{E}_0(1^\lambda)\}_{\lambda\in\mathbb{N}}$ *are computationally indistinguishable.*

Technical details. Besides the use of the extractor, the security experiment is rather technical in some of its parts. Here we explain the main technicalities. As mentioned earlier, the structure of the experiment is that the adversary receives the group public key and then starts interacting with the oracle. The experiment maintains lists $L_{join}, L_{usr}, L_{corr}, L_{msg}$ to bookkeep information on the state of the Join protocol sessions, the list of uncorrupted and corrupted platforms respectively, and the list of the messages on which the adversary obtained signatures. After interacting with the oracle, the adversary outputs a message M^*, a basename bsn^*, a signature σ^* and revocation lists $\mathsf{PrivRL}^*, \mathsf{SigRL}^*$. The adversary wins if either event (4), or the conjunction of events (1), (2) and (3) occur. Intuitively, event (4) means that the adversary has "fooled" the extractor. Namely, the adversary produced a secret key sk (provided in the private-key revocation list PrivRL^*) that the algorithm $\mathsf{CheckTK}$ recognizes as associated to a token tk extracted by $\mathcal{E}_1$, but sk is not a valid signing key. In other words, our definition requires that any secret key[14] extracted by $\mathcal{E}_1$ should be valid. For the other winning case, events (2) and (3) are a generalization of the classical winning condition of digital signatures, i.e. where the adversary returns a valid signature on a new message. The conjunction of event (2) and (3) are more general than the classical unforgeability notion because instead of considering as *new* just the message, we also include the basename, and, more importantly, the fact that the forged signature apparently comes from a machine that either has never been set up or that has never signed the basename-message tuple. Event (1) instead is there to avoid trivial attacks due to the possibility of corrupting group members. Basically, (1) ensures that for any corrupted platform we have either its secret key in PrivRL^* or a signature produced by that platform in SigRL^*. For the latter statement to be efficiently checkable in the experiment we require the existence of an algorithm $\mathsf{CheckSig}$ for this purpose and that works with the token tk extracted by $\mathcal{E}_1$. With `honest_join` queries the adversary specifies the code of a signer $\mathcal{M}_i$, which then runs the Join protocol with an honest issuer and an honest sanitizer controlled by the challenger. At the end, if the issuer accepts, we extract a secret-key token tk_i from the transcript τ of the Join protocol, and we store information about $\mathcal{M}_i$, its state, the verification token and the extracted secret-key token. The verification token svt_i is also returned to the adversary. With `dishonestP_join` queries the adversary can let a fully corrupted platform (i.e., both $\mathcal{M}_i$ and $\mathcal{S}_i$ are under its control) join the group. In this case, the adversary runs the join protocol with the honest issuer controlled by the challenger: the oracle allows the adversary to start a Join session and then sends one message, γ, at a time; lines 9–11 formalize this step-by-step execution of the honest issuer on each message sent by the adversary on behalf

[14] Precisely, $\mathcal{E}$ extracts a token tk linked to sk.

```
Experiment $\mathbf{Exp}^{\mathsf{unf}}_{\mathcal{A},\mathcal{E},\Pi}(\lambda)$:
1 :  $L_{join}, L_{usr}, L_{corr}, L_{msg} \leftarrow \emptyset$; $(\mathsf{pub}, \mathsf{tp}) \leftarrow \mathcal{E}_0(1^\lambda)$; $(\mathsf{gpk}, \mathsf{gsk}) \xleftarrow{\$} \mathsf{Setup}(\mathsf{pub})$;
2 :  $(\mathsf{bsn}^*, M^*, \sigma^*, \mathsf{PrivRL}^*, \mathsf{SigRL}^*) \leftarrow \mathcal{A}(\mathsf{gpk})^{\mathcal{C}(\mathsf{sk},\cdot)}$;
3 :  $R \leftarrow \{\mathsf{tk}_i : (i, *, *, *, \mathsf{tk}_i) \in L_{corr}\}$;
4 :  return 1 if and only if $((1) \wedge (2) \wedge (3)) \vee (4)$ :
5 :     (1) $\forall \mathsf{tk} \in R$ : $(\exists \mathsf{sk} \in \mathsf{PrivRL}^*$ : $\mathsf{CheckTK}(\mathsf{gpk}, \mathsf{sk}, \mathsf{tk}) = 1)$
                  OR $(\exists \sigma \in \mathsf{SigRL}^*$ : $\mathsf{CheckSig}(\mathsf{gpk}, \mathsf{tk}, \sigma) = 1)$
6 :     (2) $\mathsf{Ver}(\mathsf{gpk}, \mathsf{bsn}^*, M^*, \sigma^*, \mathsf{SigRL}^*, \mathsf{PrivRL}^*) = 1$
7 :     (3) $\forall (*, \mathsf{bsn}^*, M^*, \sigma) \in L_{msg}$ : $\mathsf{Link}(\mathsf{gpk}, \mathsf{bsn}^*, M^*, \sigma^*, M^*, \sigma) = 0$
8 :     (4) $\exists \mathsf{sk} \in \mathsf{PrivRL}^*$ and $\mathsf{tk} \in R$ such that $\mathsf{CheckTK}(\mathsf{gpk}, \mathsf{sk}, \mathsf{tk}) = 1$
            but $\mathsf{CheckSK}(\mathsf{gpk}, \mathsf{sk}) = 0$.

Oracle $\mathcal{C}(\mathsf{gsk}, \cdot)$
1 :  Upon query $(\texttt{honest\_join}, i, \mathcal{M}_i)$ :
2 :     if $\exists (i, *, *, *, *) \in L_{usr} \cup L_{corr}$ then return $\perp$;
3 :     $\langle b, (b, \mathsf{svt}_i), \mathsf{state}_i \rangle \leftarrow \mathsf{Join}_{\mathcal{C},\mathcal{C},\mathcal{M}_i} \langle (\mathsf{gpk}, \mathsf{isk}), \mathsf{gpk}, \mathsf{gpk} \rangle$;
4 :     let $\tau$ be the issuer-sanitizer transcript
5 :     if $b = 1$ then $\mathsf{tk}_i \leftarrow \mathcal{E}_1(\mathsf{tp}, \tau)$; $L_{usr} \leftarrow L_{usr} \cup (i, \mathcal{M}_i, \mathsf{state}_i, \mathsf{svt}_i, \mathsf{tk}_i)$;
6 :     return $\mathsf{svt}_i$

7 :  Upon query $(\texttt{dishonestP\_join}, i, \gamma)$ :
8 :     if $(i, *, *, *) \notin L_{join}$, then $L_{join} \leftarrow L_{join} \cup (i, (\mathsf{gpk}, \mathsf{gsk}), \xi, \xi)$;
9 :     Retrieve $(i, \mathsf{state}_\mathcal{I}, \xi, \tau)$ from $L_{join}$;
10 :     $(\gamma_\mathcal{I}, \mathsf{state}'_\mathcal{I}) \leftarrow \mathcal{I}.\mathsf{Join}(\mathsf{state}_\mathcal{I}, \gamma)$; $\mathsf{state}_\mathcal{I} \leftarrow \mathsf{state}'_\mathcal{I}$; $\tau \leftarrow \tau \| (\gamma, \gamma_\mathcal{I})$;
11 :     Update $(i, \mathsf{state}_\mathcal{I}, \xi, \tau)$ in $L_{join}$;
12 :     if $\gamma_\mathcal{I} = \texttt{concluded}$, then $\mathsf{tk}_i \leftarrow \mathcal{E}_1(\mathsf{tp}, \tau)$; store $(i, \perp, \perp, \perp, \mathsf{tk}_i)$ in $L_{corr}$.

13 :  Upon query $(\texttt{dishonestS\_join}, i, \mathcal{U}, \gamma)$ : /* $\mathcal{U} \in \{\mathcal{I}, \mathcal{M}\}$
14 :     if $(i, *, *, *) \notin L_{join}$ then $L_{join} \leftarrow L_{join} \cup (i, (\mathsf{gpk}, \mathsf{gsk}), \xi, \xi)$;
15 :     Retrieve $(i, \mathsf{state}_\mathcal{I}, \mathsf{state}_\mathcal{M}, \tau)$ from $L_{join}$;
16 :     $(\gamma_\mathcal{U}, \mathsf{state}'_\mathcal{U}) \leftarrow \mathcal{U}.\mathsf{Join}(\mathsf{state}_\mathcal{U}, \gamma)$; $\mathsf{state}_\mathcal{U} \leftarrow \mathsf{state}'_\mathcal{U}$; if $\mathcal{U} = \mathcal{I}$ then $\tau \leftarrow \tau \| (\gamma, \gamma_\mathcal{I})$;
17 :     Update $(i, \mathsf{state}_\mathcal{I}, \mathsf{state}_\mathcal{M}, \tau)$ in $L_{join}$;
18 :     if $\mathcal{U} = \mathcal{I} \wedge \gamma_\mathcal{I} = \texttt{concluded}$, then :
19 :        $\mathsf{tk}_i \leftarrow \mathcal{E}_1(\mathsf{tp}, \tau)$; $\mathsf{sk}_i \leftarrow \mathsf{state}_\mathcal{M}$; store $(i, \Pi.\mathcal{M}, \mathsf{sk}_i, \perp, \mathsf{tk}_i)$ in $L_{usr}$.

20 :  Upon query $(\texttt{sign}, i, \mathsf{bsn}, M, \mathsf{SigRL})$ :
21 :     Retrieve the tuple $(i, \mathcal{M}_i, \mathsf{state}_i, \mathsf{svt}_i, \mathsf{tk}_i)$ from $L_{usr}$; if not found return $\perp$;
22 :     $(\mathsf{state}'_i, \sigma) \leftarrow \mathsf{Sig}(\mathcal{M}_i, \mathsf{state}_i, \mathsf{svt}_i, \mathsf{bsn}, M, \mathsf{SigRL})$;
23 :     $L_{msg} \leftarrow L_{msg} \cup (i, \mathsf{bsn}, M, \sigma)$; Update$(i, \mathcal{M}_i, \mathsf{state}'_i, \mathsf{svt}_i, \mathsf{tk}_i)$; return $\sigma$.

24 :  Upon query $(\texttt{corrupt}, i)$ :
25 :     Retrieve tuple$(i, \mathcal{M}_i, \mathsf{state}_i, \mathsf{svt}_i, \mathsf{tk}_i)$ from $L_{usr}$; Move the tuple from $L_{usr}$ to $L_{cor}$;
26 :     return $\mathsf{state}_i$.
```

Fig. 2. Subversion-resilient unforgeability experiment. The algorithm $\mathsf{CheckSK}(\mathsf{gpk}, \mathsf{sk})$ is a shorthand for the following process: sample a random message, generate a signature on it using sk and output 1 iff the signature verifies. The symbol ξ denotes the empty string.

of $\mathcal{S}_i$. At the end, if the issuer accepts, we extract a secret-key token tk_i from the transcript τ of the Join protocol, and we store this token in the list L_{corr} of corrupted users. With `dishonestS_join` queries we consider the case in which the adversary fully controls the sanitizer but the signer is not subverted. In this case, the oracle allows the adversary to run in the Join protocol with the honest issuer and honest signer. This is done by letting the adversary send messages to either $\mathcal{M}$ or $\mathcal{I}$; lines 15–17 formalize this step-by-step execution of the honest issuer and honest signer on each message γ sent by the corrupted sanitizer. At the end, if the issuer accepts, we extract a secret-key token tk_i from the transcript τ of the Join protocol, and we store all the relevant information in the list L_{usr} of honest platforms. Note that in this case we do not necessarily know the verification token since this is received by the sanitizer, which is the adversary. For `sign` queries, the oracle first checks that the platform has joined the system and if so it lets the (possibly subverted) signer $\mathcal{M}_i$ generate a signature σ' and corresponding proof π'_σ. Next, if $\mathsf{svt}_i \neq \perp$ the signature is sanitized and given to the adversary, otherwise a non-sanitized signature is returned. Notice that the case $\mathsf{svt}_i = \perp$ (when i is in L_{usr}) can occur only if the platform joined the system using a `dishonestS_join` query, in which case the sanitizer is controlled by the adversary but – we recall – the signer is not subverted. Finally `corrupt` queries allow the adversary to corrupt an existing platform, which may have joined through either a `honest_join` or `dishonestS_join` query. As a result, the adversary learns the internal state of the signer, which is supposed to contain the secret key (note that the state of the sanitizer, that is the verification token, was already returned after the Join).

Subversion-Resilient Unforgeability in the Random Oracle Model. To capture also constructions in the random oracle model (ROM) we provide a suitable adaptation of the unforgeability definition. A dedicated ROM-based definition is needed in order to consider extractors that may simulate, and program, the random oracle. The ROM definition is the same as Definition 4, except that condition (3) accounts for the ROM-programmability granted to the extractor.

Comparison with Unforgeability of EPID. The notion of unforgeability defined above closely follows the one defined for EPID in [9], with the following main differences. First, in [9] there is no sanitizer. Second, in [9] the adversary cannot specify a subverted signer, namely `honest_join` and `sign` queries are executed according to the protocol description. Third, valid forgeries in [9] include fresh signatures on messages already signed by the oracle. Such a forgery is not valid in our case since signatures are sanitizable (essentially re-randomizable).

Notice that the unforgeability definition of [9] requires the adversary to return the secret key obtained via `dishonest_join` queries (called Join of type (i) in [9]). Nevertheless, the definition does not enforce at any point that the adversary is returning the correct key. It is possible that the authors are implicitly making the assumption that the adversary is honest at this stage, and this what seems to be used in the security proof (where the reduction does not even look at the key returned by the adversary but uses the key extracted from the PoK made by $\mathcal{A}$

during the Join protocol). This is a quite strong assumption. If this assumption is not made we can show an attack. $\mathcal{A}$ first performs a `dishonest_join` query by playing honestly (the same works if this query is `honest_join` followed by `corrupt`), it obtains a key sk_1. Next $\mathcal{A}$ performs another `dishonest_join` query where it plays honestly in the Join protocol, it obtains another key sk_2 but returns to the challenger sk_1. When it comes to the forgery step, from the point of view of the challenger the key that must be in PrivRL^* is sk_1 (maybe twice). This means that technically sk_2 is not revoked and thus the adversary can use it to create a signature that would pass the forgery checks and win the game. Note that this attack works even if the forgery checks ensure that all sk in PrivRL^* must be "valid" (this check was proposed as part of the Revoke algorithm of the EPID construction).

In our definition of unforgeability we avoid the above attack by requiring a security property of the Join protocol. Specifically, the join protocol is such that, if the execution of the protocol ends successfully, then the platform must have learnt one (and only one) secret key. We formalize this by requiring the existence on an extractor that can find this key by only looking at the transcript. In this way, we avoid the unrealistic requirement that the adversary *surrenders* all the corrupted secret keys. Notice that the existence of the extractor is only for definitional purpose, namely, only to asses the security statement that "unforgeability holds if all the corrupted secret keys are revoked".

3 Building Blocks

An asymmetric bilinear group generator is an algorithm $\mathcal{G}$ that upon input a security parameter 1^λ produces a tuple $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \mathcal{P}_1, \mathcal{P}_2)$, where $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are groups of prime order $p \geq 2^\lambda$, the elements $\mathcal{P}_1, \mathcal{P}_2$ are generators of $\mathbb{G}_1, \mathbb{G}_2$ respectively, $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable, non-degenerate bilinear map. In our construction we use Type-3 groups in which it is assumed that there is no efficiently computable isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$. We use the bracket notation introduced in [19]. Elements in $\mathbb{G}_i$, are denoted in implicit notation as $[a]_i := a\mathcal{P}_i$, where $i \in \{1, 2, T\}$ and $\mathcal{P}_T := e(\mathcal{P}_1, \mathcal{P}_2)$. Given $a, b \in \mathbb{Z}_q$ we distinguish between $[ab]_i$, namely the group element whose discrete logarithm base $\mathcal{P}_i$ is ab, and $[a]_i \cdot b$, namely the execution of the multiplication of $[a]_i$ and b, and $[a]_1 \cdot [b]_2 = [a \cdot b]_T$, namely the execution of a pairing between $[a]_1$ and $[b]_2$. Vectors and matrices are denoted in boldface. We extend the pairing operation to vectors and matrices as $e([\mathbf{A}]_1, [\mathbf{B}]_2) = [\mathbf{A}^\top \cdot \mathbf{B}]_T$. All the algorithms take implicitly as input the public parameters bgp.

Structure-Preserving Signatures. A signature scheme over groups generated by $\mathcal{G}$ is a triple of efficient algorithms $(\mathsf{KGen}, \mathsf{Sig}, \mathsf{Ver})$. Algorithm KGen outputs a public verification key vk and a secret signing key sk. Algorithm Sig takes as input a signing key and a message m in the message space, and outputs a signature σ. Algorithm Ver takes as input a verification key vk, a message m and a signature σ, and returns either 1 or 0 (i.e., "accept" or "reject", respectively). The scheme

$(\mathsf{KGen}, \mathsf{Sig}, \mathsf{Ver})$ is correct if for every correctly generated key-pair vk, sk, and for every message m in the message space, we have $\mathsf{Ver}(vk, m, \mathsf{Sig}(sk, m)) = 1$. We consider the standard notion of existential unforgeability under chosen-messages attacks. For space reason, formal definition is omitted from the manuscript. Finally, a signature scheme over groups generated by $\mathcal{G}$ is *structure-preserving* [1] if (1) the verification key, the messages, and signatures consist of solely elements of $\mathbb{G}_1, \mathbb{G}_2$, and (2) the verification algorithm evaluates the signature by deciding group membership of elements in the signature and by evaluating pairing product equations.

Non-Interactive Zero-Knowledge Proof of Knowledge. A non-interactive zero-knowledge (NIZK) proof system for a relation $\mathcal{R}$ is a tuple $\mathcal{NIZK} = (\mathsf{Init}, \mathsf{P}, \mathsf{V})$ of PPT algorithms such that: Init on input the security parameter outputs a (uniformly random) common reference string $\mathsf{crs} \in \{0,1\}^\lambda$; $\mathsf{P}(\mathsf{crs}, x, w)$, given $(x, w) \in \mathcal{R}$, outputs a proof π; $\mathsf{V}(\mathsf{crs}, x, \pi)$, given instance x and proof π outputs 0 (reject) or 1 (accept). In this paper we consider the notion of *NIZK with labels*, that are NIZKs where P and V additionally take as input a label $L \in \mathcal{L}$ (e.g., a binary string). A NIZK (with labels) is *correct* if for every $\mathsf{crs} \overset{\$}{\leftarrow} \mathsf{Init}(1^\lambda)$, any label $L \in \mathcal{L}$, and any $(x, w) \in \mathcal{R}$, we have $\mathsf{V}(\mathsf{crs}, L, x, \mathsf{P}(\mathsf{crs}, L, x, w)) = 1$. As for security we consider the standard notions of adaptive composable zero-knowledge and adaptive perfect knowledge soundness [25].

Malleable and Re-Randomizable NIZKs. We use the definitional framework of Chase et al. [16] for malleable proof systems. For space reason we introduce only informally the framework here. A malleable NIZK comes with an NP relationship $\mathcal{R}$ and a set of *allowable* transformations $\mathcal{T}$. An allowable transformation $T = (T_x, T_w) \in \mathcal{T}$ maps tuple $(x, w) \in \mathcal{R}$ to another tuple $(T_x(x), T_w(w)) \in \mathcal{R}$, namely, the transformations are closed under $\mathcal{R}$. Additionally, a malleable NIZK has an algorithm ZKEval that upon input x, π and transformation T returns a new proof π' which is a valid proof for the instance $T_x(x)$. The work of [16] defines the notion of *strong derivation privacy* which informally states that, for any tuple x, π and transformation T adaptively chosen by the adversary, the proof π' is indistinguishable from a fresh simulated proof for the statement $T_x(x)$. In the special case of a malleable NIZK where the allowable transformation is the identity function we simply say that it is a *re-randomizable NIZK* and we omit the transformation from the inputs of ZKEval.

4 Our SR-EPID Construction

An Overview of Our Scheme. We elaborate further on the overview from Sect. 1.1. Recall that our construction follows the classical template similar to many group signature schemes to prove in zero-knowledge the knowledge of a signature originated by the issuer. In particular: (I) The issuer $\mathcal{I}$ keeps a secret key isk of a (structure-preserving) signature scheme. (II) The secret key of a platform is a signature σ_{sp} on a Pedersen commitment $[t]_1$ whose opening $\mathbf{y}$ is known to the signer only. Following the description given in Sect. 1.1, the conjunction of σ_{sp} and $[t]_1$ forms a blind signature on $\mathbf{y}$. (III) The signer generates a signature on a message M and basename bsn by creating a NIZK with label (bsn, M) of the knowledge of a valid signature σ_{sp} made by $\mathcal{I}$ on message a commitment $[t]_1$ and the knowledge of the opening of such commitment to a value $\mathbf{y}$. To realize the NIZK, our idea is to use a random oracle H to hash the string bsn, M and use the output string as the common-reference string of a (malleable) NIZK for the knowledge of the σ_{sp}, the commitment $[t]_1$ and the opening $\mathbf{y} = (y_0, y_1)$. Furthermore, to be able to re-randomize the signature, we make use the re-randomizable NIZK. (IV) To support revocation and linkability the final signature additionally contains the pseudorandom value $[c_1]_1 := \mathsf{K}(\mathsf{bsn}) \cdot y_0$, where K is a random oracle. More in details, linkability is trivially obtained, as two signatures by the same signer and for the same basename share the same value for $[c_1]_1$, while for (signature-based) revocation we additionally let the signer prove that all the revoked signatures contain a $[c_1]_1$ of the form $\mathsf{K}(\mathsf{bsn}) \cdot y_0'$ where $y_0' \neq y_0$.

Specific Building Blocks. Our scheme makes use of the following building blocks:

– A structure-preserving signature scheme $\mathcal{SS} = (\mathsf{KGen}_{sp}, \mathsf{Sig}_{sp}, \mathsf{Ver}_{sp})$ where messages are elements of $\mathbb{G}_1$ and signatures are in $\mathbb{G}_1^{\ell_1} \times \mathbb{G}_2^{\ell_2}$.
– An re-randomizable NIZK $\mathcal{NIZK}_{\mathsf{sign}}$ for the relationship $\mathcal{R}_{\mathsf{sign}}$ defined as:

$$\left\{ \begin{matrix} (\mathsf{gpk}, [\mathbf{b}]_1, \mathsf{SigRL}), \\ ([t]_1, \sigma_{sp}, [\mathbf{y}]_2) \end{matrix} : \begin{matrix} [\mathbf{b}]_1 \in \mathit{span}([1, y_0]_1^{\mathsf{T}}) \\ [t]_{\mathsf{t}} = [\mathbf{h}^{\mathsf{T}} \cdot \mathbf{y}]_{\mathsf{t}} \\ \mathsf{Ver}_{sp}(\mathsf{pk}_{sp}, [t]_1, \sigma_{sp}) = 1 \\ \forall i : [\mathbf{b}_i]_1 \notin \mathit{span}([1, y_0]_1^{\mathsf{T}}) \end{matrix} \right\}$$

where $\mathsf{SigRL} = \{[\mathbf{b}_i]_1\}_{i=1}^r$, $\mathsf{gpk} = ([\mathbf{h}]_1, \mathsf{pk}_{sp})$, and $\mathbf{y} = (y_0, y_1)^{\mathsf{T}}$. To simplify the exposition, in the description of the protocol below we omit gpk (the public key of the scheme) from the instance and we consider $([\mathbf{b}]_1, \mathsf{SigRL})$ as an instance for the relation.
– A malleable and re-randomizable NIZK $\mathcal{NIZK}_{\mathsf{com}}$ for the following relationship $\mathcal{R}_{\mathsf{com}}$ and set of transformations $\mathcal{T}_{\mathsf{com}}$ defined below:

$$\mathcal{R}_{\mathsf{com}} := \{([\mathbf{h}]_1, [t]_1),\ [\mathbf{y}]_2 : [t]_{\mathsf{t}} = e([\mathbf{h}]_1, [\mathbf{y}]_2)\}$$
$$\mathcal{T}_{\mathsf{com}} := \left\{ T = (T_x, T_w) : \begin{matrix} T_x([\mathbf{h}]_1, [t]_1) = [\mathbf{h}]_1, [t + h_2 \cdot y']_1 \\ T_w([\mathbf{y}]_2) = [y_0, y_1 + y']_2^{\mathsf{T}} \end{matrix} \right\}$$

Namely, the relation proves the knowledge of the opening of a Pedersen's commitment (in $\mathbb{G}_1$) whose commitment key is $[\mathbf{h}]_1$. The transformation allows to re-randomize the commitment by adding fresh randomness.
– A $\mathcal{NIZK}_{\mathsf{svt}}$ for the relation $\mathcal{R}_{\mathsf{svt}} = \{[x, xy, z, zy]_1, y : x, y, z \in \mathbb{Z}_p\}$.
– Three cryptographic hash functions H, J and K modeled as random oracles, where $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$, $\mathsf{J} : \{0,1\}^* \to \{0,1\}^\lambda$ and $\mathsf{K} : \{0,1\}^\lambda \to \mathbb{G}_1$.

Our SR-EPID Scheme. We describe our scheme based on building block described above. For an instantiation and its efficiency see the full version [20].

$\mathsf{Init}(1^\lambda) \to \mathsf{pub}$: Generate description of a type-3 bilinear group $\mathsf{bgp} \overset{\$}{\leftarrow} \mathcal{G}(1^\lambda)$, the common reference string $\mathsf{crs}_{\mathsf{svt}} \overset{\$}{\leftarrow} \mathcal{NIZK}_{\mathsf{svt}}.\mathsf{Init}(\mathsf{bgp})$, and sample $\mathbf{h} \overset{\$}{\leftarrow} \mathbb{Z}_p^2$. Output $\mathsf{pub} = (\mathsf{bgp}, \mathsf{crs}_{\mathsf{svt}}, [\mathbf{h}]_1)$[15]

$\mathsf{Setup}(\mathsf{pub}) \to (\mathsf{gpk}, \mathsf{isk})$: sample $(\mathsf{sk}_{sp}, \mathsf{pk}_{sp}) \overset{\$}{\leftarrow} \mathsf{KGen}_{sp}(\mathsf{bgp})$, and set $\mathsf{isk} := \mathsf{sk}_{sp}$, $\mathsf{gpk} := \mathsf{pk}_{sp}$.

$\mathsf{Join}_{\mathcal{I},\mathcal{S},\mathcal{M}}\langle(\mathsf{gpk}, \mathsf{isk}), \mathsf{gpk}, \mathsf{gpk}\rangle \to \langle b, (b, \mathsf{svt}), (\mathsf{sk}, \mathsf{svt})\rangle$: the platform $\mathcal{P} = (\mathcal{M}, \mathcal{S})$ and issuer $\mathcal{I}$ start an interactive protocol that proceeds as described below:

1. $\mathcal{I}$ samples $id \overset{\$}{\leftarrow} \{0,1\}^\lambda$, sends id to $\mathcal{S}$ and $\mathcal{M}$. Parties compute $\mathsf{crs}_{\mathsf{com}} \leftarrow \mathsf{J}(id)$.
2. $\mathcal{M}$ samples $y_{0,\mathcal{M}}, c_{\mathcal{M}} \overset{\$}{\leftarrow} \mathbb{Z}_p$, set $\mathsf{svt}' := [c_{\mathcal{M}}, c_{\mathcal{M}} y_{0,\mathcal{M}}]_1$ and sends svt' to $\mathcal{S}$.
3. $\mathcal{S}$ parses $\mathsf{svt}' = (\mathsf{svt}'_0, \mathsf{svt}'_1)$, checks that $\mathsf{svt}'_0, \mathsf{svt}'_1 \neq [0]_1$ and if so it sets
$$\mathsf{svt} := c_{\mathcal{S}} \cdot (\mathsf{svt}' + ([0]_1, y_{0,\mathcal{S}} \cdot \mathsf{svt}'_0)) = [c_{\mathcal{S}} c_{\mathcal{M}}, c_{\mathcal{S}} c_{\mathcal{M}}(y_{0,\mathcal{S}} + y_{0,\mathcal{M}})]_1$$
and sends $(y_{0,\mathcal{S}}, \mathsf{svt}, [c_{\mathcal{S}}]_2)$ to $\mathcal{M}$.
4. $\mathcal{M}$ does as described below:
   - Parse $\mathsf{svt}' = (\mathsf{svt}'_0, \mathsf{svt}'_1)$, $\mathsf{svt} = (\mathsf{svt}_0, \mathsf{svt}_1)$ and assert $e(\mathsf{svt}'_0, [c_{\mathcal{S}}]_2) = e(\mathsf{svt}_0, [1]_2)$ and $e(\mathsf{svt}'_1 + [c_{\mathcal{M}} \cdot y_{0,\mathcal{S}}], [c_{\mathcal{S}}]_2) = e(\mathsf{svt}_1, [1]_2)$
   - Set $y_0 = y_{0,\mathcal{M}} + y_{0,\mathcal{S}}$, sample $y_{1,\mathcal{M}} \overset{\$}{\leftarrow} \mathbb{Z}_p$ and compute $[t_{\mathcal{M}}]_1 := (y_0, y_{1,\mathcal{M}}) \cdot [\mathbf{h}]_1$;
   - $\pi_{\mathcal{M}} \leftarrow \mathcal{NIZK}_{\mathsf{com}}.\mathsf{P}(\mathsf{crs}_{\mathsf{com}}, ([\mathbf{h}]_1, [t_{\mathcal{M}}]_1), [y_0, y_{1,\mathcal{M}}]_2)$;
   - Send $([t_{\mathcal{M}}]_1, \pi_{\mathcal{M}})$ to $\mathcal{S}$.
5. $\mathcal{S}$ checks $\mathcal{NIZK}_{\mathsf{com}}.\mathsf{V}(\mathsf{crs}_{\mathsf{com}}, ([\mathbf{h}]_1, [t_{\mathcal{M}}]_1), \pi_{\mathcal{M}}) = 1$; if the check passes:
   - Sample $y_{1,\mathcal{S}} \overset{\$}{\leftarrow} \mathbb{Z}_p$ and set $[t]_1 := [t_{\mathcal{M}} + h_2 \cdot y_{1,\mathcal{S}}]_1$;
   - Compute $\pi_{\mathcal{S}} \leftarrow \mathcal{NIZK}_{\mathsf{com}}.\mathsf{ZKEval}(\mathsf{crs}_{\mathsf{com}}, \pi_{\mathcal{M}}, [y_{1,\mathcal{S}}]_1)$;
   - Send $y_{1,\mathcal{S}}$ to $\mathcal{M}$ and $([t]_1, \pi_{\mathcal{S}})$ to $\mathcal{I}$.
6. $\mathcal{I}$ checks $\mathcal{NIZK}_{\mathsf{com}}.\mathsf{V}(\mathsf{crs}_{\mathsf{com}}, ([\mathbf{h}]_1, [t]_1), \pi_{\mathcal{S}}) = 1$, and if the check passes then $\mathcal{I}$ computes $\sigma_{sp} \leftarrow \mathsf{Sig}_{sp}(\mathsf{sk}_{sp}, [t]_1)$ and sends σ_{sp} to $\mathcal{M}$ (through $\mathcal{S}$).
7. $\mathcal{M}$ does as described below:
   - Compute $y_1 = y_{1,\mathcal{M}} + y_{1,\mathcal{S}}$, $y_0 = y_{0,\mathcal{M}} + y_{0,\mathcal{S}}$, and set $\mathbf{y} := (y_0, y_1)^\mathsf{T}$;
   - Verify (1) $[\mathbf{h}]_1^\mathsf{T} \cdot \mathbf{y} = [t]_1$ and (2) $\mathsf{Ver}_{sp}(\mathsf{pk}_{sp}, [t]_1, \sigma_{sp}) = 1$
   - If so, send the special message `completed` to $\mathcal{I}$ (through $\mathcal{S}$) and output $\mathsf{sk} := ([t]_1, \sigma_{sp}, \mathbf{y})$ and svt.
8. $\mathcal{S}$ outputs svt.
9. If $\mathcal{I}$ receives the special message `completed` then outputs it.

[15] Notice that we could consider a stronger model of subversion where the adversary could additionally subvert the public parameters. Our scheme, indeed, could be proved secure under this stronger model if we generate $[\mathbf{h}]_1$ using the ROM and use $\mathcal{NIZK}_{\mathsf{svt}}$ with subversion-resistant soundness [4].

$\mathsf{Sig}(\mathsf{gpk}, \mathsf{sk}, \mathsf{svt}, \mathsf{bsn}, M, \mathsf{SigRL}) \rightarrow (\sigma, \pi_\sigma)$: On input $\mathsf{gpk}, \mathsf{sk} = ([t]_1, \sigma_{sp}, \mathbf{y})$, the base name $\mathsf{bsn} \in \{0,1\}^\lambda$, the message $M \in \{0,1\}^m$, and a signature revocation list $\mathsf{SigRL} = \{(\mathsf{bsn}_i, M_i, \sigma_i)\}_{i \in [n]}$, generate a signature σ and a proof π_σ as follows:

1. Set $[c]_1 \leftarrow \mathsf{K}(\mathsf{bsn})$ and set $[\mathbf{c}]_1 := [c, c \cdot y_0]_1$;
2. Compute $\pi \leftarrow \Pi_{\mathsf{sign}}.\mathsf{P}(\mathsf{H}(\mathsf{bsn}, M), ([\mathbf{c}]_1, \mathsf{SigRL}), ([t]_1, [\sigma_{sp}]_1, [\mathbf{y}]_2))$;
3. Compute $\pi_\sigma \leftarrow \Pi_{\mathsf{svt}}.\mathsf{P}(\mathsf{crs}_{\mathsf{svt}}, (\mathsf{svt}, [\mathbf{c}]_1), y_0)$;
4. Output $\sigma := ([\mathbf{c}]_1, \pi)$ and π_σ.

$\mathsf{Sanitize}(\mathsf{gpk}, \mathsf{bsn}, M, (\sigma, \pi_\sigma), \mathsf{SigRL}, \mathsf{svt})$: Parse $\sigma = ([\mathbf{c}]_1, \pi)$ and proceed as follows:

1. If $\Pi_{\mathsf{sign}}.\mathsf{V}(\mathsf{crs}_{\mathsf{sign}}, \mathsf{H}(\mathsf{bsn}, M), ([\mathbf{c}]_1, \mathsf{SigRL}), \pi) = 0$ or $\Pi_{\mathsf{svt}}.\mathsf{V}(\mathsf{crs}_{\mathsf{svt}}, (\mathsf{svt}, [\mathbf{c}]_1), \pi_\sigma) = 0$ then output $\perp$.
2. Re-randomize π by computing $\pi' \leftarrow \Pi_{\mathsf{sign}}.\mathsf{ZKEval}(\mathsf{H}(\mathsf{bsn}, M), ([\mathbf{c}]_1, \mathsf{SigRL}), \pi)$
3. Output $\sigma' := ([\mathbf{c}]_1, \pi')$.

$\mathsf{Ver}(\mathsf{gpk}, \mathsf{bsn}, M, \sigma, \mathsf{PrivRL}, \mathsf{SigRL})$: Parse $\sigma = ([\mathbf{c}]_1, \pi)$ and $\mathsf{PrivRL} := \{f_1, \ldots, f_{n_1}\}$. Return 1 if and only if:

1. $\mathsf{K}(\mathsf{bsn}) = [c]_1$,
2. $\Pi_{\mathsf{sign}}.\mathsf{V}(\mathsf{H}(\mathsf{bsn}, M), ([\mathbf{c}]_1, \mathsf{SigRL}), \pi)$ and
3. for $\forall \mathsf{sk} \in \mathsf{PrivRL}$: let $\mathsf{sk} = ([t]_1, \sigma_{sp}, (y_0, y_1))$ check $(-y_0, 1) \cdot [\mathbf{c}]_1 \neq [0]_1$.

$\mathsf{Link}(\mathsf{gpk}, \mathsf{bsn}, M_1, \sigma_1, M_2, \sigma_2)$: Parse $\sigma_i = ([\mathbf{c}_i]_1, \pi_i)$ for $i = 1, 2$. Return 1 if and only if $[\mathbf{c}_1]_1 = [\mathbf{c}_2]_1$ and both signatures are valid, i.e., $\mathsf{Ver}(\mathsf{gpk}, \mathsf{bsn}, M_1, \sigma_1) = 1$ and $\mathsf{Ver}(\mathsf{gpk}, \mathsf{bsn}, M_2, \sigma_2) = 1$.

Remark 1 (On correctness without verification list). Additionally, we assume that for any $\mathsf{crs}, (\mathsf{gpk}, [\mathbf{b}]_1, \mathsf{SigRL})$ and π if $\mathcal{NIZK}_{\mathsf{sign}}.\mathcal{V}(\mathsf{crs}, (\mathsf{gpk}, [\mathbf{b}]_1, \mathsf{SigRL}), \pi) = 1$ then $\mathcal{NIZK}_{\mathsf{sign}}.\mathcal{V}(\mathsf{crs}, (\mathsf{gpk}, [\mathbf{b}]_1, \emptyset), \pi) = 1$. We notice that, by only minor modifications of the verification algorithm, this property holds for GS-NIZK proof system for the relation $\mathcal{R}_{\mathsf{sign}}$. The reason is that GS-NIZK is a commit-and-prove NIZK system where each group element of the witness is committed separately, and where there are different pieces of proof for each of the equation in the conjunction defined by the relation.

Assumption 2 (XDH Assumption). Given a bilinear group description $\mathsf{bgp} \xleftarrow{\$} \mathcal{G}(1^\lambda)$, we say that the *External Diffie-Hellman* (XDH) assumption holds in $\mathbb{G}_\beta$ where $\beta \in \{1,2\}$ if the distribution $[x, y, xy]_\beta$ and the distribution $[x, y, z]_\beta$ where $(x, y, z) \xleftarrow{\$} \mathbb{Z}_p^3$ are computationally indistinguishable.

Theorem 1. *If $\mathcal{SS}$ is EUF-CM secure, both $\mathcal{NIZK}_{\mathsf{sign}}$ and $\mathcal{NIZK}_{\mathsf{com}}$ are adaptive extractable sound, perfect composable zero-knowledge and strong derivation private, $\mathcal{NIZK}_{\mathsf{svt}}$ is adaptive extractable sound, composable zero-knowledge, and both the XDH assumption holds in $\mathbb{G}_1$ and the Assumption 1 holds, the SP-EPID presented above is unforgeable in the ROM.*

To prove unforgeability we need to define an extractor: the main idea is to program the random oracle J to output strings (used as common reference strings in the protocol) that come with extraction trapdoors. Recall that by the properties of the NIZK, such strings are indistinguishable from random strings. Then, whenever required, the extractor can run the NIZK extractor over the NIZK proof provided by the platform during the join protocol to obtain a value $[\mathbf{y}]_2$. Finally, looking at the transcript of the join protocol, the extractor can produce the token $\mathsf{tk} = ([t]_1, \sigma_{sp}, [\mathbf{y}]_2)$. Notice that the created token looks almost like the secret key with the only difference that, in the secret key, the value $\mathbf{y}$ is given in $\mathbb{Z}_q^2$.[16] It is clear that the token is uniquely linked to the secret key.

With this extractor, we proceed with a sequence of hybrid experiments to prove unforgeability. In the first part of the hybrid argument we exploit the programmability of the random oracle to puncture the tuple (bsn^*, M^*) selected by the adversary for its forgery. In particular, we reach a stage where we can always extract the witnesses from valid signatures for (bsn^*, M^*), while for all the other basename-message tuples the challenger can always send to the adversary simulated signatures. To reach this point, we make use of the strong derivation privacy property of the NIZK proof system (which states that re-randomization of valid proofs are indistinguishable from brand-new simulated proofs for the same statement). Specifically, we can switch from signatures produced by the subverted hardware and re-randomized by the challenger of the experiment to signatures directly simulated by the challenger. The latter cutoff any possible channels that the subverted machines can setup with the adversary using biased randomness. At this point we can define the set $\mathcal{Q}_{sp}$ of all the messages $[t]_1$ signed by the challenger (impersonating the issuer) using the structure-preserving signature scheme. Notice that our definition allows the adversary to query the challenger for a signature on the message (bsn^*, M^*) itself. As the signatures for such basename-message tuple are always extractable, the challenger has no chances to simulate such signatures. However, by the security definition, the adversary is bound to output a forgery that does not link to any of the signatures for (bsn^*, M^*) output by the challenger. We exploit this property together with the fact that two not-linkable signatures must have different value for y_0, to show that the forged signature must be produced with a witness that contains a fresh value $[t^*]_1$ that is not in $\mathcal{Q}_{sp}$. More technically, we can reduce this to the binding property of the Pedersen's commitment scheme that we use.

Now, we can divide the set of the adversaries in two classes: the ones which produce a forged signature where $[t^*]_1$ is in $\mathcal{Q}_{sp}$ and the ones where $[t^*]_1$ is not in $\mathcal{Q}_{sp}$. For the latter, we can easily reduce to the unforgeability of the structure preserving signature scheme. For the former, instead, we need to proceed with more caution. First of all, we are assured by the previous step that adversaries from the first class of adversaries would never query the signature oracle on (bsn^*, M^*). Secondly, we use the puncturing technique again, however, this time we select the platform (let it be the platform number j^*) that is linked to the

[16] In our concrete instantiation we use GS-NIZK proof system, for which extraction in the source groups is more natural and efficient.

forged signature. By the definition of the class of adversaries this platform always exists. For this platform we switch the common-reference string used in the join protocol to be zero-knowledge. Once we are in zero-knowledge mode, we can use strong derivation privacy to make sure that the join protocol does not leak any information about the secret key that the platform computes (even if the machine is corrupted). At this point the secret key of the j^*-th platform is apparently completely hidden from the view of the adversary, in fact: (1) all the signatures are simulated and (2) the join protocol of the j-th platform is simulated. However, the j^*-th platform is still using a subverted machine, which, although cannot communicate anymore using biased randomness with the outside adversary, still receives the secret key. We show that we can substitute this subverted machine with a *well-behaving* machine that might abort during the join protocol but that, if it does not so then it always sign every basename-message tuple received (here we rely on Assumption 1). The last step is to show that such forgery would break the hiding property of the Pedersen's commitment scheme that we make use of.

Theorem 2. *If $\mathcal{NIZK}_{\mathsf{sign}}$ and $\mathcal{NIZK}_{\mathsf{com}}$ are strongly derivation private, adaptively extractable sound and adaptively composable perfect zero-knowledge, both the XDH assumption in $\mathbb{G}_1$ holds and the Assumption 1 holds, and $\mathcal{NIZK}_{\mathsf{svt}}$ is adaptively sound, then the SR-EPID described above is anonymous in the ROM.*

First we notice that adaptive corruption and selective corruption for anonymity are equivalent up to a polynomial degradation of the advantage of the adversary. In particular, we can assume that the adversary corrupts all the platforms but the i_1-th and the i_2-th platforms used for the challenge of security game. The idea of the reduction is to switch to zero-knowledge the common reference strings used in the join protocols for the platforms i_1 and i_2 by programming the random oracle. Similarly, switch to zero-knowledge and simulate all the signatures output by the two platforms (again by programming the random oracle). Thus using the strong derivation privacy property of $\mathcal{NIZK}_{\mathsf{sign}}$ and $\mathcal{NIZK}_{\mathsf{com}}$ to make sure that no information about the platform keys is exfiltrated. Notice that at this point the machines cannot communicate any information using biased randomness, on the other hand, they could still communicate using valid/invalid signatures. Although, the definition of anonymity disallows telling apart i_1 from i_2 using this channel, for technical reasons, in the last step of the proof (when we reduce to XDH) we need to completely disconnect the subverted machines and, again, substitute them with well-behaving machines, thus here we need to rely on Assumption 1. At this point the element $y_0^{(1)}$ (resp. $y_0^{(2)}$) of the key $\mathbf{y}^{(1)}$ of the platform i_1 (resp. key $\mathbf{y}^{(2)}$ of the platform i_2) are almost hidden to the view of the adversary. However, the challenge signature $\sigma = ([\mathbf{c}^*]_1, \pi)$ still contains the value $[c_1^*]_1 = \mathsf{K}(\mathsf{bsn}^*) \cdot y_0^{(b)}$. The last step of the proof of anonymity is to change the way the challenge signature is computed. In particular, the value above is computed as $\mathsf{K}(\mathsf{bsn}^*) \cdot x$ for a uniformly sampled x. This step is proved indistinguishable using the XDH assumption on $\mathbb{G}_1$.

Acknowledgements. This work has received funding in part from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program under project PICOCRYPT (grant agreement No. 101001283), by the Spanish Government under projects SCUM (ref. RTI2018-102043-B-I00), CRYPTOEPIC (ref. EUR2019-103816), and SECURITAS (ref. RED2018-102321-T), and by the Madrid Regional Government under project BLOQUES (ref. S2018/TCS-4339). This work has been supported by SPATIAL project. SPATIAL has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 101021808. This work has been partially funded by the EU H2020-SU-ICT-03-2018 Project No. 830929 CyberSec4Europe. The first author was a postdoctoral fellow at the IMDEA Software Institute where he performed the research leading to this paper.

References

1. Abe, M., Fuchsbauer, G., Groth, J., Haralambiev, K., Ohkubo, M.: Structure-preserving signatures and commitments to group elements. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 209–236. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_12
2. Ateniese, G., Francati, D., Magri, B., Venturi, D.: Public immunization against complete subversion without random oracles. In: Deng, R.H., Gauthier-Umaña, V., Ochoa, M., Yung, M. (eds.) ACNS 2019. LNCS, vol. 11464, pp. 465–485. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21568-2_23
3. Ateniese, G., Magri, B., Venturi, D.: Subversion-resilient signature schemes. In: ACM CCS 2015 (2015)
4. Bellare, M., Fuchsbauer, G., Scafuro, A.: NIZKs with an untrusted CRS: security in the face of parameter subversion. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part II. LNCS, vol. 10032, pp. 777–804. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53890-6_26
5. 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 2003. LNCS, vol. 2656, pp. 614–629. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-39200-9_38
6. Bellare, M., Paterson, K.G., Rogaway, P.: Security of symmetric encryption against mass surveillance. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 1–19. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44371-2_1
7. Bellare, M., Sandhu, R.: The security of practical two-party RSA signature schemes. Cryptology ePrint Archive, Report 2001/060 (2001). https://eprint.iacr.org/2001/060
8. Bernhard, D., Fuchsbauer, G., Ghadafi, E., Smart, N.P., Warinschi, B.: Anonymous attestation with user-controlled linkability. Int. J. Inf. Secur. **12**(3), 219–249 (2013)
9. Brickell, E., Li, J.: Enhanced privacy ID: a direct anonymous attestation scheme with enhanced revocation capabilities. In: ACM WPES (2007)
10. Brickell, E., Li, J.: Enhanced privacy ID: a direct anonymous attestation scheme with enhanced revocation capabilities. IEEE Trans. Dependable Sec. Comput. **9**(3), 345–360 (2011)

11. Camenisch, J., Chen, L., Drijvers, M., Lehmann, A., Novick, D., Urian, R.: One TPM to bind them all: fixing TPM 2.0 for provably secure anonymous attestation. In: 2017 IEEE S&P, pp. 901–920 (2017)
12. Camenisch, J., Drijvers, M., Lehmann, A.: Anonymous attestation with subverted TPMs. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part III. LNCS, vol. 10403, pp. 427–461. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63697-9_15
13. Camenisch, J., Lehmann, A.: (Un)linkable pseudonyms for governmental databases. In: ACM CCS 2015 (2015)
14. Catalano, D., Fiore, D., Nizzardo, L.: Programmable hash functions go private: constructions and applications to (homomorphic) signatures with shorter public keys. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 254–274. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48000-7_13
15. Chakraborty, S., Dziembowski, S., Nielsen, J.B.: Reverse firewalls for actively secure MPCs. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 732–762. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56880-1_26
16. Chase, M., Kohlweiss, M., Lysyanskaya, A., Meiklejohn, S.: Malleable proof systems and applications. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 281–300. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_18
17. Chen, R., Mu, Y., Yang, G., Susilo, W., Guo, F., Zhang, M.: Cryptographic reverse firewall via malleable smooth projective hash functions. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 844–876. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_31
18. Chow, S.S.M., Russell, A., Tang, Q., Yung, M., Zhao, Y., Zhou, H.-S.: Let a non-barking watchdog bite: cliptographic signatures with an offline watchdog. In: Lin, D., Sako, K. (eds.) PKC 2019, Part I. LNCS, vol. 11442, pp. 221–251. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17253-4_8
19. Escala, A., Herold, G., Kiltz, E., Ràfols, C., Villar, J.: An algebraic framework for Diffie-Hellman assumptions. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 129–147. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_8
20. Faonio, A., Fiore, D., Nizzardo, L., Soriente, C.: Subversion-resilient enhanced privacy ID. Cryptology ePrint Archive (2020). https://ia.cr/2020/1450
21. Faust, S., Kohlweiss, M., Marson, G.A., Venturi, D.: On the non-malleability of the Fiat-Shamir transform. In: Galbraith, S., Nandi, M. (eds.) INDOCRYPT 2012. LNCS, vol. 7668, pp. 60–79. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34931-7_5
22. Fischlin, M., Mazaheri, S.: Self-guarding cryptographic protocols against algorithm substitution attacks (2018)
23. Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for cryptographers. Discret. Appl. Math. **156**(16), 3113–3121 (2008)
24. Ganesh, C., Magri, B., Venturi, D.: Cryptographic reverse firewalls for interactive proof systems. Theor. Comput. Sci. **855**, 104–132 (2021)
25. Groth, J.: Simulation-sound NIZK proofs for a practical language and constant size group signatures. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 444–459. Springer, Heidelberg (2006). https://doi.org/10.1007/11935230_29

26. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Smart, N. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 415–432. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78967-3_24
27. Libert, B., Peters, T., Joye, M., Yung, M.: Non-malleability from malleability: simulation-sound quasi-adaptive NIZK proofs and CCA2-secure encryption from homomorphic signatures. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 514–532. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_29
28. Mavroudis, V., Cerulli, A., Svenda, P., Cvrcek, D., Klinec, D., Danezis, G.: A touch of evil: high-assurance cryptographic hardware from untrusted components. In: ACM CCS, pp. 1583–1600 (2017)
29. Mironov, I., Stephens-Davidowitz, N.: Cryptographic reverse firewalls. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 657–686. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_22
30. Russell, A., Tang, Q., Yung, M., Zhou, H.-S.: Cliptography: clipping the power of kleptographic attacks. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part II. LNCS, vol. 10032, pp. 34–64. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53890-6_2
31. Russell, A., Tang, Q., Yung, M., Zhou, H.S.: Generic semantic security against a kleptographic adversary. In: ACM CCS 2017 (2017)
32. Young, A., Yung, M.: The dark side of "black-box" cryptography or: should we trust capstone? In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 89–103. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_8