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 luca@protocol.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.

* This is the full version of the paper that appears in the proceedings of CT-RSA 2022.

Table of Contents

1 Introduction

Anonymous attestation is a key feature of secure hardware platforms, such as Intel SGX[5] or the Trusted Computing Group's Trusted Platform Module[6]. 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 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[7] 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.

[5] `https://www.intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions.html`
[6] `https://trustedcomputinggroup.org/resource/tpm-library-specification/`
[7] `https://snowdenarchive.cjfe.org/`

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.[8] 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 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[9]. 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 also 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. For completeness, we describe this simpler and non-subversion-resilient version of our notion in Appendix A. 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. Given that EPID schemes are already deployed in real-world systems, we believe this is a result that can be of independent interest for the community.

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

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

[9] 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.

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 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. The approach described so far follows closely the one of EPID [9,10].

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 Section 2.3.

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 in order to reconcile our extensive use of malleable NIZKs (and in particular, Groth-Sahai proofs [25]) 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 not only malleable but also to have a form of simulation-extractable soundness.[10] 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 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[11]. 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 [26,14] 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 $n \geq 7$).

[10] 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.

[11] 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].

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. [18] 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 roughly interpreted as a new EPID scheme equipped with a 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 Sec. 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 Attestion (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.

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. [19], 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. Thus, a valid signature σ must carry a zero-knowledge proof that the private key used to compute σ is different from any of the keys used to produce any of the signatures in SigRL.

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 no adversary—not even a corrupted group manager—can create a signature on a message m and basename B, that links to a signature of an honest group member (when this honest group member never signed m, B). 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.

2.1 Subversion-Resilient EPID

Overview and rationale of the definition. For simplicity, we assume each signer to be paired with a sanitizer and we denote a pair of signer-sanitizer as a "platform".[12] 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.[13]

The goal of the sanitizer is to remove any possible covert channel from the signer to an external adversary. For example, a subverted signer could establish a covert channel through the randomness used at signature generation. Alternatively, a subverted signer may maliciously influence the join protocol to obtain as output a fixed secret key that it is a prior known to the adversary; later on, the adversary may simply use this known private key to break anonymity (since, by definition, private key based revocation allows a verifier to tell if a signature has been produced with a given private key). Yet another option is for the signer to behave honestly during the join protocol, but later use a preloaded secret key to produce signatures.[14] Once again, the adversary may use that known key and a signature to break platform anonymity.

To deal with these issues, our notion of SR-EPID is designed so that (i) the sanitizer participates to the join protocol contributing to the private key of the signer, (ii) each signature output by the signer carries a proof (for the sanitizer to verify) that the private key used for signing is the very same one obtained during the join protocol, and (iii) the sanitizer sanitizes signatures to avoid covert channel based on maliciously-sampled randomness.

The resulting syntax is a generalization of EPID that adds a Sanitize algorithm and modifies the original Join and Sig algorithms.

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

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

[14] Since anonymity must hold also against a malicious group authority, it is possible for the signer to hold one or more certified private keys.

2.2 Syntax of Subversion-Resilient EPID (SR-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 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 outputs 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* $\mathsf{Ver}(\mathsf{gpk}, \mathsf{bsn}, M, \sigma') = 1$*.*

The next definition says that an SR-EPID scheme is correct if any signature produced by a non-revoked group member passes the verification procedure. 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:*

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

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

2.3 Subversion-resilient Security

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. 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

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

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

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

Upon query $(\mathtt{join}, i, \gamma_{\mathcal{I}})$:
  Retrieve $(i, \mathcal{M}_i, \mathsf{state}_{\mathcal{S}}, \mathsf{state}_{\mathcal{M}})$ from L_{join}, ;
  If not find parse $\gamma_{\mathcal{I}} = \mathcal{M}_i$ and add $(i, \mathcal{M}_i, \perp, \perp)$ in L_{join} and **return** ;
  $(\mathsf{state}''_{\mathcal{S}}, \mathsf{state}'_{\mathcal{M}}, \gamma_{\mathcal{S}}) \leftarrow \mathsf{Join}(\mathcal{M}_i, \mathsf{state}_{\mathcal{S}}, \mathsf{state}_{\mathcal{M}}, \gamma_{\mathcal{I}})$;
  Store $(i, \mathcal{M}_i, \mathsf{state}''_{\mathcal{S}}, \mathsf{state}'_{\mathcal{M}})$ in L_{join};
  if $\gamma_{\mathcal{S}} = \mathtt{concluded}$ **then**
    $\mathsf{svt}_i \leftarrow \mathsf{state}''_{\mathcal{S}}$, store $(i, \mathcal{M}_i, \mathsf{state}'_{\mathcal{M}}, \mathsf{svt}_i, \emptyset)$ in L_{usr}; **return** $(\gamma_{\mathcal{S}}, \mathsf{svt}_i)$;
  else return $\gamma_{\mathcal{S}}$.
Upon query $(\mathtt{sign}, i, \mathsf{bsn}, M, \mathsf{SigRL})$:
  if $(i, *, *, *, *) \notin L_{usr}$ **then** $\mathsf{Bad} \leftarrow \mathsf{true}$;
  else retrieve $(i, \mathcal{M}_i, \mathsf{state}_i, \mathsf{svt}_i, B_i) \in L_{usr}$;
  $(\mathsf{state}'_i, \sigma) \leftarrow \mathsf{Sig}(\mathcal{M}_i, \mathsf{state}_i, \mathsf{svt}_i, \mathsf{bsn}, M, \mathsf{SigRL})$;
  Update $(i, \mathcal{M}_i, \mathsf{state}'_i, \mathsf{svt}_i, B_i \cup \{\mathsf{bsn}\})$;
  if $post = 0$ or $i \notin \{i_0, i_1\}$ **then return** σ
  else let $i = i_\beta$ and $\beta \in \{0, 1\}$;
  if $\mathsf{bsn} = \mathsf{bsn}^*$ **then** $\mathsf{Bad} \leftarrow \mathsf{true}$;
  Let $i = i_{1-\beta}$, retrieve $(i_\beta, \mathcal{M}_{i_\beta}, \mathsf{state}_{i_\beta}, \mathsf{svt}_{i_\beta}, B_{i_\beta}) \in L_{usr}$;
  $(\mathsf{state}'_{i_\beta}, \tilde{\sigma}) \leftarrow \mathsf{Sig}(\mathcal{M}_{i_\beta}, \mathsf{state}_{i_\beta}, \mathsf{svt}_{i_\beta}, \mathsf{bsn}, M, \mathsf{SigRL})$;
  Update $(i_\beta, \mathcal{M}_{i_\beta}, \mathsf{state}'_{i_\beta}, \mathsf{svt}_{i_\beta}, B_{i_\beta} \cup \{\mathsf{bsn}\})$ in L_{usr};
  if $\perp \in \{\sigma, \tilde{\sigma}\}$ **then** $\mathsf{Bad} \leftarrow \mathsf{true}$; **endif return** σ;
Upon query $(\mathtt{corrupt}, i)$:
  if $post = 1 \wedge i \in \{i_0, i_1\}$ **then** $\mathsf{Bad} \leftarrow \mathsf{true}$;
  else retrieve $(i, \mathcal{M}_i, \mathsf{state}_i, \mathsf{svt}_i)$ from L_{usr};
    move the tuple from L_{usr} to L_{cor};
    return $(\mathsf{state}_i, \mathsf{svt}_i)$.

Fig. 1: Subversion-resilient anonymity experiment.

a subverted machine $\tilde{\mathcal{M}}$ with hardcoded a secret key $\tilde{\mathsf{sk}}$ that during joining time acts honestly, thus 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 and provide a formal proof in Appendix B. Finally we notice that the model without verification token mechanism, after some necessary cosmetic changes, fits with the cryptographic reverse firewall framework. In the lingo of [29], the sanitizer of a scheme satisfying the anonymity property which works without the verification token mechanism, is a cryptographic reverse firewall that *weakly preserve* the anonymity property for the signer $\mathcal{S}$. 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[15]

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.

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

[15] 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.

have no guarantee that sk_i is the "real" secret key.[16] 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. [17] 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.

We formalize subversion-resilient unforgeability for SR-EPID via the experiment of Fig. 2, and we formally define unforgeability as follows.

Definition 4. *Consider the experiment described in Fig. 2. We say that an SR-EPID Π is unforgeable if there exist PPT algorithms* CheckTK, 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* CheckSig *allow to verify if a signature comes from a specific secret key.)*
3. *For any PPT adversary $\mathcal{A}$,* $\mathbf{Adv}^{\mathtt{unf}}_{\mathcal{A},\mathcal{E},\Pi}(\lambda) := \Pr\left[\mathbf{Exp}^{\mathtt{unf}}_{\mathcal{A},\mathcal{E},\Pi}(\lambda) = 1\right] \in \mathsf{negl}(\lambda)$.
4. *The distribution* $\{\mathsf{pub} \xleftarrow{\$} \mathsf{Init}(1^\lambda)\}_{\lambda\in\mathbb{N}}$ *and* $\{\mathsf{pub}|\mathsf{pub}, \mathsf{tp} \xleftarrow{\$} \mathcal{E}_0(1^\lambda)\}_{\lambda\in\mathbb{N}}$ *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 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[18] 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 CheckSig for this purpose and that works with the token tk extracted by $\mathcal{E}_1$.

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

[17] 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.

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

Experiment $\mathbf{Exp}^{\mathsf{unf}}_{\mathcal{A},\mathcal{E},\Pi}(\lambda)$:

$L_{join}, L_{usr}, L_{corr}, L_{msg} \leftarrow \emptyset$; $(\mathsf{pub}, \mathsf{tp}) \leftarrow \mathcal{E}_0(1^\lambda)$; $(\mathsf{gpk}, \mathsf{gsk}) \xleftarrow{\$} \mathsf{Setup}(\mathsf{pub})$;
$(\mathsf{bsn}^*, M^*, \sigma^*, \mathsf{PrivRL}^*, \mathsf{SigRL}^*) \leftarrow \mathcal{A}(\mathsf{gpk})^{\mathcal{C}(\mathsf{sk},\cdot)}$;
$R \leftarrow \{\mathsf{tk}_i : (i, *, *, *, \mathsf{tk}_i) \in L_{corr}\}$;
return 1 if and only if $((1) \wedge (2) \wedge (3)) \vee (4)$:
  (1) $\forall \mathsf{tk} \in R : (\exists \mathsf{sk} \in \mathsf{PrivRL}^* : \mathsf{CheckTK}(\mathsf{gpk}, \mathsf{sk}, \mathsf{tk}) = 1)$
    OR $(\exists \sigma \in \mathsf{SigRL}^* : \mathsf{CheckSig}(\mathsf{gpk}, \mathsf{tk}, \sigma) = 1)$
  (2) $\mathsf{Ver}(\mathsf{gpk}, \mathsf{bsn}^*, M^*, \sigma^*, \mathsf{SigRL}^*, \mathsf{PrivRL}^*) = 1$
  (3) $\forall (*, \mathsf{bsn}^*, M^*, \sigma) \in L_{msg} : \mathsf{Link}(\mathsf{gpk}, \mathsf{bsn}^*, M^*, \sigma^*, M^*, \sigma) = 0$
  (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 $\mathcal{C}(\mathsf{gsk}, \cdot)$

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

Upon query $(\texttt{dishonestP\_join}, i, \gamma)$:
  if $(i, *, *, *) \notin L_{join}$, **then** $L_{join} \leftarrow L_{join} \cup (i, (\mathsf{gpk}, \mathsf{gsk}), \xi, \xi)$;
  Retrieve $(i, \mathsf{state}_{\mathcal{I}}, \xi, \tau)$ from L_{join};
  $(\gamma_{\mathcal{I}}, \mathsf{state}'_{\mathcal{I}}) \leftarrow \mathcal{I}.\mathsf{Join}(\mathsf{state}_{\mathcal{I}}, \gamma)$; $\mathsf{state}_{\mathcal{I}} \leftarrow \mathsf{state}'_{\mathcal{I}}$; $\tau \leftarrow \tau \| (\gamma, \gamma_{\mathcal{I}})$;
  Update $(i, \mathsf{state}_{\mathcal{I}}, \xi, \tau)$ in L_{join};
  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}.

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

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

Upon query $(\texttt{corrupt}, i)$:
  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};
  return 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.

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 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. In order to capture also constructions in the random oracle model (ROM)—as ours—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 essentially the same as Def 4, except that condition (3) is modified to account for the programmability powers granted to the extractor. More in detail, all the random oracle queries (both made by the adversary and by the corrupted signer $\mathcal{M}_i$) are passed to the extractor, which is now a stateful machine; the extractor must provide a view to the adversary that is indistinguishable from the *real world* view, where the ROM outputs uniformly random strings. To formalize this, we consider a dummy extractor $\tilde{\mathcal{E}}$ that (i) initializes the public parameters as done by the SR-EPID scheme, and (ii) it does not program the ROM answers, but simply outputs uniformly random values. We additionally require that the view of the adversary in an execution of the experiment with the extractor and the view of the adversary in an execution of the experiment with the dummy extractor are indistinguishable.

Definition 5 (Unforgeability in the ROM.). *Consider a game similar to Fig 2 where additionally the extractor can program the random oracle. Namely, all the queries to the random oracle made by $\mathcal{A}$ are redirected and answered by the extractor $\mathcal{E}$. We say that an SR-EPID scheme Π is* unforgeable in the ROM *if conditions (1), (2), (3) and (4) of Def. 4 hold, and additionally, the view of the adversary at the end of the experiment $\mathbf{Exp}^{\mathtt{unf}}_{\mathcal{A},\mathcal{E},\Pi}(1^\lambda)$ and the view of the adversary at the end of the experiment $\mathbf{Exp}^{\mathtt{unf}}_{\mathcal{A},\tilde{\mathcal{E}},\Pi}(1^\lambda)$ are computationally indistinguishable.*

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".

TODO:TO BE REDONE AFTER THE CT-RSA COMMENTS

Subversion-Resilient Non-Frameability. We also define non-frameability for SR-EPID. This property models that an adversarial issuer should not be able to produce a signature that links to the identity of an honest platform with a possibly subverted machine. Since subversion-resilient non-frameability should generalize the standard notion of non-frameability, we also require that a malicious issuer and sanitizer should not be able to produce a signature that links to the identity of an honest machine.

First we define correctness for the linking algorithm.

Definition 6. *We say that an SR-EPID scheme satisfies linking 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_0, M_1, *for any* PrivRL *and* SigRL, *and any* $\sigma_b \leftarrow \mathsf{Sanitize}(\mathsf{gpk}, \mathsf{bsn}, M_b, \mathsf{Sig}(\mathsf{gpk}, \mathsf{sk}, \mathsf{bsn}, M, \mathsf{SigRL}), \mathsf{SigRL}, \mathsf{svt})$ *for* $b \in \{0, 1\}$ *we have:*

$$(\sigma_0 \neq \perp \vee \sigma_1 \neq \perp) \Rightarrow \mathsf{Link}(\mathsf{gpk}, \mathsf{bsn}, M_0, \sigma_0, M_1, \sigma_1) = 1 \tag{3}$$

We now define subversion-resilient non-frameability. This notion formalizes the idea that an adversarial issuer should not be able to produce a signature that links to the identity of a platform where either (1) the machine is subverted but the sanitizer is honest or (2) the machine is honest and the sanitizer is malicious. Since "linking" is only possible across signatures, we treat non-frameability as the property that guarantees that no adversary can output a signature that links to another signature output by an honest platform.

We formalize subversion-resilient non-frameability for SR-EPID in a security experiment in Fig. 3, and we formally define non-frameability as follows.

Definition 7. *Consider the experiment described in Fig. 3. We say that an SR-EPID* Π *is non-framable if for any PPT adversary* $\mathcal{A}$*:*

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

Here we provide an intuition on the notion. Similar to the anonymity experiment, in the non-frameability one, the adversary plays the role of the issuer and can do the following: (1) ask platforms with subverted signers to join the system; (2) ask platforms with honest signers but malicious sanitizers to join; (3) ask platform to sign messages; (4) corrupt platforms. For (1), it means that the adversary specifies the code of a signer $\mathcal{M}_i$, run the Join protocol with $\mathcal{S}_i$ controlled by the challenger and the issuer controlled by the adversary. For (2), the adversary plays the join protocol with an honest signer $\mathcal{M}$, it controls both the sanitizer and the issuer. For (3), a platform that joined the system creates a signature. Depending on the type of corruption of the platform (either (1) or (2)) the adversary gets to see the sanitized signature (case (1)) or both the signature and the proof π_σ (case (2)). 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.

The adversary must output $(i^*, \mathsf{bsn}^*, M^*, \sigma^*)$ providing the victim platform index i^* and a basename-message-signature triple $\mathsf{bsn}^*, M^*, \sigma^*$. The adversary wins the experiment if (1) σ^* is a valid signature for bsn^*, M^*, (2) the signature "links" to one of the signatures produced by the oracle when queried on platform i^*, and (3) if the oracle has output a signature on bsn, M on behalf of platform i and if $\mathsf{bsn} = \mathsf{bsn}^*$, then $M \neq M^*$. In the experiment, the challenger keeps a list L_i of signatures and their respective basename-message pairs, for each of the non-corrupted platforms that have joined the group.

3 Building Blocks

3.1 Bilinear groups

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 [20]. Elements in $\mathbb{G}_i$, are denoted in implicit notation as $[a]_i := a\mathcal{P}_i$, where $i \in \{1, 2, T\}$ and $\mathcal{P}_T := e(\mathcal{P}_1, \mathcal{P}_2)$. Every element in $\mathbb{G}_i$ can be written as $[a]_i$ for some $a \in \mathbb{Z}_q$, but note that given $[a]_i$, it is in general hard to compute $a \in \mathbb{Z}_q$ (discrete logarithm problem). 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.

3.2 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 say that a signature scheme $(\mathsf{KGen}, \mathsf{Sig}, \mathsf{Ver})$ is existentially unforgeable under adaptive chosen message attack (EUF-CMA) if for any PTT adversary $\mathcal{A}$ we have that:

$$\Pr\left[\mathsf{Ver}(vk, m, \sigma) = 1 \wedge m \notin Q : \begin{array}{c}(vk, sk) \xleftarrow{\$} \mathsf{KGen}(\mathcal{G}(1^\lambda)),\\ (m, \sigma) \leftarrow \mathcal{A}^{\mathsf{Sig}(sk, \cdot)}(vk)\end{array}\right] \in \mathsf{negl}(\lambda),$$

where Q is the set of messages queried by $\mathcal{A}$ to the signing oracle. A stronger notion of unforgeability, named "strong" EUF-CMA or sEUF-CMA, further prevents the adversary to forge a new signature on a message that has already been signed. This notion is captured by modifying the above definition so that $(m, \sigma) \notin Q$ whereas Q is defined as the set of message-signature pairs stemming from the adversary's queries

Experiment $\mathbf{Exp}^{\mathsf{non\text{-}fram}}_{\mathcal{A},\Pi}(\lambda)$

$\mathsf{pub} \leftarrow \mathsf{Init}(1^\lambda)$; $\mathsf{gpk} \leftarrow \mathcal{A}(\mathsf{pub})$; $L_{join}, L_{usr}, L_{corr} \leftarrow \emptyset$;
$(i^*, \mathsf{bsn}^*, M^*, \sigma^*) \leftarrow \mathcal{A}(\mathsf{gpk})^{\mathcal{C}(\mathsf{gpk},\cdot)}$;
Retrieve the tuple $(i^*, \mathcal{M}_{i^*}, \mathsf{state}_{i^*}, \mathsf{svt}_{i^*}, L_{i^*})$ from L_{usr};
Output 1 if all of the following conditions hold:
  (1) $(i^*, *, *, *, *) \in L_{usr}$,
  (2) $\mathsf{Ver}(\mathsf{gpk}, \mathsf{bsn}^*, M^*, \sigma^*, \emptyset) = 1$,
  (3) $\exists\ \langle M, \mathsf{bsn}, \sigma\rangle \in L_{i^*}$ such that $\mathsf{Link}(\mathsf{gpk}, M, \sigma, M^*, \sigma^*) = 1$,
  (4) $\forall\ \langle M, \mathsf{bsn}, \sigma\rangle \in L_{i^*}$: **if** $\mathsf{bsn} = \mathsf{bsn}^*$ **then** $M \neq M^*$.

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

Upon query $(\texttt{join}, i, \gamma_{\mathcal{I}})$:
  Retrieve $(i, \mathcal{M}_i, \mathsf{state}_{\mathcal{S}}, \mathsf{state}_{\mathcal{M}}, 0)$ from L_{join}, ;
  If not find parse $\gamma_{\mathcal{I}} = \mathcal{M}_i$ and add $(i, \mathcal{M}_i, \perp, \perp, 0)$ in L_{join} and **return** ;
  $(\mathsf{state}''_{\mathcal{S}}, \mathsf{state}'_{\mathcal{M}}, \gamma_{\mathcal{S}}) \leftarrow \mathsf{Join}(\mathcal{M}_i, \mathsf{state}_{\mathcal{S}}, \mathsf{state}_{\mathcal{M}}, \gamma_{\mathcal{I}})$;
  Store $(i, \mathcal{M}_i, \mathsf{state}''_{\mathcal{S}}, \mathsf{state}'_{\mathcal{M}}, 0)$ in L_{join};
  if $\gamma_{\mathcal{S}} = \texttt{concluded}$ **then**
    $\mathsf{svt}_i \leftarrow \mathsf{state}''_{\mathcal{S}}$, store $(i, \mathcal{M}_i, \mathsf{state}'_{\mathcal{M}}, \mathsf{svt}_i, 0, \emptyset)$ in L_{usr};
    return $(\gamma_{\mathcal{S}}, \mathsf{svt}_i)$;
  else return $\gamma_{\mathcal{S}}$.

Upon query $(\texttt{dishonestS\_join}, i, \gamma_{\mathcal{S}})$:
  Retrieve $(i, \perp, \perp, \mathsf{state}_{\mathcal{M}}, 1)$ from L_{join}, ;
  If not find add $(i, \perp, \perp, \xi, 1)$ in L_{join} and **return** ;
  $(\gamma_{\mathcal{M}}, \mathsf{state}'_{\mathcal{M}}) \leftarrow \mathcal{M}.\mathsf{Join}(\mathsf{gpk}, \mathsf{state}_{\mathcal{M}}, \gamma'_{\mathcal{S}})$;
  Store $(i, \perp, \perp, \mathsf{state}'_{\mathcal{M}}, 1)$ in L_{join};
  if $\gamma_{\mathcal{M}} = \texttt{concluded}$ **then** store $(i, \perp, \mathsf{state}'_{\mathcal{M}}, \perp, 1, \emptyset)$ in L_{usr};
  return $\gamma_{\mathcal{M}}$.

Upon query $(\texttt{sign}, i, \mathsf{bsn}, M, \mathsf{SigRL})$:
  Retrieve $(i, \mathcal{M}_i, \mathsf{state}_i, \mathsf{svt}_i, b, L_i) \in L_{usr}$;
  if $b = 0$ **then** // malicious $\mathcal{M}$
    $(\mathsf{state}'_i, \sigma) \leftarrow \mathsf{Sig}(\mathcal{M}_i, \mathsf{state}_i, \mathsf{svt}_i, \mathsf{bsn}, M, \mathsf{SigRL})$;
    Update $(i, \mathcal{M}_i, \mathsf{state}'_i, \mathsf{svt}_i, b, L_i \cup \{\langle \mathsf{bsn}, M, \sigma\rangle\})$;
    return σ
  else // malicious $\mathcal{S}$
    $(\sigma, \pi_\sigma) \leftarrow \mathcal{M}.\mathsf{Sig}(\mathsf{state}_i, \mathsf{bsn}, M, \mathsf{SigRL})$;
    return (σ, π_σ);

Upon query $(\texttt{corrupt}, i)$:
  Retrieve $(i, \mathcal{M}_i, \mathsf{state}_i, \mathsf{svt}_i, 0, L_i)$ from L_{usr}; move tuple from L_{usr} to L_{cor};
  return $(\mathsf{state}_i, \mathsf{svt}_i)$.

Fig. 3: Subversion-resilient non-frameability experiment.

to the signing oracle. 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.

3.3 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 $\mathsf{crs} \xleftarrow{\$} \mathsf{Init}(1^\lambda)$, any label $L \in \mathcal{L}$, and any $(x, w) \in \mathcal{R}$, we have $\mathsf{V}(\mathsf{crs}, L, x, \mathsf{P}(\mathsf{crs}, L, x, w)) = 1$.

Definition 8 (Adaptive composable perfect zero-knowledge). *A NIZK $\mathcal{NIZK}$ for relation $\mathcal{R}$ satisfies adaptive composable perfect zero-knowledge if the following properties hold:*

(i) There exists an algorithm $\overline{\mathsf{Init}}_{zk}$ that outputs crs, and a simulation trapdoor $\mathsf{tp_s}$ such that for any sequence $\{\mathsf{bgp}_\lambda \leftarrow \mathcal{G}(1^\lambda)\}_{\lambda \geq 0}$ and for any PPT distinguisher D

$$\begin{aligned} |\Pr[\mathsf{D}(\mathsf{crs}) = 1 : (\mathsf{crs}, \mathsf{tp_s}) \xleftarrow{\$} \overline{\mathsf{Init}}_{zk}(\mathsf{bgp}_\lambda)] \\ - \Pr[\mathsf{D}(\mathsf{crs}) = 1 : \mathsf{crs} \xleftarrow{\$} \mathsf{Init}(\mathsf{bgp}_\lambda)]| \in \mathsf{negl}(\lambda) \end{aligned}$$

(ii) There exists a PPT simulator $\mathcal{S}$ such that for any $(x, w) \in \mathcal{R}$, $L \in \mathcal{L}$ and all $(\mathsf{crs}, \mathsf{tp_s}) \xleftarrow{\$} \overline{\mathsf{Init}}_{zk}(\mathsf{bgp})$, the proofs generated via $\pi \xleftarrow{\$} \mathcal{S}(\mathsf{tp_s}, L, x)$ and $\pi \xleftarrow{\$} \mathsf{P}(\mathsf{crs}, L, x, w)$ are identically distributed.

Definition 9 (Adaptive extractable soundness). *A NIZK $\mathcal{NIZK}$ for relation $\mathcal{R}$ is* adaptive extractable sound *(Ext) if the following properties hold:*

(i) There exists an algorithm $\overline{\mathsf{Init}}_{snd}$ that outputs crs and an extraction trapdoor $\mathsf{tp_e}$ such that for any sequence $\{\mathsf{bgp}_\lambda \leftarrow \mathcal{G}(1^\lambda)\}_{\lambda \geq 0}$ and for any PPT distinguisher D

$$\begin{aligned} |\Pr[\mathsf{D}(\mathsf{crs}) = 1 : (\mathsf{crs}, \mathsf{tp_s}) \xleftarrow{\$} \overline{\mathsf{Init}}_{snd}(\mathsf{bgp}_\lambda)] \\ - \Pr[\mathsf{D}(\mathsf{crs}) = 1 : \mathsf{crs} \xleftarrow{\$} \mathsf{Init}(\mathsf{bgp}_\lambda)]| \in \mathsf{negl}(\lambda) \end{aligned}$$

(ii) There exists a PPT algorithm $\mathcal{E}(\mathsf{tp_e}, x, \pi)$ such that every PPT adversary $\mathcal{A}$:

$$\mathbf{Adv}^{\texttt{ext-sound}}_{\mathcal{A},\mathcal{NIZK},\mathcal{E}}(\lambda) := \Pr\left[\mathbf{Exp}^{\texttt{ext-sound}}_{\mathcal{NIZK},\mathcal{A},\mathcal{E}}(\lambda) = 1\right] \in \mathsf{negl}(\lambda)$$

where the experiment is defined in Fig. 4.

Malleable and Re-Randomizable NIZKs. We use the definitional framework of Chase et al [16] for malleable proof systems. For simplicity of the exposition we consider only unary transformations (see the aforementioned paper for more details). Let $T = (T_x, T_r)$ be a pair of efficiently computable functions, that we refer as a *transformation*.

Definition 10 (Admissible transformations [16]). *An efficient relation $\mathcal{R}$ is* closed *under a transformation $T = (T_x, T_w)$ if for any $(x, w) \in \mathcal{R}$ the pair $(T_x(x), T_w(w)) \in \mathcal{R}$. If $\mathcal{R}$ is closed under T then we say that T is* admissible *for $\mathcal{R}$. Let $\mathcal{T}$ be a set of transformations. If for every $T \in \mathcal{T}$, T is admissible for $\mathcal{R}$, then $\mathcal{T}$ is an* allowable *set of transformations.*

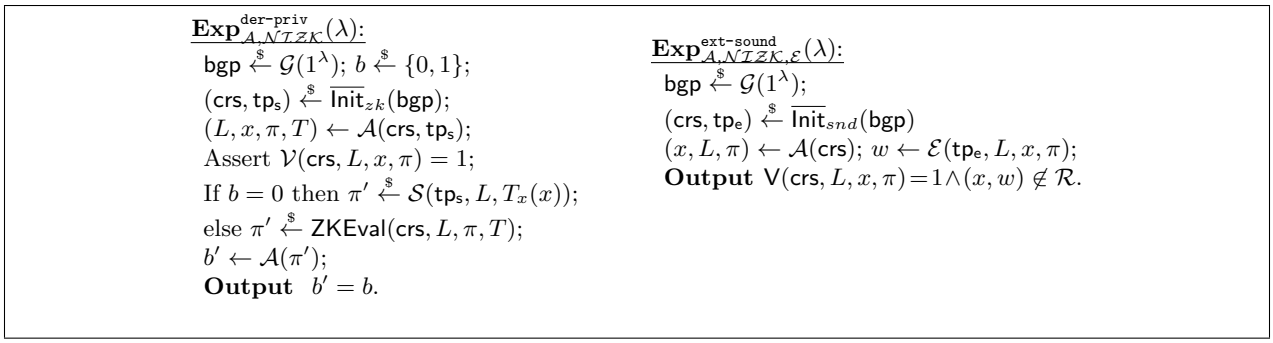

Fig. 4: The security experiments for the strong derivation privacy and adaptive extractable soundness.

Definition 11 (Malleable NIZK [16]). *Let $\mathcal{NIZK} = (\mathsf{Init}, \mathsf{P}, \mathsf{V})$ be a NIZK for a relation $\mathcal{R}$. Let $\mathcal{T}$ be an allowable set of transformations for $\mathcal{R}$. The proof system $\mathcal{NIZK}$ is* malleable with respect to $\mathcal{T}$ *if there exists an PPT algorithm* ZKEval *that on input $(\mathsf{crs}, L, (x, \pi), T)$, where $T \in \mathcal{T}$, L is a label and $\mathcal{V}(\mathsf{crs}, L, x, \pi) = 1$, outputs a valid proof π' for the statement $x' = T_x(x)$.*

For malleable NIZKs one can define the property that one should not distinguish between "freshly" generated proofs and derived ones. This property is formalized with the notion of *derivation privacy*.

Definition 12. *Let $\mathcal{NIZK} = (\mathsf{Init}, \mathsf{P}, \mathsf{V}, \mathsf{ZKEval})$ be a malleable NIZK argument for a relation $\mathcal{R}$ and an allowable set of transformations $\mathcal{T}$. We say that $\mathcal{NIZK}$ is* strong derivation private *if for any PPT adversary $\mathcal{A}$ we have that*

$$\mathbf{Adv}^{\mathtt{der\text{-}priv}}_{\mathcal{A},\mathcal{NIZK}}(\lambda) := \left| \Pr\left[\mathbf{Exp}^{\mathtt{der\text{-}priv}}_{\mathcal{A},\mathcal{NIZK}}(1^\lambda) = 1\right] - \tfrac{1}{2} \right| \in \mathsf{negl}(\lambda)$$

where $\mathbf{Exp}^{\mathtt{der\text{-}priv}}$ is the game described in Fig. 4. Moreover, we say that $\mathcal{NIZK}$ is perfectly strong derivation private *(resp.* statistically strong derivation private*) when for any (possibly unbounded) adversary the advantage above is 0 (resp. negligible).*

Re-randomizable NIZKs. First we notice that the derivation privacy property implicitly says that proofs are re-randomized (since outputs of ZKEval are indistinguishable from freshly generated proofs). 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

In this section we describe our construction of a subversion-resilient EPID. We start by providing a high-level explanation of our technique, next we describe the scheme, discuss how to instantiate it efficiently, and prove its security.

An Overview of Our Scheme. We elaborate further on the overview from Sec. 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 Sec. 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 works over bilinear groups generated by a generator $\mathcal{G}$, and itmakes 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:

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

  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:

$$\mathcal{R}_{\mathsf{com}} := \{([\mathbf{h}]_1, [t]_1),\ [\mathbf{y}]_2 : [t]_\mathsf{t} = e([\mathbf{h}]_1, [\mathbf{y}]_2)\}$$

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

  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.

$\mathsf{Init}(1^\lambda) \to \mathsf{pub}$**:** Generate description of a type-3 bilinear group $\mathsf{bgp} \xleftarrow{\$} \mathcal{G}(1^\lambda)$, the common reference string $\mathsf{crs}_{\mathsf{svt}} \xleftarrow{\$} \mathcal{NIZK}_{\mathsf{svt}}.\mathsf{Init}(\mathsf{bgp})$, and sample $\mathbf{h} \xleftarrow{\$} \mathbb{Z}_p^2$. Output $\mathsf{pub} = (\mathsf{bgp}, \mathsf{crs}_{\mathsf{svt}}, [\mathbf{h}]_1)$[19]

$\mathsf{Setup}(\mathsf{pub}) \to (\mathsf{gpk}, \mathsf{isk})$**:** sample $(\mathsf{sk}_{sp}, \mathsf{pk}_{sp}) \xleftarrow{\$} \mathsf{KGen}_{sp}(\mathsf{bgp})$, 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 $id \xleftarrow{\$} \{0,1\}^\lambda$, sends id to $\mathcal{S}$ and $\mathcal{M}$. Parties compute $\mathsf{crs}_{\mathsf{com}} \leftarrow \mathsf{J}(id)$.
2. $\mathcal{M}$ samples $y_{0,\mathcal{M}}, c_\mathcal{M} \xleftarrow{\$} \mathbb{Z}_p$, 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 $\mathcal{M}$.

[19] 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].

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 $y_{1,\mathcal{M}} \xleftarrow{\$} \mathbb{Z}_p$ 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 $y_{1,\mathcal{S}} \xleftarrow{\$} \mathbb{Z}_p$ 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.

$\mathsf{Sig}(\mathsf{gpk}, \mathsf{sk}, \mathsf{svt}, \mathsf{bsn}, M, \mathsf{SigRL}) \to (\sigma, \pi_\sigma)$**:** 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 verication 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.

Efficiency. A suitable SPS for our construction of SR-EPID is the one in [17] (see Section 5.3 of the reference). It features signatures in $\mathbb{G}_1^2 \times \mathbb{G}_2$ and verification requires 2 PPEs and 6 pairings. To evaluate efficiency of our construction we look at $\mathcal{R}_{\mathsf{sign}}$ where we have $\mathsf{SigRL} = \{[\mathbf{b}_i]_1\}_{i=1}^r$, $\mathsf{gpk} = ([\mathbf{h}]_1, \mathsf{pk}_{sp})$, and $\mathbf{y} = (y_0, y_1)^{\mathsf{T}}$ and we rely on [20] . We are committing to $[t]_1, \sigma_{sp}, [\mathbf{y}]_2$: since σ_{sp} has size 3 and $[\mathbf{y}]_2$ has size 2, we have a total of 6 variables, thus the commitment is composed by 12 elements (since we are using $\ell = 2$). Looking at the proof, we have the following relations: $[\mathbf{b}]_1 \in span([1, y_0]_1^{\mathsf{T}})$ is a linear equation, so the proof has size $k + 1 = 2$; $[t]_{\mathsf{t}} = [\mathbf{h}^{\mathsf{T}} \cdot \mathbf{y}]_{\mathsf{t}}$ is a PPE and hence requires $\ell \cdot (k + 1) = 4$ group elements. Moreover, $\mathsf{Ver}_{sp}(\mathsf{pk}_{sp}, [t]_1, \sigma_{sp}) = 1$ is the verification of a signature which requires 2 PPEs to be verified,

for a total of 8 elements. Finally, $\forall i : [\mathbf{b}_i]_1 \notin span([1, y_0]_1^\mathsf{T})$ consists in n linear equations, for a total of $2n$ elements (2 elements for each signature in SigRL). It follows that the resulting SR-EPID has signatures of size $28 + 2n$ group elements, where n is the number of signatures in SigRL. The original EPID scheme [9] has signatures of size $8 + 5n$. We note that signatures produced by our scheme, as well as the ones in related works [9,11], have sizes that are linear in the size of the revocation list SigRL. This is because each signature σ carries a proof that the private key used to produce σ is different from any of the keys used to produce any of the signatures in SigRL. We leave finding a scheme with signatures *sublinear* in $|\mathsf{SigRL}|$ (e.g., constant or logarithmic) as an interesting open problem. Nevertheless, note that in practical scenarios, if SigRL becomes too large it may be cheaper to have non-revoked members re-join the group.

4.1 Proof of Security

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.*

We first give a proof sketch. 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 $\mathbb{Z}_q^2$.[20] 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 (from $\mathbf{H}_0$ to $\mathbf{H}_6$ in the formal proof) 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 $\mathcal{Q}_{sp}$. Slightly more technically, we can reduce this to the binding property[21] of the Pedersen's commitment scheme that we use.

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

[21] In the formal proof, we rely directly on the XDH assumption (see hybrid $\mathbf{H}_7$).

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 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.

Proof. Given an adversary $\mathcal{A}$ for the unforgeability game, we assume, w.l.g. that if the adversary sends the query $(\mathtt{sign}, *, \mathsf{bsn}, M, *)$ for some bsn, M then the adversary has already queried the random oracle H on the tuple (bsn, M). Notice that this assumption is without loss of generality[22].

Given a PPT adversary $\mathcal{A}$ we define the extractor $\mathcal{E}$. Let $\mathcal{E}_{\mathsf{com}}$ be the extractor for the $\mathcal{NIZK}_{\mathsf{com}}$. The extractor $\mathcal{E}$ is defined below:

Extractor $\mathcal{E}(\cdot)$:

- At the first call initialize the database $D_{\mathtt{RO}}$ as empty and generates the group parameter $\mathsf{bgp} \xleftarrow{\$} \mathcal{G}(1^\lambda)$.
- Upon input $(\mathtt{RO}, \mathsf{H}, x)$ check if $(\mathsf{H}, x, y, \bot)$ exists in $D_{\mathtt{RO}}$ and if so return y, else sample $y \xleftarrow{\$} \{0,1\}^\lambda$, add the tuple $(\mathsf{H}, x, y, \bot)$ into the database and return y.
- Upon input $(\mathtt{RO}, \mathsf{J}, x)$ check if $(\mathsf{J}, x, \mathsf{crs}, \mathsf{tp_e})$ exists in $D_{\mathtt{RO}}$ and if so return crs_1, else sample $\mathsf{crs}, \mathsf{tp_e} \xleftarrow{\$} \mathcal{NIZK}_{\mathsf{com}}.\overline{\mathsf{Init}}_{snd}(\mathsf{bgp})$, add the tuple $(\mathsf{J}, x, \mathsf{crs}, \mathsf{tp_e})$ into the database and return crs_1.
- Upon input $(\mathtt{extract}, \tau)$ parse the transcript τ as described by the messages sent in the join protocol and find the value id, lookup for the tuple $(\mathsf{J}, id, \mathsf{crs}, \mathsf{tp_e})$ into the database $D_{\mathtt{RO}}$, and if it does not exist then it output $\bot$. Else, find the message $([t]_1, \pi_{\mathcal{S}})$ from $\mathcal{S}$, run the extractor $[\mathbf{y}]_2 \leftarrow \mathcal{E}_{\mathsf{com}}(\mathsf{tp_e}, \pi_{\mathcal{S}})$, find the message σ_{sp} sent from the issuer $\mathcal{I}$, and output $\mathsf{tk} = ([t]_1, \sigma_{sp}, [\mathbf{y}]_2)$.

We define the $\mathsf{CheckTK}$ algorithm. The algorithm given in input gpk, sk and tk parses sk as $([t]_1, [\sigma_{sp}]_1, \mathbf{y})$ and check if $\mathsf{tk} = ([t]_1, [\sigma_{sp}]_1, [\mathbf{y}]_2)$. We define the $\mathsf{CheckSig}$ algorithm. The algorithm given in input gpk, tk and a signature σ, parses $\mathsf{tk} = ([t]_1, [\sigma_{sp}]_1, [y_0, y_1]_2)$ and $\sigma = ([c_0, c_1]_1, \pi)$ and return 1 if and only if $e([c_0]_1, [y_0]_2) = e([c_1]_1, [1]_2)$. The property 1 is obviously true, in fact, the function that map $x \in \mathbb{Z}_p$ to $[x]_2 \in \mathbb{G}$ is injective, moreover, the property 2 is true too, in fact, the step (3) of the verification algorithm checks that $[c_0]_1 y_0 = [c_1]_1$, which is the same of verifying $e([c_0]_1, [y_0]_2) = e([c_1]_1, [1]_2)$.

In the following we define two sequences of hybrid experiments. In the first sequence of hybrids experiment we consider the random variable $view_{\mathcal{A},i}$ that is the view of the adversary $\mathcal{A}$ in the hybrid experiment $\mathbf{H}'_i$. Recall that Def. 5 also requires to compare the view of the adversary in the unforgeability experiment with the *dummy extractor* and the same view with the extractor defined above.

Let $\mathbf{H}'_0(\lambda) := \mathbf{Exp}^{\mathtt{unf}}_{\mathcal{A},\tilde{\mathcal{E}},\Pi}(\lambda)$, namely the experiment run with the dummy extractor that answers the random-oracle queries as a random oracle would do.

[22] Given an adversary $\mathcal{A}'$ that does not respect this rule, we can always define a new adversary $\mathcal{A}$ that runs internally $\mathcal{A}'$ and whenever it receives a message $(\mathtt{sign}, *, \mathsf{bsn}, M, *)$ first query the RO H and then forward the signing query.

Hybrid $\mathbf{H}'_1(\lambda)$. Let $\mathbf{H}'_1(\lambda) := \mathbf{Exp}^{\mathtt{unf}}_{\mathcal{A},\mathcal{E}',\Pi}(\lambda)$, where $\mathcal{E}'$ is the same as $\mathcal{E}$, as defined above, but where when it is called upon input $(\mathtt{extract}, \tau)$ simply it returns $\perp$.

Lemma 1. *For any PPT* D *we have* $|\Pr[\mathsf{D}(view_{\mathcal{A},1}) = 1] - \Pr[\mathsf{D}(view_{\mathcal{A},0}) = 1]| \in \mathsf{negl}(\lambda)$.

Proof. The proof of the lemma follows by the composable zero-knowledge property. Details omitted.

Hybrid $\mathbf{H}'_2(\lambda)$. Let $\mathbf{H}'_2(\lambda) := \mathbf{Exp}^{\mathtt{unf}}_{\mathcal{A},\mathcal{E},\Pi}(\lambda)$.

Lemma 2. *For any PPT* D *we have* $\Pr[\mathsf{D}(view_{\mathcal{A},2}) = 1] = \Pr[\mathsf{D}(view_{\mathcal{A},1}) = 1]$.

Proof. Notice that the difference between the two hybrids is that in the second the extractor additionally computes the tokens tk. However, the tokens are never add in the view of the adversary.

By the two lemmas above and the triangular inequality we already have the extra condition of the unforgeability in the ROM (Def 5).

In the next sequence of hybrids we will gradually modify the winning condition of the adversary. Recall that in the unforgeability experiment of Fig 2, we defined the winning condition of the adversary to be $W := ((1) \wedge (2) \wedge (3)) \vee (4)$. For notation, we call W_i the winning condition in the hybrid experiment $\mathbf{H}_i$, we set $W_0 := W$ and, whenever we don't mention it explicitly, we set $W_{i+1} := W_i$. Let $\mathbf{H}_0(\lambda) := \mathbf{Exp}^{\mathtt{unf}}_{\mathcal{A},\mathcal{E},\Pi}(\lambda)$.

Hybrid $\mathbf{H}_1(\lambda)$. Let $\mathbf{H}_1$ be the same as $\mathbf{H}_0$ but where the winning condition is changed. In particular, the condition (4) is omitted[23], thus the winning condition is $W_1 := (1) \wedge (2) \wedge (3)$.

Lemma 3. $|\Pr[\mathbf{H}_1(\lambda) = 1] - \Pr[\mathbf{H}_0(\lambda) = 1]| \in \mathsf{negl}(\lambda)$.

Proof. We reduce to the adaptive knowledge soundness of the $\mathcal{NIZK}_{\mathsf{com}}$. Moreover we rely on the perfect correctness of $\mathcal{NIZK}_{\mathsf{sign}}$ and the perfect correctness of the signature scheme $\mathcal{SS}$. The extractor $\mathcal{E}$ computes $[\mathbf{y}]_2$ using the knowledge extractor of $\mathcal{NIZK}_{\mathsf{com}}$ and output $\mathsf{tk} = ([t]_1, \sigma_{sp}, [\mathbf{y}]_2)$, since $[t]_1, \sigma_{sp}$ are generated by the issuer $\mathcal{I}$ they form a valid message-signature pair. Suppose that exists $\mathsf{sk} \in \mathsf{PrivRL}^*$ linked to tk, therefore $\mathsf{sk} = ([t]_1, \sigma_{sp}, \mathbf{y})$ and that $\mathsf{CheckSK}(\mathsf{gpk}, \mathsf{sk}) = 0$, therefore, either $[\mathbf{h}^\top \cdot \mathbf{y}]_T \neq [t]_T$, but this would violate the adaptive knowledge soundness of $\mathcal{NIZK}_{\mathsf{com}}$, or the latter holds but, the signature $([\mathbf{c}]_1, \pi)$ for a random message M does not verify, but this would violate either the correctness of $\mathcal{NIZK}_{\mathsf{sign}}$ or the correctness of $\mathcal{SS}$.

Hybrid $\mathbf{H}_2(\lambda)$. Let $\mathbf{H}_2$ be the same as $\mathbf{H}_1$ but where the winning condition is changed. In particular, let q_{H} be an upper bound on the number of oracle queries made by $\mathcal{A}$ to H, w.l.g. we assume the adversary does not query twice the RO with the same input. The hybrid samples an index $i^* \xleftarrow{\$} [q_{\mathsf{H}}]$ and a common-reference string $\mathsf{crs}^*, \mathsf{tp_e}^* \xleftarrow{\$} \mathcal{NIZK}_{\mathsf{sign}}.\overline{\mathsf{Init}}_{snd}(\mathsf{bgp})$. At the i^*-th call to the random oracle H it sets the output of the random oracle to be crs^*. Moreover, consider the condition (5) defined as:

> (bsn^*, M^*) (the basename-message tuple of the forgery) is queried to the random oracle H at the i^*-th query.

The new winning condition is $W_2 := W_1 \wedge (5)$.

Lemma 4. $\Pr[\mathbf{H}_2(\lambda) = 1] \geq \Pr[\mathbf{H}_1(\lambda) = 1]/q_{\mathsf{H}} - \mathsf{negl}(\lambda)$.

Proof. First consider the intermediate hybrid $\mathbf{H}_{2,1}$ equal to $\mathbf{H}_2$ (we sample crs^* and assign it to the i^*-th query to the random oracle), but where we do not change the winning condition. By property (i) of Def. 9 (extractable sound CRSs are indistinguishable from random strings) we know that $|\Pr[\mathbf{H}_{2,1}(\lambda) = 1] - \Pr[\mathbf{H}_1(\lambda) = 1]| \in \mathsf{negl}(\lambda)$.

Notice that $\Pr[\mathbf{H}_2(\lambda) = 1] = \Pr[\mathbf{H}_{2,1}(\lambda) = 1 \wedge (5)] = \Pr[\mathbf{H}_{2,1}(\lambda)] \Pr[(5)]$, in fact, the view of the adversary is independent of the random variable i^*. Moreover, the probability of (5) is $1/q_{\mathsf{H}}$.

[23] Recall that condition (4) states that $\exists \mathsf{sk} \in \mathsf{PrivRL}^*$ and $\mathsf{tk} \in R$, where R is the set of secret-key tokens of the corrupted users, such that $\mathsf{CheckTK}(\mathsf{gpk}, \mathsf{sk}, \mathsf{tk}) = 1$ but $\mathsf{CheckSK}(\mathsf{gpk}, \mathsf{sk}) = 0$.

Hybrid $\mathbf{H}_3(\lambda)$. Let $\mathbf{H}_3$ be the same as $\mathbf{H}_2$ but where the winning condition of the adversary is changed. In particular, after the adversary outputs its forgery the hybrid additionally computes $([t^*]_1, [\sigma^*_{sp}]_1, [\mathbf{y}^*]_2) \leftarrow \mathcal{E}_{\mathsf{sign}}(\mathsf{tp_e}^*, \pi^*)$, where $(\pi^*, [\mathbf{c}^*]_1)$ is the forged signature. The winning condition is changed to $W_3 := W_2 \wedge (6)$ where (6) is defined as:

> Check that $\mathsf{Ver}_{sp}(\mathsf{pk}_{sp}, [t^*]_1, [\sigma^*_{sp}]_1) = 1$ and $[t^*]_\mathtt{t} = [\mathbf{h}^\mathsf{T} \cdot \mathbf{y}^*]_\mathtt{t}$ and for any $([c'_0, c'_1]_1, \pi') \in \mathsf{SigRL}^*$ we have $[c'_1]_\mathtt{t} \neq [c'_0 y^*_0]_\mathtt{t}$.

Lemma 5. $\Pr[\mathbf{H}_3(\lambda) = 1] = \Pr[\mathbf{H}_2(\lambda) = 1]$.

Proof. We reduce to adaptive extractable soundness of $\mathcal{NIZK}_{\mathsf{sign}}$. Notice that by the Shoup's difference lemma we need to bound the probability of the event $W_2 \wedge \neg(6)$, namely that the hybrid $\mathbf{H}_2$ outputs 1 but $\mathbf{H}_3$ does not. The event $W_2 \wedge \neg(6)$ implies that the proof π^* for the statement $[\mathbf{c}^*]_1, \mathsf{SigRL}^*$ and with label y^* does verify but (by the condition $\neg(6)$) either the $\mathsf{Ver}_{sp}(\mathsf{gpk}, [t^*]_1, \sigma_{sp}) = 0$ or $[t]_\mathtt{t} = [\mathbf{h}^\mathsf{T} \cdot \mathbf{y}]_\mathtt{t}$ exists $([c'_0, c'_1]_1, \pi') \in \mathsf{SigRL}^*$ where $[c'_1]_\mathtt{t} = [c'_0 y^*_0]_\mathtt{t}$, which violates the soundness of the proof system.

Hybrid $\mathbf{H}_4(\lambda)$. Let $\mathbf{H}_4$ be the same as $\mathbf{H}_3$ but where we program differently the random oracle H. In particular, upon the i-th query $(\mathtt{R0}, \mathsf{H}, x)$ where $i \neq i^*$ sample $\mathsf{crs}, \mathsf{tp_s} \xleftarrow{\$} \mathcal{NIZK}_{\mathsf{sign}}.\overline{\mathsf{Init}}_{zk}(\mathsf{bgp})$ and set the tuple $(\mathsf{H}, x, \mathsf{crs}, \mathsf{tp_s})$ into the database $D_{\mathtt{R0}}$.

Lemma 6. $|\Pr[\mathbf{H}_4(\lambda) = 1] - \Pr[\mathbf{H}_3(\lambda) = 1]| \in \mathsf{negl}(\lambda)$.

Proof. The proof of the lemma follows by property (i) of Def. 8 (adaptive composable perfect zero-knowledge) of $\mathcal{NIZK}_{\mathsf{sign}}$.

Hybrid $\mathbf{H}_5(\lambda)$. Let $\mathbf{H}_5$ be the same as $\mathbf{H}_4$ but where the queries $(\mathtt{sign}, *, *, *)$ are answered in a different way. Let $\mathcal{S}_{\mathsf{sign}}$ be the zero-knowledge simulator of $\mathcal{NIZK}_{\mathsf{sign}}$. Upon query $(\mathtt{sign}, i, \mathsf{bsn}, M, \mathsf{SigRL})$ where $(i, \mathcal{M}_i, \mathsf{state}_i, \mathsf{svt}_i, \mathsf{tk}_i) \in L_{urs}$ and $\mathsf{svt}_i \neq \perp$ (namely, the sanitizer $\mathcal{S}$ is honest) and $(\mathsf{bsn}, M) \neq (\mathsf{bsn}^*, M^*)$ (where (bsn^*, M^*) is the i^*-th query to the random oracle H), the hybrid computes $\sigma = ([\mathbf{c}]_1, \pi), \mathsf{state}'_i \leftarrow \mathcal{M}_i(\mathsf{state}_i, \mathsf{bsn}, M, \mathsf{SigRL})$, retrieve the tuple $(\mathsf{H}, (\mathsf{bsn}, M, \mathsf{crs}, \mathsf{tp_s})$ from $D_{\mathtt{R0}}$ (or create it if it does not exist), computes $\tilde{\pi} \leftarrow \mathcal{S}(\mathsf{tp_s}, ([\mathbf{c}]_1, \mathsf{SigRL}))$ and outputs $([\mathbf{c}]_1, \tilde{\pi})$.

Lemma 7. $|\Pr[\mathbf{H}_5(\lambda) = 1] - \Pr[\mathbf{H}_4(\lambda) = 1]| \in \mathsf{negl}(\lambda)$.

Proof. We reduce to the strong derivation privacy of $\mathcal{NIZK}_{\mathsf{sign}}$. As the reduction is almost straight forward, here we just give a sketch. Let q_{sign} be an upper bound on the number of signing queries sessions the adversaries perform. Due to strong derivation privacy we know that, for each query, we have

$$\mathbf{Adv}^{\mathtt{der\text{-}priv}}_{\mathcal{A},\mathcal{NIZK}}(\lambda) := \left|\Pr\left[\mathbf{Exp}^{\mathtt{der\text{-}priv}}_{\mathcal{A},\mathcal{NIZK}}(1^\lambda) = 1\right] - \tfrac{1}{2}\right| = \epsilon(\lambda) \in \mathsf{negl}(\lambda).$$

This means that $|\Pr[\mathbf{H}_5(\lambda) = 1] - \Pr[\mathbf{H}_4(\lambda) = 1]| \leq q_{sign} \cdot \epsilon(\lambda) \in \mathsf{negl}(\lambda)$.

Hybrid $\mathbf{H}_6(\lambda)$. Let $\mathbf{H}_6$ be the same as $\mathbf{H}_5$ but where, for any signature produced by honest platforms with a subverted machine, we control explicitly that $[\mathbf{c}]_1$ is of the right form. Specifically, upon query $(\mathtt{sign}, i, \mathsf{bsn}, M, \mathsf{SigRL})$ where $(i, \mathcal{M}_i, \mathsf{state}_i, \mathsf{svt}_i, *) \in L_{urs}$ and $\mathsf{svt}_i \neq \perp$ (namely, the sanitizer $\mathcal{S}$ is honest), the hybrid computes $\sigma = ([c_0, c_1]_1, \pi), \mathsf{state}'_i \leftarrow \mathcal{M}_i(\mathsf{state}_i, \mathsf{bsn}, M, \mathsf{SigRL})$, and return $\perp$ to the adversary if $e([c_1]_1, [1]_2) \neq e([c_0]_1, [y_0]_2)$.

Lemma 8. $\Pr[\mathbf{H}_6(\lambda) = 1] = \Pr[\mathbf{H}_5(\lambda) = 1]$.

Proof. Recall that $\mathsf{svt}_i = [c, cy_0]_1$, and that the proof π_σ proves that $(\mathsf{svt}_i, [\mathbf{c}]_1)$ are of form $[x, xy, z, zy]$ for $x, y, z \in \mathbb{Z}_p$. Therefore, if the hybrid $\mathbf{H}_6$ outputs $\perp$ but $\mathbf{H}_5$ does not, then the proof π_σ verifies but $\mathbf{c}$ does not lie in the subspace spanned by svt_i, therefore breaking the adaptive perfect soundness of $\mathcal{NIZK}_{\mathsf{svt}}$.

Let $\mathcal{Q}_{sp} := \{[t]_1 : (*, *, *, *, ([t]_1, \sigma_{sp}, [\mathbf{y}]_2)) \in L_{usr}\}$. Namely, the set of signatures of the structure-preserving signature scheme computed by the challenger of the game.

Hybrid $\mathbf{H}_7(\lambda)$. Let $\mathbf{H}_7$ be the same as $\mathbf{H}_6$ but where the winning condition is changed. Specifically, consider the condition (7) defined below:

$$(*, \mathsf{bsn}^*, M^*, *) \notin L_{msg} \vee [t^*]_1 \notin \mathcal{Q}_{sp}.$$

The winning condition is changed to $W_7 := W_6 \wedge (7)$.

Lemma 9. $|\Pr[\mathbf{H}_7(\lambda) = 1] - \Pr[\mathbf{H}_6(\lambda) = 1]| \in \mathsf{negl}(\lambda)$.

Proof. We need to bound the probability of the event $\mathtt{Bad} := W_6 \wedge (*, \mathsf{bsn}^*, M^*, *) \in L_{msg} \wedge [t^*]_1 \in \mathcal{Q}_{sp}$, We reduce to the SXDH assumption over $\mathbb{G}_1$. Clearly, this problem is equivalent to SXDH (just permute the second and forth coordinates of the challenge). Consider the following reduction:

Adversary $\mathcal{B}(\mathsf{bgp}, [1, x, y, z]_1)$:

1. **(Install the Challenge in the parameters.)** Run the adversary $\mathcal{A}$ and simulate the hybrid $\mathbf{H}_7$. Specifically, compute the public parameter as the hybrid does but using the bgp given in input to $\mathcal{B}$ and setting $[\mathbf{h}]_1 := [x, y]_1$.
2. Simulate the random oracle K by making sure that no collisions will appear.
3. Eventually the adversary outputs its forgery $(\mathsf{bsn}^*, M^*, \sigma^*, \mathsf{PrivRL}^*, \mathsf{SigRL}^*)$, let $([t^*]_1, [\sigma^*_{sp}]_1, [\mathbf{y}^*]_2)$ be the extracted witness and $[\mathbf{y}^*]_2 = [y_0^*, y_1^*]_2$.
4. Let $[\mathbf{y}]_2$ such that $(*, *, *, *, ([t^*]_1, *, [\mathbf{y}]_2)) \in L_{usr}$ and $(*, \mathsf{bsn}^*, M^*, \sigma) \in L_{msg}$. Compute $[d_0]_2 := [y_0^* - y_0]_2$ and $[d_1]_2 := [y_1^* - y_1]_2$ and return 1 if and only if $e([z]_1, [d_1]_2) = -e([h_0]_1, [d_0]_2)$.

First notice that the simulation given by $\mathcal{B}$ is statistically close to the hybrid experiment $\mathbf{H}_7$. In fact, the only difference is that in $\mathbf{H}_7$ there might be collisions in K, however the probability of such event is negligible in the security parameter.

Let parse $\sigma^* = ([\mathbf{c}^*]_1, \pi^*)$ and $\sigma = ([\mathbf{c}]_1, \pi)$. When $\mathtt{Bad}$ happens, because of condition (3) then $[c_1^*] \neq [c_1]$ (the signatures are unlinkable), also, because of $(*, \mathsf{bsn}^*, M^*, *) \in L_{msg} \wedge [t^*]_1 \in \mathcal{Q}_{sp}$ there must exist $[\mathbf{y}]_2$ and σ as defined in step 4 of $\mathcal{B}$. Thus we have that $\mathbf{y} \neq \mathbf{y}^*$ and $t = \mathbf{h} \cdot \mathbf{y} = \mathbf{h} \cdot \mathbf{y}^*$. Therefore $h_0 d_0 + h_1 d_1 = 0$, thus if $z = xy = h_0 h_1$ then the pairing test $e([z]_1, [d_1]_2) = -e([h_0]_1, [d_0]_2)$ must hold, while if z is uniformly random in $\mathbb{Z}_p$ then the test hold with negligible probability.

Next, we define two different classes of adversaries. Let $\mathbb{A}_1$ be the class of adversaries such that the event $[t^*]_1 \in \mathcal{Q}_{sp}$ happens with noticeable probability in $\mathbf{H}_7$. Similarly, let $\mathbb{A}_2$ be the class of adversaries such that the same event happens with negligible probability in $\mathbf{H}_7$. The two classes partition the entire class of adversaries.

We now fork our hybrid argument in two. The first sequence is to argue the unforgeability for the adversaries from the class $\mathbb{A}_1$.

Hybrid $\mathbf{H}_8(\lambda)$. Let $\mathbf{H}_8$ be the same as $\mathbf{H}_7$ but where the winning condition is changed. Let q_{join} be a polynomial in λ that upper bounds the number of join that the adversary performs. Pick $j^* \xleftarrow{\$} [q_{join}]$ and change the winning condition to $W_8 := W_8 \wedge (8)$ where (8) is defined as described below:

> Check that $(j^*, *, *, *, ([t^*]_1, *, *)) \in L_{usr}$. Namely, the witness $[t^*]_1$ extracted from the proof π^* in the forged signature was signed by the issuer at the j^*-th join protocol, and the parties $\mathcal{S}_{j^*}, \mathcal{M}_{j^*}$ were not (both) corrupted.

Lemma 10. *For any $\mathcal{A} \in \mathbb{A}_1$ there is a polynomial p such that* $\Pr[\mathbf{H}_8(\lambda) = 1] \geq \Pr[\mathbf{H}_7(\lambda) = 1] / p(\lambda)$.

Proof. Let $p'(\lambda)$ be a polynomial such that $\Pr[[t^*]_1 \in \mathcal{Q}_{sp}] \geq 1/p'(\lambda)$. By the definition of $\mathbb{A}_1$, this polynomial exists. Notice that the condition (8) holds when $[t^*]_1 \in M$ and $[t^*]_1$ is the message signed by $\mathcal{I}$ (using $\mathcal{SS}$) at the j^*-th join session. In particular these two events are independent, so the probability that (7) holds is $1/q_{join} \cdot 1/p'(\lambda)$ which is noticeable in λ.

Hybrid $\mathbf{H}_9(\lambda)$. Let $\mathbf{H}_9$ be the same as $\mathbf{H}_8$ but where the random oracle J is programmed differently. Let $\mathcal{NIZK}_{\mathsf{com}}.\mathsf{Init}_{zk}$ be the zero-knowledge common-reference string generator for $\mathcal{NIZK}_{\mathsf{com}}$. In particular, when the challenger is queried with either $(\texttt{honest\_join}, j^*, *)$ or with $(\texttt{dishonestH\_join}, j^*, \mathcal{I}, \xi)$[24], the challenger picks a random $id^* \xleftarrow{\$} \{0,1\}^\lambda$ (we assume that id^* was not queried to J), computes $\mathsf{crs}, \mathsf{tp}_\mathsf{s} \leftarrow \mathcal{NIZK}_{\mathsf{com}}.\mathsf{Init}_{zk}(\mathsf{bgp})$ and set the entry $(\mathsf{J}, id^*, \mathsf{crs}, \mathsf{tp}_\mathsf{s})$ in the database $D_{\mathtt{RO}}$. Finally it outputs the message id^* as the first message of the issuer $\mathcal{I}$ in the join protocol.

Lemma 11. *For any* $\mathcal{A} \in \mathbb{A}_1$ $|\Pr[\mathbf{H}_9(\lambda) = 1] - \Pr[\mathbf{H}_8(\lambda) = 1]| \in \mathsf{negl}(\lambda)$.

Proof. We reduce to composable zero-knowledge property. Also notice that the probability that id^* was queried already to J is $q_{\mathtt{RO}}/2^\lambda$ where $q_{\mathtt{RO}}$ upper bounds the number of queries made to the RO.

Hybrid $\mathbf{H}_{10}(\lambda)$. Let $\mathbf{H}_{10}$ be the same as $\mathbf{H}_9$ but where the transcript output in the j^*-th join protocol is different. Let $\mathcal{S}_{\mathsf{com}}$ be the zero-knowledge simulator of $\mathcal{NIZK}_{\mathsf{com}}$. Upon query $(\texttt{honest\_join}, j^*, \mathcal{M})$, let τ be the transcript at the end of the execution of the join protocol, find in τ the message $([t]_1, \pi_\mathcal{S})$, compute $\tilde{\pi}_\mathcal{S} \leftarrow \mathcal{S}_{\mathsf{com}}(\mathsf{tp}_{\mathsf{s}_{\mathsf{com}}}, [t]_1)$ and set $\tilde{\tau}$ be the same as τ but where the message $([t]_1, \pi_\mathcal{S})$ is substituted with the message $([t]_1, \tilde{\pi}_\mathcal{S})$. Return $\mathsf{svt}_i, \tilde{\tau}$ to the adversary.

Lemma 12. *For any* $\mathcal{A} \in \mathbb{A}_1$ $\Pr[\mathbf{H}_{10}(\lambda) = 1] = \Pr[\mathbf{H}_9(\lambda) = 1]$.

Proof. The proof of the lemma follows by the strong derivation privacy of the $\mathcal{NIZK}_{\mathsf{com}}$. In particular, we can perform an hybrid argument over the number of execution of the join protocol with an honest sanitizer. The reduction is straight forward therefore omitted.

Hybrid $\mathbf{H}_{11}(\lambda)$. Let $\mathbf{H}_{11}$ be the same as $\mathbf{H}_{10}$ but where at the j^*-th join protocol, if the adversary plays with a subverted machine and an honest sanitizer, then we substitute the subverted machine with *well behaving* machine. Recall that, in the description of the join protocol the machine $\mathcal{M}$ sends three messages. Consider the machine $\tilde{M}$ that samples a random index $r \xleftarrow{\$} \{1,2,3,4\}$ and that executes the same code of the honest machine $\Pi.\mathcal{M}$ but that, if $r = 1$ it does not send the first message (or the message is invalid), if $r = 2$ it does not send the second message (or the message is invalid), if $r = 3$ does send the third message, and if $r = 4$ it completes the join protocol. If the adversary sends a query of the kind $(\texttt{honest\_join}, j^*, \mathcal{M}_i)$ then the hybrid executes the query with the machine $\tilde{\mathcal{M}}$ instead of $\mathcal{M}_i$.

Lemma 13. *For any* $\mathcal{A} \in \mathbb{A}_1$ *there is a polynomial* p *such that* $\Pr[\mathbf{H}_{11}(\lambda) = 1] \geq \Pr[\mathbf{H}_{10}(\lambda) = 1]/4$.

Proof. Let r' be the random variable that is 1 if the machine $\mathcal{M}_i$ does not send the first message (or the message is invalid), 2 if it does not send the second (or the message is invalid), 3 if it does not send the third message, and 4 otherwise.

We prove that for any assignment $l \in \{1,2,3,4\}$, $\Pr[\mathbf{H}_{10}|r = l] = \Pr[\mathbf{H}_9|r' = l]$. Notice that the joint distribution of the transcript between the honest sanitizer and the issuer in the join protocol and the value svt, conditioned on $r' = l$, it is the same either if the machine is $\mathcal{M}_i$ or $\tilde{\mathcal{M}}$. In fact, if $l = 1$ then both distributions are trivially equivalent (as no message was sent by $\mathcal{M}_i$ or $\tilde{\mathcal{M}}$). If $l = 2$ then the first message sent by the machine $\mathcal{M}_i$ is svt' which is arbitrarily distributed, while $\tilde{\mathcal{M}}$ would send an uniformly random tuple, notice there is no message between the sanitizer and the issuer and moreover the tuple svt produced by the honest sanitizer is uniformly distributed independently of the value svt' (as long as $\mathsf{svt}'_0, \mathsf{svt}'_1 \neq [0]_1$). If $l = 3$ then the first message of the transcript is $([t]_1, \tilde{\pi}_\mathcal{S})$ where the proof is simulated and therefore independent of the machine's message, and t is a uniformly chosen vector in the span of $(1, y_0)$. If $l = 4$ then the last message is a deterministic message (the message $\texttt{completed}$)) moreover by the Assumption 1 the machine $\mathcal{M}_i$ never aborts after the protocol join successfully completed.

Also, if $\mathbf{H}_9 = 1$ then the sanitizer $\mathcal{S}_i$ is honest, therefore all the signatures are re-randomized and for the correct key y_0. Specifically, let $([\mathbf{c}]_1, \pi)$ be a signature output by the challenger on query $(\texttt{sign}, j^*, M, \mathsf{SigRL})$,

[24] Recall that the first message of the protocol is sent by the issuer, however, in the security experiment the sessions are started with a first message from the adversary, we handle this assuming that the adversary, acting as the sanitizer in the join protocol, initiates the join protocol by sending an empty string ξ to $\mathcal{I}$.

the vector $[\mathbf{c}]_1$ is a function of K and y_0 (we used the soundness of the proof π_σ sent by the machine to the $\mathcal{S}_i$ to state this in $\mathbf{H}_6$), so independent of the machine's messages, moreover, by the change introduced in the hybrid $\mathbf{H}_5$ we simulate the proof π, which is therefore independent of the machine's messages.

With the following derivation we can conclude the proof of the lemma:

$$\begin{aligned}\tfrac{1}{4}\Pr[\mathbf{H}_{10}] &= \tfrac{1}{4}\sum_{l=1}^{4}\Pr[\mathbf{H}_{10}|r'=l]\Pr[r'=l] = \tfrac{1}{4}\sum_{l=1}^{4}\Pr[\mathbf{H}_{11}|r=l]\Pr[r'=l]\\ &\leq \tfrac{1}{4}\sum_{l=1}^{4}\Pr[\mathbf{H}_{11}|r=l] = \Pr[\mathbf{H}_{11}].\end{aligned}$$

Hybrid $\mathbf{H}_{12}(\lambda)$. Let $\mathbf{H}_{12}$ be the same as $\mathbf{H}_{11}$ but where the values π_σ are computed differently in the j^*-th platform. Let $\mathcal{S}_{\mathsf{svt}}$ be the zero-knowledge simulator of $\mathcal{NIZK}_{\mathsf{svt}}$ and let $\mathcal{NIZK}_{\mathsf{svt}}.\overline{\mathsf{Init}}$ be the zero-knowledge common-reference string generator. At initialization time, the hybrid $\mathbf{H}_{12}$ computes $\mathsf{crs}_{\mathsf{svt}}, \mathsf{tp}_{\mathsf{svt}} \leftarrow \mathcal{NIZK}_{\mathsf{svt}}.\overline{\mathsf{Init}}(\mathsf{bgp})$, and, whenever the adversary queries $(\mathtt{sign}, j^*, M, \mathsf{SigRL})$ when $(j^*, *, *, \bot, *) \in L_{usr}$ (namely the sanitizer is corrupt but the platform is honest), the signature is computed as before but the proof π_σ is computed as $\pi_\sigma \leftarrow \mathcal{S}_{\mathsf{svt}}(\mathsf{tp}_{\mathsf{svt}}, [\mathsf{svt}^{(j^*)}]_1, [\mathbf{c}]_1)$.

Lemma 14. *For any $\mathcal{A} \in \mathbb{A}_1$ $|\Pr[\mathbf{H}_{12}(\lambda)=1] - \Pr[\mathbf{H}_{11}(\lambda)=1]| \in \mathsf{negl}(\lambda)$.*

Proof. First notice that during the join protocol the honest machine asserted that $e(\mathsf{svt}'_0, [c_{\mathcal{S}}]_2) = e(\mathsf{svt}_0^{(j^*)}, [1]_2)$ and $e(\mathsf{svt}'_1 + [c_{\mathcal{M}} \cdot y_{0,\mathcal{S}}], [c_{\mathcal{S}}]_2) = e(\mathsf{svt}_1^{(j^*)}, [1]_2)$, where $\mathsf{svt}' = (\mathsf{svt}'_0, \mathsf{svt}'_1)$ is the value sent by the machine in the first round. Thus the tuple $([\mathsf{svt}^{(j^*)}]_1, [\mathbf{c}]_1)$ belongs in the language proved by $\mathcal{NIZK}_{\mathsf{svt}}$. Noticed that, the lemma follows easily by the composable zero-knowledge property of $\mathcal{NIZK}_{\mathsf{svt}}$.

We can now show that the winning probability in $\mathbf{H}_{12}$ is negligible.

Lemma 15. $\Pr[\mathbf{H}_{12}(\lambda)=1] \in \mathsf{negl}(\lambda)$.

Proof. We reduce to the SXDH Assumption. There are two possible cases for the adversary: either the adversary sends to the challenger the message $(\mathtt{honest\_join}, j^*, \mathcal{M}_i)$ or it sends $(\mathtt{dishonestH\_join}, j^*, \mathcal{I}, \xi)$. In the following we show a reduction for the first case. A similar reduction can be given for the second case, thus we omit here the details.

Consider the following reduction:

Adversary $\mathcal{B}(\mathsf{bgp}, [1, x, y, z]_1)$:

1. Run the adversary $\mathcal{A}$ and simulate the hybrid $\mathbf{H}_{11}$. Specifically, compute the public parameter as the hybrid does but using the bgp given in input to $\mathcal{B}$.
2. **(Install the Challenge - part 1.)** Run the setup algorithm Setup but set $[\mathbf{h}]_1 = [\alpha x, x]_1$ where $\alpha \xleftarrow{\$} \mathbb{Z}_p$.
3. **(Install the Challenge - part 2.)** Eventually, the adversary sends the query $(\mathtt{honest\_join}, j^*, \mathcal{M}_i))$. By the change introduced in $\mathbf{H}_{10}$, the adversary $\mathcal{B}$ simulates an execution of the join protocol using the machine $\tilde{\mathcal{M}}$. In particular, it computes $[t^*]_1 := [h_1 \cdot y_0]_1 + [z]_1$. Recall that by the change introduced in the hybrid $\mathbf{H}_9$ the proof $\pi_{\mathcal{S}}$ is computed using the simulator, thus without the need of the witness (y_0, y).
4. At every signature query $(\mathtt{sign}, j^*, \mathsf{bsn}, M, \mathsf{SigRL})$ if $(\mathsf{bsn}, M) \neq (\mathsf{bsn}^*, M^*)$ then both the signature $\sigma = ([\mathbf{c}]_1, \pi)$ and the proof π_σ can be computed using the respective simulator. Else if $(\mathsf{bsn}, M) = (\mathsf{bsn}^*, M^*)$ then stop the simulation and return a random bit.
5. Eventually the adversary outputs is forgery $(\mathsf{bsn}^*, M^*, \sigma^*, \mathsf{PrivRL}^*, \mathsf{SigRL}^*)$, let $([t^*]_1, [\sigma^*_{sp}]_1, [\mathbf{y}^*]_2)$ be the extracted witness and $[\mathbf{y}^*]_2 = [y_0^*, y_1^*]_2$.
   If W_{11} (namely, the winning condition of $\mathbf{H}_{11}$) does not hold outputs a random bit. Else output 1 if and only if $e([y]_1, [1]_2) = e([1]_1, [\alpha y_0^* + y_1^* - \alpha y_0]_2)$.

First we notice that if the reduction outputs a random bit (because of step 4 or step 5) then the winning condition does not hold. In particular, in step 4 the reduction outputs a random bit if $(\mathsf{bsn}, M) = (\mathsf{bsn}^*, M^*)$, so a tuple $(*, \mathsf{bsn}^*, M^*, *)$ would appear in L_{msg} and therefore, if $[t^*] \in \mathcal{Q}_{sp}$ then condition (7) would not be met.

Secondly, we notice that the distribution of $[t_{\mathcal{M}}]_1, [t_{\mathcal{M}}]_2, \pi_{\mathcal{M}}$ is equivalent to the one in $\mathbf{H}_{12}$, thus $\mathcal{B}$ perfectly simulates $\mathbf{H}_{12}$. Also notice that if $\mathbf{H}_{11} = 1$ then the extracted value $[\mathbf{y}*]_2$ is such that $[\mathbf{h}^{\mathsf{T}} \cdot \mathbf{y}^*]_{\mathrm{t}} = [t^*]_{\mathrm{t}}$. Suppose $z = xy$, then rewriting the equation we have:

$$\alpha x y_0^* + x y_1^* = x \alpha y_0 + xy$$

By simplification, the equation above implies that $y = \alpha y_0^* + y_1^* - \alpha y_0$, thus the reduction $\mathcal{B}$ will always output 1. On the other hand, if z is uniformly random, the reduction $\mathcal{B}$ will output 0 (with overwhelming probability).

By the triangular inequalities and by putting together all the lemmas above, we have now showed that adversaries from the class $\mathbb{A}_1$ can win the unforgeability game only with negligible probability. Thus, we need to show the same statement for the adversary from the class $\mathbb{A}_2$. We roll back to hybrid $\mathbf{H}_7$. We can now show that the winning probability in $\mathbf{H}_7$ is negligible.

Lemma 16. *For any adversary $\mathcal{A} \in \mathbb{A}_2$ we have* $\Pr[\mathbf{H}_7(\lambda) = 1] \in \mathsf{negl}(\lambda)$.

Proof. We reduce to the unforgeability of structure preserving signature $\mathcal{SS}$. Consider the following adversary $\mathcal{B}$ against the existential unforgeability against chosen-message attacks of $\mathcal{SS}$:

Adversary $\mathcal{B}(\mathsf{pk}_{sp})$ with oracle access to $\mathcal{O}_{\mathsf{sign}}(\mathsf{sk}_{sp}, \cdot)$

1. Simulate the hybrid $\mathbf{H}_6$, in particular use pk_{sp} to define the public material in the Setup.
2. Simulate the join protocol using the oracle access to $\mathcal{O}_{\mathsf{sign}}$, in particular whenever the hybrid executes the party $\mathcal{I}$ in a join protocol and receives the message $([t]_1, \pi_{\mathcal{S}})$ query a signature for the message $[t]_1$, then proceed as the hybrid does.
3. When the adversary outputs its forgery, compute $([t^*]_1, [\sigma_{sp}^*]_1, [\mathbf{c}^*]_1)$ as the hybrid does, and output $[t^*]_1, [\sigma_{sp}^*]_1$.

By the definition of $\mathbf{H}_7 = 1$ we have that $[t^*]_1 \notin \{[t]_1 : (i, *, *, *, ([t]_1, *, *)) \in L_{corr}\}$. Also by the definition of $\mathcal{A}$ being in $\mathbb{A}_2$ we have that $[t^*]_1 \notin \{[t]_1 : (i, *, *, *, ([t]_1, *, *)) \in L_{usr}\}$ (with overwhelming probability). Therefore, the adversary $\mathcal{B}$ has not queried $[t^*]_1$ to its signature oracle. Moreover, by the definition of $\mathbf{H}_7 = 1$ the signature verifies, thus this is a valid forgery for $\mathcal{SS}$.

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.*

We first give a sketch of the proof. 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 $\mathbb{G}_1$.

Proof. Recall the security experiment in Fig 1. The experiment postulates adaptive corruption of the platforms, namely, the query $(\texttt{corrupt}, *)$ can be function of the view of the adversary. It is not hard to see that for any PPT adversary that adaptive corrupts the platforms there exists another adversary that commits to its corruptions at the very beginning of the experiment, and, in particular, independently of all the public parameters. More in details, given an adversary $\mathcal{A}$ which performs at most q different join protocols, let $\mathcal{A}'$ be the adversary that (1) first samples two indexes $i_1^*, i_2^* \xleftarrow{\$} [q]$, then corrupts all the platforms expect that i_1^*-th and the i_2^*-th, and (2) runs the same as $\mathcal{A}$ but aborts and returns a random bit if the indexes chosen by $\mathcal{A}$ in the challenge are not i_1^*, i_2^*.

Clearly, for any $b \in \{0,1\}$ we have:

$$\Pr\left[\mathbf{Exp}^{\mathsf{anon}}_{\mathcal{A}',\Pi}(\lambda, b) = b\right] = \tfrac{1}{q^2} \Pr\left[\mathbf{Exp}^{\mathsf{anon}}_{\mathcal{A},\Pi}(\lambda, b) = b\right] + (1 - \tfrac{1}{q^2})/2$$

In the following, we therefore consider adversaries that non-adaptively corrupts the platforms. We give a sequence of hybrid experiments, where $\mathbf{H}_0(\lambda, b) := \mathbf{Exp}^{\mathsf{anon}}_{\mathcal{A}',\Pi}(\lambda, b)$. Moreover, we can assume that the machines $\mathcal{M}_{i_1}$ and $\mathcal{M}_{i_2}$ do not abort during the join protocol. In fact, if this happened then the challenge signature would be $\perp$ so the adversary can guess the challenge almost with probability $\frac{1}{2}$.

Hybrid $\mathbf{H}_1(\lambda, b)$. Let $\mathbf{H}_1$ be the same as $\mathbf{H}_0$ but where the random oracle J is programmed to output random common-reference strings in zero-knowledge mode. In particular, the hybrid $\mathbf{H}_1$ keeps track of all the random oracle query to J recording them in a database $D_{\mathtt{R0}}$, exactly in the same way as the extractor $\mathcal{E}$ of the proof of Thm. 1.

Lemma 17. *For $b \in \{0,1\}$, $|\Pr[\mathbf{H}_1(\lambda, b) = b] - \Pr[\mathbf{H}_0(\lambda, b) = b]| \in \mathsf{negl}(\lambda)$.*

The proof of the lemma follows similarly to the proof of Lemma 1 and therefore it is omitted.

Hybrid $\mathbf{H}_2(\lambda, b)$. Let $\mathbf{H}_2$ be the same as $\mathbf{H}_1$ but where the transcripts output in the i_1-th and i_2-th join protocol are different. Let $\mathcal{S}_{\mathsf{com}}$ be the zero-knowledge simulator of $\mathcal{NIZK}_{\mathsf{com}}$ and let $\mathsf{tp}^{(i_1)}, \mathsf{tp}^{(i_2)}$ be the trapdoor information relative the values $id^{(i_1)}$ and $id^{(i_2)}$ as recorded in the database $D_{\mathtt{R0}}$. (Notice, because all the CRS are simulated such trapdoors always exist.) Upon query $(\texttt{join}, i, (id^{(i)}))$ and $i \in \{i_1, i_2\}$ (namely, the first message in the join protocol sent by the adversary acting as the issuer) compute $([t^{(i)}_{\mathcal{M}}]_1, \pi^{(i)}_{\mathcal{M}}), \mathsf{state}'_i \leftarrow \mathcal{M}_i(\mathsf{state}_i, id^{(i)})$, performs the same verification that $\mathcal{S}$ does and if the checks hold compute $[t^{(i)}]_1$ as $\mathcal{S}$ does and $\tilde{\pi}^{(i)} \leftarrow \mathcal{S}_{\mathsf{com}}(\mathsf{tp}^{(i)}, [t^{(i)}]_1)$. Return $([t^{(i)}]_1, \tilde{\pi}^{(i)})$.

Lemma 18. *For $b \in \{0,1\}$, $|\Pr[\mathbf{H}_2(\lambda, b) = b] - \Pr[\mathbf{H}_1(\lambda, b) = b]| \in \mathsf{negl}(\lambda)$.*

The proof of the lemma follows similarly to the proof of Lemma 11 and therefore it is omitted.

Hybrid $\mathbf{H}_3(\lambda, b)$. Let $\mathbf{H}_3$ be the same as $\mathbf{H}_2$ but with a new condition on signature queries. Specifically, upon oracle query $(\mathtt{sign}, i, M, \mathsf{bsn}, \mathsf{SigRL})$ and $i \in \{i_1, i_2\}$ if $\exists (M_j, \mathsf{bsn}_j, \sigma_j) \in \mathsf{SigRL}$ such that $\sigma_j = ([\mathbf{c}_j]_1, \pi_j)$ and $(-y_0^{(i)}, 1) \cdot [\mathbf{c}_j]_1 = [0]_1$ then output directly $\perp$.

Lemma 19. *For $b \in \{0,1\}$, $\Pr[\mathbf{H}_3(\lambda, b) = b] = \Pr[\mathbf{H}_2(\lambda, b) = b]$.*

Proof. Recall the relation $\mathcal{R}_{\mathsf{sign}}$ in Eq. (4) states that for any tuple $(M_j, \mathsf{bsn}_j, ([\mathbf{c}_j], \pi_j))$ in SigRL we have that $(-y_0^{(i)}, 1) \cdot [\mathbf{c}_j]_1 = [0]_1$ (namely, $\mathbf{c}_j$ is not in the span of $(1, y_0^{(i)})$). Therefore, by correctness of the NIZK scheme, if the event checked by the hybrid happens then the honest prover would not be able to produce a valid proof because the instance is not in the language.

Hybrid $\mathbf{H}_4(\lambda, b)$. Let $\mathbf{H}_4$ be the same as $\mathbf{H}_3$ but where we program differently the random oracle H. In particular, upon a query $(\mathtt{RO}, \mathsf{H}, x)$ the hybrid samples $\mathsf{crs}, \mathsf{tp}_\mathsf{s} \xleftarrow{\$} \mathcal{NIZK}_{\mathsf{sign}}.\overline{\mathsf{Init}}_{zk}(\mathsf{bgp})$ and sets the tuple $(\mathsf{H}, x, \mathsf{crs}, \mathsf{tp}_\mathsf{s})$ into the database $D_{\mathtt{RO}}$.

Lemma 20. $|\Pr[\mathbf{H}_4(\lambda, b) = 1] - \Pr[\mathbf{H}_3(\lambda, b) = 1]| \in \mathsf{negl}(\lambda)$.

Proof. The proof of the lemma follows by property (i) of Def. 8 (adaptive composable perfect zero-knowledge) of $\mathcal{NIZK}_{\mathsf{sign}}$.

Hybrid $\mathbf{H}_5(\lambda, b)$. Let $\mathbf{H}_5$ be the same as $\mathbf{H}_4$ but where the signatures for the platforms i_1 and i_2 are computed differently. In particular, the hybrid $\mathbf{H}_5$ upon query $(\mathtt{sign}, i, M, \mathsf{bsn}, \mathsf{SigRL})$ where $i \in \{i_1, i_2\}$ (resp. the challenge $(M^*, \mathsf{bsn}^*, i_1, i_2, \mathsf{SigRL})$) computes $\pi \xleftarrow{\$} \mathcal{S}_{\mathsf{sign}}(\mathsf{tp}_\mathsf{s}, [\mathbf{c}])$ (resp. computes $\pi^* \xleftarrow{\$} \mathcal{S}_{\mathsf{sign}}(\mathsf{tp}_\mathsf{s}, [\mathbf{c}^*])$).

Lemma 21. *For $b \in \{0,1\}$,* $\Pr[\mathbf{H}_5(\lambda, b) = b] = \Pr[\mathbf{H}_4(\lambda, b) = b]$.

Proof. The additional check introduced in the hybrid experiment $\mathbf{H}_3$ guarantees that if the instance $[\mathbf{c}]_1, \mathsf{SigRL}$ is not in the language of the NIZK $\mathcal{NIZK}_{\mathsf{sign}}$ then the challenger answers the query with $\perp$. Thus the simulated proofs are generated only for instances in the language. Moreover, by the Assumption 1 the (possibly subverted) machines $\mathcal{M}_{i_1}$ and $\mathcal{M}_{i_2}$ always produce a signature (when the correctness property of the EPID is verified). The lemma follows by the perfect composable zero-knowledge property of $\mathcal{NIZK}_{\mathsf{sign}}$.

Hybrid $\mathbf{H}_6(\lambda, b)$. Let $\mathbf{H}_6$ be the same as $\mathbf{H}_5$ but where $t^{(i)} \xleftarrow{\$} \mathbb{Z}_p$ for $i \in \{i_1, i_2\}$. (The hybrid $\mathbf{H}_5$ samples the value y_0 when simulating the honest sanitizer $\mathcal{S}_{i^*}$.)

Lemma 22. *For $b \in \{0,1\}$,* $\Pr[\mathbf{H}_6(\lambda, b) = b] = \Pr[\mathbf{H}_5(\lambda, b) = b]$.

Proof. For $i \in \{i_1, i_2\}$ the distribution $t^{(i)} \xleftarrow{\$} \mathbb{Z}_p$ and $t^{(i)}_{\mathcal{M}} + h_1 y_{\mathcal{S}}$ where $y_{\mathcal{S}} \xleftarrow{\$} \mathbb{Z}_p$ are equivalent. Moreover, conditioning on a specific value for $t^{(i)}$ the view of the adversary is independent of $y_{\mathcal{S}}$ because the NIZK proofs from $\mathcal{S}$ to $\mathcal{I}$ are simulated.

Hybrid $\mathbf{H}_7(\lambda)$. Let $\mathbf{H}_7$ be the same as $\mathbf{H}_6$ but where, for any signature produced by the platforms i_1 and i_2 we additionally control that $[\mathbf{c}]_1$ is of the right form. Specifically, upon query $(\mathtt{sign}, i, M, \mathsf{SigRL})$ where and $i \in \{i_1, i_2\}$, the hybrid computes $\sigma = ([c_0, c_1]_1, \pi), \mathsf{state}'_i \leftarrow \mathcal{M}_i(\mathsf{state}_i, M)$, and return $\perp$ to the adversary if $e([c_1]_1, [1]_2) \neq e([c_0]_1, [y_0]_2)$.

Lemma 23. *For $b \in \{0,1\}$,* $\Pr[\mathbf{H}_7(\lambda, b) = b] = \Pr[\mathbf{H}_6(\lambda, b) = b]$.

The proof of the lemma follows identically to the proof of Lemma 8, therefore the proof is omitted.

Hybrid $\mathbf{H}_8(\lambda, b)$. Let $\mathbf{H}_8$ be the same as $\mathbf{H}_7$ but where the challenge signature is computed differently. Specifically, a fresh $z \xleftarrow{\$} \mathbb{Z}_p$ is sampled and $[\mathbf{c}^*]_1 \leftarrow [c_0^*, z \cdot c_0]_1$ where $[c_0^*] = \mathsf{K}(\mathsf{bsn}^*)$.

Lemma 24. *For $b \in \{0,1\}$,* $|\Pr[\mathbf{H}_8(\lambda, b) = b] - \Pr[\mathbf{H}_7(\lambda, b) = b]| \in \mathsf{negl}(\lambda)$.

Proof. We give a reduction to the SXDH problem in $\mathbb{G}_1$. We assume that $\mathcal{A}$ does not query more than once the random oracles on the same point, also we assume that $\mathcal{A}$ queries the challenge basename bsn^* before outputting its challenge. Both the assumptions are without loss of generality. Consider the following adversary $\mathcal{B}$:

Adversary $\mathcal{B}([1, x, y, z]_1)$:

1. Let q be an upper bound of the number of queries to K made by $\mathcal{A}$, let $j^* \xleftarrow{\$} [q]$.
2. Simulate the experiment $\mathbf{H}_8(\lambda, b)$ and the random oracles $\mathcal{H}, \mathsf{K}, \mathcal{J}$, in particular, simulate the random oracle K by maintaining a list $D_{\mathtt{RO}}$ initially empty. At the j-th query to K, if $j = j^*$ then add the tuple $(\mathsf{bsn}', 1, [x]_1)$ and reply with $[x]_1$ else add the entry $(\mathsf{bsn}', 0, \alpha)$ where $\alpha \xleftarrow{\$} \mathbb{Z}_p$ and return $[\alpha]_1$.

3. Upon query $(\texttt{sign}, i_b, M, \mathsf{bsn}, \mathsf{SigRL})$, if the tuple $(\mathsf{bsn}, 1, *) \in D_{\texttt{RO}}$ stop the simulation and return a random bit, else retrieve (or create) the tuple $(\mathsf{bsn}, 0, \alpha)$ from $D_{\texttt{RO}}$. If it exists $(M_j, \mathsf{bsn}_j, \sigma_j = ([c_{j,0}, c_{j,1}]_1, \pi_j))$ in SigRL such that $(\mathsf{bsn}, 0, \alpha') \in D_{\texttt{RO}}$ and $[c_{j,1}] = \alpha'[y]_1$ output directly $\perp$ (simulating the check introduced in $\mathbf{H}_4$). Compute the signature by setting $[\mathbf{c}] \leftarrow \alpha \cdot [1, y]_1$ and compute π using the simulator of $\mathcal{NIZK}_{\mathsf{sign}}$ (according to the change introduced in $\mathbf{H}_8$).
4. Let $(M^*, \mathsf{bsn}^*, i_0, i_1, \mathsf{SigRL})$ be the output of $\mathcal{A}$ as in step 3 of Fig. 1. Retrieve the tuple (bsn^*, α) from the list $D_{\texttt{RO}}$ (or create it if it does not exist). Compute the signature by setting $[\mathbf{c}^*] \leftarrow \cdot[x, z]_1$ and computing π using the simulator of $\mathcal{NIZK}_{\mathsf{sign}}$ (according to the change introduced in $\mathbf{H}_8$).
5. Continue simulating the experiment $\mathbf{H}_8$ and output what $\mathcal{A}$ outputs.

First we notice that given the transcript of the join protocol for the platform i_b the value $y_0^{(i_b)}$ is uniformly distributed. In particular, the distribution of $[y]_1$ and $[y_0^{(i_b)}]_1$ are the same, so the signatures for i_b produced by $\mathcal{B}$ are distributed equivalently to the signatures in $\mathbf{H}_7$ (and $\mathbf{H}_8$). Secondly notice that, if $z = xy$ then the distribution of the challenge signature is exactly as in $\mathbf{H}_7$, while if z is uniformly random then the distribution is exactly as in $\mathbf{H}_8$. This concludes the proof.

Lemma 25. *For any $b \in \{0, 1\}$, $\Pr[\mathbf{H}_8(\lambda, b) = 0] = 1/2$*

Proof. Let us analyze the view of the adversary in the hybrid $\mathbf{H}_8$. Notice that the adversary cannot get any extra information about the bit b using the signature oracle and different SigRL, because the definition of anonymity implies that if one of the two platform (i_1 or i_2) would output $\perp$ then the output of the oracle is $\perp$. Also, notice that for both the challenge signature is made with a value y_0 that is uncorrelated to both the transcript of the join protocol and the signatures released by the two platform. Therefore the full view of the adversary is independent of the bit b, thus the winning probability.

Non frameability Similarly to anonymity, adaptive corruption and selective corruption for non-frameability are equivalent up to a polynomial degradation of the advantage of the adversary. So we can assume that the challenger knows the honest platform that will be attacked by the adversary, let such platform be the i^*-th platform. Again, similarly to the proof of anonymity and the proof of unforgeability of our scheme, we switch, thanks to the strong derivation privacy of the NIZK schemes, to an hybrid experiment where all the messages, both during the join protocol and the signature queries, are simulated by challenger and where, moreover, the signature forged by the adversary is extractable. Also, similarly to the proof of unforgeability, thanks to Assumption 1, we substitute the machine of the i^*-th platform with a *well-behaving* machine.

At this point we can reduce the security to the computational problem of finding $[x]_2$ given $[x]_1$, which directly implies the XDH assumption in $\mathbb{G}_1$. The idea of the reduction is that, given the challenge $[x]_1$ we can (implicitly) install the element x as the first element of the platform key of the i^*-th platform. Notice that given $[x]_1$, by programming the random oracle and thanks to the simulation trapdoors, we can faithfully run this hybrid-version of the non-frameability game. Moreover, we do not need to explicitly communicate the platform key to the machine of the i^*-th platform because we substituted it with a well-behaving one. Once the adversary output its forgery, we can use the extraction trapdoor to extract the witness from the signature. A successful adversary forges a signature $([\mathbf{c}^*], \pi^*)$ that links to another signature $([\mathbf{c}]_1, \pi)$ produced by the i^*-th platform, recall that the linking procedure, given the two signatures on the same basename bsn^*, checks that $[\mathbf{c}^*]_1 = [\mathbf{c}]_1$ and verifies the signatures, thus we have $[\mathbf{c}_1^*] = \mathsf{K}(\mathsf{bsn}) \cdot x$ and the reduction must have extracted the value $[x]_2$ from proof π^* of the forged signature.

Theorem 3. *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 our SR-EPID is non-frameable in the ROM.*

Proof. Recall the security experiment in Fig 3. The experiment postulates adaptive corruption of the platforms, namely, the query $(\texttt{corrupt}, *)$ can be function of the view of the adversary. As for anonymity, it is not hard to see that for any PPT adversary that adaptive corrupts the platforms there exists another adversary

that commits to its corruptions at the very beginning of the experiment, and, in particular, independently of all the public parameters. More in details, given an adversary $\mathcal{A}$ which performs at most q different join protocols, let $\mathcal{A}'$ be the adversary that (1) first samples an index $i' \xleftarrow{\$} [q]$, then corrupts all the platforms expect that i'-th, and (2) runs the same as $\mathcal{A}$ but aborts if the index i^* chosen by $\mathcal{A}$ in the forgery is not equal to i'.

Clearly, for any $b \in \{0,1\}$ we have:

$$\Pr\left[\mathbf{Exp}^{\mathsf{non\text{-}frame}}_{\mathcal{A}',\Pi}(\lambda) = 1\right] = \tfrac{1}{q}\Pr\left[\mathbf{Exp}^{\mathsf{non\text{-}frame}}_{\mathcal{A},\Pi}(\lambda) = 1\right]$$

In the following, we therefore consider adversaries that non-adaptively corrupts the platforms. We give a sequence of hybrid experiments, where $\mathbf{H}_0(\lambda) := \mathbf{Exp}^{\mathsf{non\text{-}frame}}_{\mathcal{A}',\Pi}(\lambda)$. Moreover, we can assume that the machines $\mathcal{M}_{i^*}$ (if the i^*-th platform has a subverted signer) does not abort during the join protocol. Also, we can assume that the i^*-th join protocol between an honest machine $\mathcal{M}$ and a malicious sanitizer does not abort. In fact, if these two events happened then the winning condition (3) would not be satisfied.

The first step of the hybrid argument proceed exactly the same as in the hybrid step $\mathbf{H}_2$ and $\mathbf{H}_3$ of the proof of Theorem 1. In the next hybrid we summarize the change. As in the proof of Theorem 1, we call W_i the winning condition in the hybrid experiment $\mathbf{H}_i$, we set $W_1 := (1) \wedge (2) \wedge (3) \wedge (4)$ and, whenever we don't mention it explicitly, we set $W_{i+1} := W_i$.

Hybrid $\mathbf{H}_1(\lambda)$. Let $\mathbf{H}_1$ be the same as $\mathbf{H}_0$ but where the winning condition is changed and random oracle H is programmed. In particular, the hybrid samples an index $j^* \xleftarrow{\$} [q_{\mathsf{H}}]$ where q_{H} is an upper bound on the number of oracle queries made by $\mathcal{A}$ to H, and upon the j-th query $(\mathtt{RO}, \mathsf{H}, x)$ to the random oracle (w.l.g. we assume the adversary does not query twice the RO with the same input) if $j = j^*$ then the hybrid samples $\mathsf{crs}, \mathsf{tp}_{\mathsf{e}} \xleftarrow{\$} \overline{\mathsf{Init}}_{snd}(\mathsf{bgp})$ and sets the tuple $(\mathsf{H}, x, \mathsf{crs}, \mathsf{tp}_{\mathsf{e}})$ in $D_{\mathtt{RO}}$, else it samples $\mathsf{crs}, \mathsf{tp}_{\mathsf{s}} \xleftarrow{\$} \overline{\mathsf{Init}}_{zk}(\mathsf{bgp})$ and sets the tuple $(\mathsf{H}, x, \mathsf{crs}, \mathsf{tp}_{\mathsf{s}})$ in $D_{\mathtt{RO}}$.

Moreover, the new winning condition is set to be $W_2 := W_1 \wedge (5)$, where (5) is defined as:

(bsn^*, M^*) (the basename-message tuple of the forgery) is queried to the random oracle H at the j^*-th query.

Lemma 26. $\Pr[\mathbf{H}_1(\lambda) = 1] \geq \Pr[\mathbf{H}_0(\lambda) = 1]/q_{\mathsf{H}} - \mathsf{negl}(\lambda)$.

The proof of the lemma follows similarly to the proof of Lemmas 4 and 6, therefore it is omitted.

The second step of the hybrid argument proceeds similar to the proof of Theorem 2. In particular, we apply the same modifications as in the hybrids $\mathbf{H}_1$, $\mathbf{H}_2$ and $\mathbf{H}_3$ of Theorem 2. We summarize in the next hybrid the changes.

Hybrid $\mathbf{H}_2(\lambda)$. Let $\mathbf{H}_2$ be the same as $\mathbf{H}_1$ but where:

1. The common reference string $\mathsf{crs}_{\mathsf{svt}}$ is sampled in zero-knowledge mode. Thus the hybrids knows $\mathsf{tp}_{\mathsf{svt}}$ for simulations.
2. The random oracle J is programmed to output random common-reference strings in zero-knowledge mode.
3. Let $\mathcal{S}_{\mathsf{com}}$ be the zero-knowledge simulator of $\mathcal{NIZK}_{\mathsf{com}}$ and let tp_{i^*} be the trapdoor information relative the value id_{i^*} as recorded in the database $D_{\mathtt{RO}}$.
   Upon query $(\mathtt{join}, i^*, (id))$ (namely, the first message in the join protocol sent by the adversary acting as the issuer) or upon query $(\mathtt{dishonest\_joinS}, i^*, (y_{0,\mathcal{S}}, \mathsf{svt}))$ pick $t^{(i^*)} \xleftarrow{\$} \mathbb{Z}_p$ and $\tilde{\pi}^{(i^*)} \leftarrow \mathcal{S}_{\mathsf{com}}(\mathsf{tp}_{i^*}, [t^{(i^*)}]_1)$, and send $([t^{(i^*)}]_1, \tilde{\pi}^{(i^*)})$ to the adversary.
4. Upon oracle query $(\mathtt{sign}, i^*, M, \mathsf{bsn}, \mathsf{SigRL})$ if $\exists (M_j, \mathsf{bsn}_j, \sigma_j) \in \mathsf{SigRL}$ such that $\sigma_j = ([\mathbf{c}]_1, \pi)$ and $(-y_0^{(i^*)}, 1) \cdot [\mathbf{c}]_1 = [0]_1$ then output directly $\perp$. (Notice that if the i^*-th machine is honest then this condition is always true.)

Lemma 27. $|\Pr[\mathbf{H}_2(\lambda) = 1] - \Pr[\mathbf{H}_1(\lambda) = 1]| \in \mathsf{negl}(\lambda)$.

The proof of the lemma follows very similar to the conjunction of Lemmas 8,17,18,20 and 19 therefore it is omitted.

Hybrid $\mathbf{H}_3(\lambda)$. Let $\mathbf{H}_3$ be the same as $\mathbf{H}_2$ but where the signatures are computed differently. Specifically, upon oracle query $(\mathtt{sign}, i^*, \mathsf{bsn}, M, \mathsf{SigRL})$ the hybrid experiment $\mathbf{H}_3$ computes $[\mathbf{c}]_1 := (\mathsf{K}(\mathsf{bsn}), \mathsf{K}(\mathsf{bsn})y_0^{(i^*)})$, retrieves the tuple $(\mathsf{H}, (\mathsf{bsn}, M), \mathsf{crs}, \mathsf{tp_s})$ from $D_{\mathtt{RO}}$ and computes $\sigma \xleftarrow{\$} \mathcal{S}_{\mathsf{sign}}(\mathsf{tp_s}, [\mathbf{c}])$ and $\pi_\sigma \xleftarrow{\$} \mathcal{S}_{\mathsf{svt}}(\mathsf{tp_{svt}}, (\mathsf{svt}, [\mathbf{c}]_1))$ and returns (σ, π_σ).

Lemma 28. $\Pr[\mathbf{H}_3(\lambda) = 1] = \Pr[\mathbf{H}_2(\lambda) = 1]$.

Proof. The additional check introduced in the hybrid experiment $\mathbf{H}_2$ guarantees that if the instance $[\mathbf{c}]_1, \mathsf{SigRL}$ is not in the language of the NIZK $\mathcal{NIZK}_{\mathsf{sign}}$ then the challenger answers the query with $\perp$. Thus the simulated proofs are generated only for instances in the language. Also by the winning conditions, the adversary would never query a signature for bsn^*, M^*, so the retrieved trapdoors $\mathsf{tp_s}$ allow for zero-knowledge. The lemma follows by the adaptive composable perfect zero-knowledge property (Def. 8) of $\mathcal{NIZK}_{\mathsf{sign}}$.

Lemma 29. $\Pr[\mathbf{H}_3(\lambda) = 1] \in \mathsf{negl}(\lambda)$.

Proof. We reduce to the SXDH Assumption in $\mathbb{G}_1$. In the following we say that the i^*-th sanitizer is malicious if the i^*-th join was performed using the interface `dishonestS_join`, formally if $(i^*, *, *, *, 0, *) \in L_{usr}$, otherwise we say that i^*-th sanitizer is honest. Recall that in the latter case the i^*-th machine might be subverted. Consider the following adversary $\mathcal{B}$:

Adversary $\mathcal{B}([1, x, y, z]_1)$:

1. Simulate the experiment $\mathbf{H}_3(\lambda)$ and the random oracles $\mathsf{H}, \mathsf{K}, \mathsf{J}$, in particular, simulate the random oracle K by maintaining a list $D_{\mathtt{RO}}$ initially empty and whenever $\mathcal{A}$ sends the query bsn' if $(\mathsf{K}, \mathsf{bsn}', *) \notin D_{\mathtt{RO}}$ then add the entry $(\mathsf{K}, \mathsf{bsn}', \alpha)$ where $\alpha \xleftarrow{\$} \mathbb{Z}_p$ and return $[\alpha]_1$.
2. Upon query $(\mathtt{dishonestS\_join}, i^*, (id))$ let $\mathsf{svt}' := c_{\mathcal{M}} \cdot [1, x]_1$ for $c_{\mathcal{M}} \xleftarrow{\$} \mathbb{Z}_p$, return it to the adversary.
3. Upon query $(\mathtt{dishonestS\_join}, i^*, (y_{0,\mathcal{S}}^{(i^*)}, \mathsf{svt}, [c_{\mathcal{S}}]_2))$ store $y_{0,\mathcal{S}}^{(i^*)}$, follow the described protocol for $\mathcal{M}$ and assert that $e(\mathsf{svt}'_1 + c_{\mathcal{M}}[x]_1, [c_{\mathcal{S}}]_2) = e(\mathsf{svt}_1, [1]_2)$.
4. If the i^*-th sanitizer is malicious then upon query $(\mathtt{sign}, i^*, \mathsf{bsn}, M, \mathsf{SigRL})$ Retrieve the tuple $(\mathsf{K}, \mathsf{bsn}, \alpha)$ from the list $D_{\mathtt{RO}}$ (or create it if it does not exist). Compute the (σ, π_σ) by setting $[\mathbf{c}] \leftarrow \alpha \cdot [1, x + y_{0,\mathcal{S}}^{(i^*)}]_1$ and π, π_σ using the simulators of $\mathcal{NIZK}_{\mathsf{sign}}$ and $\mathcal{NIZK}_{\mathsf{svt}}$.
5. Upon query $(\mathtt{sign}, i^*, \mathsf{bsn}, M, \mathsf{SigRL})$ Retrieve the tuple $(\mathsf{K}, \mathsf{bsn}, \alpha)$ from the list $D_{\mathtt{RO}}$ (or create it if it does not exist). Compute (σ, π_σ) by setting $[\mathbf{c}] \leftarrow \alpha \cdot [1, x]_1$ and π, π_σ using the simulators of $\mathcal{NIZK}_{\mathsf{sign}}$ and $\mathcal{NIZK}_{\mathsf{svt}}$. If the i^*-th sanitizer is honest then output σ else output σ, π_σ.
6. Let $(M^*, \mathsf{bsn}^*, \sigma^*)$ be forgery of $\mathcal{A}$. Retrieve the tuple $(\mathsf{K}, \mathsf{bsn}^*, \alpha)$ from the list $D_{\mathtt{RO}}$. If the winning condition W_3 holds, parse $\sigma^* = ([\mathbf{c}]^*, \pi^*)$, extract the proof π, computing $([t]_1, [\sigma_{sp}]_1, [y_0, y_1]_2) \leftarrow \mathcal{E}(\mathsf{tp_e}, \pi)$ and
   - if i^*-th sanitizer honest then output 1 if $e([y]_1, [y_0]_2) = e([z]_1, [1]_2)$,
   - else then output 1 if $e([y]_1, [y_0 - y_{0,\mathcal{S}}^{(i^*)}]_2) = e([z]_1, [1]_2)$.

First we notice that, similarly to the proof of Thm 2, given the transcript of the join protocol for the platform i^* the value $y_0^{(i^*)}$ is uniformly distributed. In particular, if the i^*-th sanitizer is honest then the distribution of $[x]_1$ and $[y_0^{(i^*)}]_1$ are the same while if the i^*-th sanitizer is malicious then the distribution of $[x]_1$ and $[y_{0,\mathcal{M}}^{(i^*)}]_1$ are the same thus the signatures for i^* produced by $\mathcal{B}$ are distributed equivalently to the signatures in $\mathbf{H}_3$. Notice that by the winning condition W_3 we have that (bsn^*, M^*) is queried at the j^*-th oracle query, thus we can use the trapdoor $\mathsf{tp_e}$ and by adaptive knowledge-soundness of $\mathcal{NIZK}_{\mathsf{sign}}$ the proof π^* can be extracted and for the extracted value $[y_0]_2$ it holds that $[\mathbf{c}_2]_1 = [y_0 \cdot \mathbf{c}]_1$. Because the signature σ^* links to a signature produced by the platform i^* it must be the case that

- Either $y_0 = x$ and the i^*-th sanitizer is honest, thus, when $z = xy$ the equation $e([y]_1, [y_0]_2) = e([z]_1, [1]_2)$ will always hold, while when z is uniformly random the equation will be false (with overwhelming probability).

– Or $y_0 = x + y_{0,\mathcal{S}}^{(i^*)}$ and the i^*-th sanitizer is malicious, thus, when $z = xy$ the equation $e([y]_1, [y_0 - y_{0,\mathcal{S}}^{(i^*)}]_2) = e([z]_1, [1]_2)$ will always hold, while when z is uniformly random the equation will be false (with overwhelming probability).

This concludes the proof.

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 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 (Aug 2010). doi: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: ACNS 19. pp. 465–485. LNCS, Springer, Heidelberg (2019). doi:10.1007/978-3-030-21568-2_23
3. Ateniese, G., Magri, B., Venturi, D.: Subversion-resilient signature schemes. In: Ray, I., Li, N., Kruegel:, C. (eds.) ACM CCS 15. pp. 364–375. ACM Press (Oct 2015). doi:10.1145/2810103.2813635
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 (Dec 2016). doi: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 (May 2003). doi: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 (Aug 2014). doi: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) (Jun 2013)
9. Brickell, E., Li, J.: Enhanced privacy id: a direct anonymous attestation scheme with enhanced revocation capabilities. In: ACM WPES
10. Brickell, E., Li, J.: Enhanced privacy ID: A direct anonymous attestation scheme with enhanced revocation capabilities. IEEE Trans. Dependable Sec. Comput. **9**(3)
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, Heidelberg (Aug 2017). doi:10.1007/978-3-319-63697-9_15
13. Camenisch, J., Lehmann, A.: (Un)linkable pseudonyms for governmental databases. In: Ray, I., Li, N., Kruegel:, C. (eds.) ACM CCS 15. pp. 1467–1479. ACM Press (Oct 2015). doi:10.1145/2810103.2813658
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.J.B. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 254–274. Springer, Heidelberg (Aug 2015). doi:10.1007/978-3-662-48000-7_13

15. Chakraborty, S., Dziembowski, S., Nielsen, J.B.: Reverse firewalls for actively secure MPCs. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2020, Part II. pp. 732–762. LNCS, Springer, Heidelberg (Aug 2020). doi: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 (Apr 2012). doi:10.1007/978-3-642-29011-4_18
17. Chatterjee, S., Menezes, A.: Type 2 structure-preserving signature schemes revisited. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015, Part I. LNCS, vol. 9452, pp. 286–310. Springer, Heidelberg (Nov / Dec 2015). doi:10.1007/978-3-662-48797-6_13
18. 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, Part I. LNCS, vol. 10031, pp. 844–876. Springer, Heidelberg (Dec 2016). doi:10.1007/978-3-662-53887-6_31
19. 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: PKC 2019, Part I. pp. 221–251. LNCS, Springer, Heidelberg (2019). doi:10.1007/978-3-030-17253-4_8
20. 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 (Aug 2013). doi:10.1007/978-3-642-40084-1_8
21. Faust, S., Kohlweiss, M., Marson, G.A., Venturi, D.: On the non-malleability of the Fiat-Shamir transform. In: Galbraith, S.D., Nandi, M. (eds.) INDOCRYPT 2012. LNCS, vol. 7668, pp. 60–79. Springer, Heidelberg (Dec 2012). doi:10.1007/978-3-642-34931-7_5
22. Fischlin, M., Mazaheri, S.: Self-guarding cryptographic protocols against algorithm substitution attacks. pp. 76–90 (2018). doi:10.1109/CSF.2018.00013
23. Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for cryptographers. Discrete Applied Mathematics **156**(16) (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., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 415–432. Springer, Heidelberg (Apr 2008). doi:10.1007/978-3-540-78967-3_24
26. Hofheinz, D., Kiltz, E.: Programmable hash functions and their applications. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 21–38. Springer, Heidelberg (Aug 2008). doi:10.1007/978-3-540-85174-5_2
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 (May 2014). doi: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 (Apr 2015). doi: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 (Dec 2016). doi: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: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 17. pp. 907–922. ACM Press (Oct / Nov 2017). doi:10.1145/3133956.3133993
32. Sanders, O., Traoré, J.: EPID with malicious revocation. pp. 177–200. LNCS, Springer, Heidelberg (2021). doi:10.1007/978-3-030-75539-3_8
33. Young, A., Yung, M.: The dark side of "black-box" cryptography, or: Should we trust capstone? In: Koblitz, N. (ed.) CRYPTO'96. LNCS, vol. 1109, pp. 89–103. Springer, Heidelberg (Aug 1996). doi:10.1007/3-540-68697-5_8

A Definitions of Unforgeability for EPID

We formalize unforgeability and strong unforgeability for EPID via the experiments of Fig. 5, we first define the notion of unforgeability:

Definition 13. *Consider the experiment* $\mathbf{Exp}^{\mathtt{unf}}_{\mathcal{A},\mathcal{E},\Pi}$ *described in Fig. 5. We say that an EPID* Π *is unforgeable if there exist PPT algorithms* CheckTK, 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* $\mathsf{sk}_1 \neq \mathsf{sk}_2 \wedge (\mathsf{CheckTK}(\mathsf{gpk}, \mathsf{sk}_1, \mathsf{tk}) = 0 \vee \mathsf{CheckTK}(\mathsf{gpk}, \mathsf{sk}_2, \mathsf{tk}) = 0)$.
(Namely, tk *is uniquely associated to an* sk *without any* collisions*.)*
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* $\mathsf{Vf}(\mathsf{gpk}, \mathsf{bsn}, M, \sigma, \mathsf{SigRL}, \mathsf{PrivRL}) = 1$ *and* $\mathsf{Vf}(\mathsf{gpk}, \mathsf{bsn}, M, \sigma, \mathsf{SigRL}, \mathsf{PrivRL} \cup \{\mathsf{sk}\}) = 0$*, 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* CheckSig *allow to verify if a signature comes from a specific secret key.)*
3. *For any PPT adversary* $\mathcal{A}$, $\mathbf{Adv}^{\mathtt{unf}}_{\mathcal{A},\mathcal{E},\Pi}(\lambda) := \Pr\left[\mathbf{Exp}^{\mathtt{unf}}_{\mathcal{A},\mathcal{E},\Pi}(\lambda) = 1\right] \in \mathsf{negl}(\lambda)$.
4. *The distribution* $\{\mathsf{pub} \xleftarrow{\$} \mathsf{Init}(1^\lambda)\}_{\lambda\in\mathbb{N}}$ *and* $\{\mathsf{pub}|\mathsf{pub}, \mathsf{tp} \xleftarrow{\$} \mathcal{E}_0(1^\lambda)\}_{\lambda\in\mathbb{N}}$ *are computationally indistinguishable.*

The difference between strong unforgeability and unforgeability, similarly for standard digital signature scheme, is that the latter does not allow for re-randomizable signature schemes:

Definition 14. *Consider the experiment* $\mathbf{Exp}^{\mathtt{s\text{-}unf}}_{\mathcal{A},\mathcal{E},\Pi}$ *described in Fig. 5. We say that an EPID* Π *is strongly unforgeable if there exist PPT algorithms* CheckTK, CheckSig, *and a PPT extractor* $\mathcal{E} = (\mathcal{E}_0, \mathcal{E}_1)$ *such that items (1),(2) and (4) hold as in Def. 13 and additionally:*

3. *For any PPT adversary* $\mathcal{A}$, $\mathbf{Adv}^{\mathtt{s\text{-}unf}}_{\mathcal{A},\mathcal{E},\Pi}(\lambda) := \Pr\left[\mathbf{Exp}^{\mathtt{s\text{-}unf}}_{\mathcal{A},\mathcal{E},\Pi}(\lambda) = 1\right] \in \mathsf{negl}(\lambda)$.

In a very recent work, Sanders and Traoré also propose a formalization of the unforgeability notion for EPID [**?**]. Their definition is different from ours. In particular, our goal of giving the definition in this section is to provide a formal version of the unforgeability notion informally described for EPID in [9,10]. The goal of [**?**] is instead to avoid definitions using an extractor and that require the adversary to essentially include secret keys or signatures of corrupted users in the revocation lists.

B The verification token is necessary

We call an SR-EPID scheme *without verification token* an SR-EPID in which the Join algorithm does not produce any svt value, the Sanitize algorithm does not have svt among its inputs, and finally Sig does not return a value π_σ. More precisely, an SR-EPID without verification token has the following differences:

- $\mathsf{Join}_{\mathcal{I},\mathcal{S}_i,\mathcal{M}_i}\langle(\mathsf{gpk}, \mathsf{isk}), \mathsf{gpk}, \mathsf{gpk})\rangle \to \langle b, b, \mathsf{sk}_i\rangle$. 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, $\mathcal{I}$ and $\mathcal{S}_i$ obtain a bit b indicating if the protocol terminated successfully, $\mathcal{M}_i$ obtains private key sk_i.
- $\mathsf{Sig}(\mathsf{gpk}, \mathsf{sk}_i, \mathsf{bsn}, M, \mathsf{SigRL}) \to \perp/\sigma$. 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, or an error $\perp$.
- $\mathsf{Sanitize}(\mathsf{gpk}, \mathsf{bsn}, M, \sigma, \mathsf{SigRL}) \to \perp/\sigma'$. The sanitization algorithm takes as input gpk, a basename bsn, a message M, a signature σ, a signature based revocation list SigRL.

The notion of correctness is the same as Def. 2 but now we do not take into account the syntax for the proof of the signature and the verification token.

Theorem 4. *If* Π *is an SR-EPID without verification token that is either linking-correct or correct with revocation lists, then SR-EPID is not anonymous (according to Def. 3).*

Proof. We show an attack against the anonymity game. We describe a subverted machine with hardcoded a secret key $\tilde{\mathsf{sk}}$

Subverted-Machine $\mathcal{M}_{\tilde{\mathsf{sk}}}$:

Experiment $\mathbf{Exp}^{\mathsf{unf}}_{\mathcal{A},\mathcal{E},\Pi}(\lambda)$, $\underline{\mathbf{Exp}^{\mathsf{s\text{-}unf}}_{\mathcal{A},\mathcal{E},\Pi}(\lambda)}$

$L_{join}, L_{usr}, L_{corr}, L_{msg} \leftarrow \emptyset$;
$(\mathsf{pub}, \mathsf{tp}) \leftarrow \mathcal{E}_0(1^\lambda)$; $(\mathsf{gpk}, \mathsf{gsk}) \xleftarrow{\$} \mathsf{Setup}(\mathsf{pub})$;
$(\mathsf{bsn}^*, M^*, \sigma^*, \mathsf{PrivRL}^*, \mathsf{SigRL}^*) \leftarrow \mathcal{A}(\mathsf{gpk})^{\mathcal{C}(\mathsf{sk},\cdot)}$;
$R \leftarrow \{\mathsf{tk}_i : (i, *, *, *, \mathsf{tk}_i) \in L_{corr}\}$;
return 1 if and only if $((1) \wedge (2) \wedge (3)) \vee (4)$:
  (1) $\forall \mathsf{tk} \in R : (\exists \mathsf{sk} \in \mathsf{PrivRL}^* : \mathsf{CheckTK}(\mathsf{gpk}, \mathsf{sk}, \mathsf{tk}) = 1)$
    OR $(\exists \sigma \in \mathsf{SigRL}^* : \mathsf{CheckSig}(\mathsf{gpk}, \mathsf{tk}, \sigma) = 1)$
  (2) $\mathsf{Ver}(\mathsf{gpk}, \mathsf{bsn}^*, M^*, \sigma^*, \mathsf{SigRL}^*, \mathsf{PrivRL}^*) = 1$
  (3) $\forall (*, \mathsf{bsn}^*, M^*, \sigma) \in L_{msg} : \mathsf{Link}(\mathsf{gpk}, \mathsf{bsn}^*, M^*, \sigma^*, M^*, \sigma) = 0$
  $\underline{(3)\ (*, \mathsf{bsn}^*, M^*, \sigma^*) \in L_{msg}}$
  (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 $\mathcal{C}(\mathsf{gsk}, \cdot)$

Upon query $(\texttt{honest\_join}, i)$:
  if $\exists (i, *, *, *, *) \in L_{usr} \cup L_{corr}$ **then return** $\perp$;
  $\langle b, b, \mathsf{sk}_i \rangle \leftarrow \mathsf{Join}_{\mathcal{C},\mathcal{C}} \langle (\mathsf{gpk}, \mathsf{isk}), \mathsf{gpk} \rangle$; Set $\mathsf{tk}_i \leftarrow \mathsf{sk}_i$;
  $L_{usr} \leftarrow L_{usr} \cup (i, \mathsf{tk}_i)$.

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

Upon query $(\texttt{sign}, i, \mathsf{bsn}, M, \mathsf{SigRL})$:
  Retrieve the tuple $(i, \mathcal{M}_i, \mathsf{state}_i, \mathsf{svt}_i, \mathsf{tk}_i)$ from L_{usr}; if not found **return** $\perp$;
  $(\sigma', \pi'_\sigma, \mathsf{state}'_i) \leftarrow \mathcal{M}_i.\mathsf{Sig}(\mathsf{state}_i, \mathsf{bsn}, M, \mathsf{SigRL})$;
  if $\mathsf{svt}_i \neq \perp$ **then** $\sigma \leftarrow \mathsf{Sanitize}(\mathsf{gpk}, \mathsf{bsn}, M, (\sigma', \pi'_\sigma), \mathsf{SigRL}, \mathsf{svt}_i)$; **else** $\sigma \leftarrow \sigma'$
  $L_{msg} \leftarrow L_{msg} \cup (i, \mathsf{bsn}, M, \sigma)$; update$(i, \mathcal{M}_i, \mathsf{state}'_i, \mathsf{svt}_i, \mathsf{tk}_i)$; **return** σ.

Upon query $(\texttt{corrupt}, i)$:
  Retrieve (i, sk_i) from L_{usr}; move the tuple from L_{usr} to L_{cor};
  return sk_i.

Fig. 5: Unforgeability experiments for EPID. 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. We give two experiments, the strong-unforgeability experiment is the same as the unforgeability experiment expect that the condition (3) is substituted with the underlined condition (3).

– At Join-protocol run the honest protocol, at the end of the protocol obtain sk', discard the secret key and instead use $\tilde{\mathsf{sk}}$.
– Upon signature request on input $\mathsf{bsn}, M, \mathsf{SigRL}$ run the honest signature procedure with secret key $\tilde{\mathsf{sk}}$.
  Namely, output $\sigma \xleftarrow{\$} \mathsf{Sig}(\mathsf{gpk}, \tilde{\mathsf{sk}}, \mathsf{bsn}, M, \mathsf{SigRL})$,

Now we describe the attacker.

Attacker $\mathcal{A}$

– Upon input pub generate $\mathsf{isk}, \mathsf{gpk}$ honestly and return gpk.
– Upon input gpk and oracle access to the challenger,
  • sample a secret key $\tilde{\mathsf{sk}}$ valid for gpk (it can be done running the join protocol with itself), run an honest join protocol with subverted machine $\mathcal{M}_{\tilde{\mathsf{sk}}}$ with index 0,
  • run an honest join protocol with honest machine with index 1
  • output $(0, 0, 0, 1, \emptyset)$, namely require as challenge a signature on bsn, M equal to 0,0 for indexes $0, 1$ and with SigRL empty.
– Upon input the challenge signature σ^*,
  • If the SR-EPID has linking correctness then generate a signature $\tilde{\sigma}$ using the secret key $\tilde{\mathsf{sk}}$ on basename 0 and message 1 and output $\mathsf{Link}(\mathsf{gpk}, 0, 0, \sigma^*, 1, \tilde{\sigma}, \tilde{\mathsf{sk}})$.
  • If the SR-EPID has correctness with revocation lists output $\mathsf{Ver}(\mathsf{gpk}, 0, 0, \sigma^*, \emptyset, \{\tilde{\mathsf{sk}}\}) = 0$.

The adversary wins with probability 1. To see this notice that the subverted machine would output a signature σ^* that verifies because it runs the honest signature algorithm on an honestly generated secret key. Therefore the output of the sanitizer will be different than $\perp$ if the signature is generated either with $\tilde{\mathsf{sk}}$ or with the honestly generated key sk'. Thus, if SR-EPID has linking correctness then when $b = 1$ the challenge signature and the signature $\tilde{\sigma}$, that are generated with the same secret key and for the same basename, would link. If the SR-EPID is correct with revocation list then when $b = 1$ the verification algorithm with the secret key $\tilde{\mathsf{sk}}$ blacklisted would output 0.