

Censorship Resistance in Credential Systems and Application to the EUDI Wallet

Pierpaolo Della Monica[✉], Ivan Visconti[✉], Andrea Vitaletti[✉], and Marco Zecchini[✉]

Department of Computer, Control & Management Engineering
Sapienza University of Rome, Rome, Italy
{dellamonica,visconti,vitaletti,zecchini}@diag.uniroma1.it

Abstract. Privacy-preserving credential systems typically focus on data protection during the presentation phase, overlooking potential privacy threats during issuance. A malicious issuer could use the non-minimal information it obtains to deny issuance selectively. Such a denial, going beyond the eligibility policy, is a form of censorship.

In this work, we revisit the security of anonymous credential systems to protect users from malicious issuers during credential issuance. We first highlight the limitations of existing definitions and introduce a security model capturing censorship resistance. Next, we present a round-optimal (i.e., one message per party) construction built from blind signatures and specifically tailored for selective disclosure in the presentation phase. Finally, we present CRACS, an instantiation of our construction, deployable within the European Digital Identity Wallet (EUDIW) framework. CRACS improves on the current reference implementation of the EUDIW, maintaining selective disclosure and yielding unlinkability between presentation and issuance sessions in addition to censorship resistance.

Unlike prior work, CRACS confines advanced cryptography to issuance, leaving secure elements, the standardized credential formats, signature schemes, and presentation mechanisms untouched. It can therefore be deployed incrementally alongside current EUDIW implementations, issuer by issuer and user by user, with no change for verifiers.

Table of Contents

1	Introduction	3
1.1	Technical Overview	6
2	Preliminaries	9
3	Censorship-Resistant Anonymous Credentials	9
3.1	Definition of Censorship-Resistant Anonymous Credentials	9
4	An RSA-based CRAC Construction	18
4.1	Security Analysis	22
4.2	On the Choice of RSA-PSS vs ECDSA	26
5	Certificates, Policies and EUDI-Compatible Credentials	26
5.1	On Certificates and Policies	26
5.2	CRACS: CRAC_{RSA} for EUDI-Compatible Credentials	27
A	EUDI Cryptographic Signatures for Credentials	34
B	Related Work	35
C	Experiments	37
C.1	Main Technical Choices	39
C.2	Experimental Results	39
D	Additional Preliminaries	39
D.1	Signature Scheme	39
D.2	RSA Signature	40
D.3	Commitment Scheme	41
D.4	Non-Interactive Proof Systems	43
E	Security Proofs	44
E.1	Proof of Thm. 1	44
E.2	Proof of Thm. 2	44
E.3	Proof of Thm. 3	46
E.4	Proof of Thm. 4	49
F	On Revocation in CRACS	53

1 Introduction

Credential systems let users prove verifiable attributes¹ about themselves and are seeing rapid deployment in national digital identity programmes. A credential is a signed attestation about a sequence of attributes encoded in some well-defined (e.g., SD-JWT VC, ISO mdoc/mDL) format, and associated with secrets known to the user only. A designated entity, the issuer (e.g., university, government agency, commercial entity), issues a credential upon request of a user, after making sure that attributes appearing in the credential are correct according to some policy. The user can subsequently present attributes encoded in a credential to a verifier.

EUDIW. A real-world large-scale credential system is being deployed across Europe². The European Union has established an official architecture for digital identity and attribute management to be implemented across all Member States through the European Digital Identity Wallet (EUDIW) framework. The Architecture and Reference Framework (ARF) [EUD] and the implementing regulations specify the technical standards (e.g., standardized credential formats and signature schemes) required to build compliant wallet components³. Following the updated eIDAS Regulation, Member States are required to make at least one compliant EUDIW available to citizens by December 2026.

Privacy requirements. The importance of privacy in credential systems dates back to the work of Chaum [Cha85] who introduced the concept of *anonymous credentials* to allow a user to get a credential from an issuer and to present it to verifiers without the issuer and verifiers being able to link the issuance session to subsequent presentations, or presentations to each other. The term *selective disclosure* has recently gained popularity and refers to privacy-preserving presentation mechanisms enabling users to reveal only a chosen subset of attributes to a verifier, who can still verify their authenticity.

The eIDAS Regulation, as amended in [Eur24], requires the EUDIW to incorporate privacy-preserving measures, including selective disclosure of personal data (*Article 5a(4)(a)*) and privacy-preserving techniques preventing transactions or user behaviour from being tracked, linked or correlated (*Article 5a(16)*).

We highlight that several papers have pointed out privacy and security issues of the reference implementations and have proposed alternative systems [CAHLT25, LSZ26, FHLL26, KS25, FFK⁺26, FFL26, BBL⁺25, Fs26, BL26a, MLs25, CLLZ26]. We refer the reader to App. B for a deeper discussion of these works.

An overlooked property: censorship resistance. The EUDIW framework relies on a restricted set of authorized issuers that establish the content of the credentials that they issue in accordance with standards referenced by the ARF⁴. Typically, an EUDIW issuer holds complete control over credential generation. This process usually takes place after verifying the correctness of the attributes that are supposed to appear in the credential. After collecting all such data from users requesting credentials, a malicious issuer can adaptively refuse to issue a credential to a specific user, effectively exercising censorship.

Risks of censorship by malicious issuers are not confined to the EUDIW framework but arise in general also in anonymous credential systems. Indeed, in general, formal definitions over the

¹Throughout the paper, we use the term *attribute* for the value stored in a position of the credential, and the term *field* for the position itself. For instance, in a credential recording the nationality of its holder, the field is the nationality and the attribute is, say, US.

²Other non-EU countries (e.g., CH, JP, UK) are also adopting similar systems.

³We consider version 3.0.0, the latest version available at the time of writing.

⁴Credentials are issued according to the OpenID4VCI standard [Ope]; in particular, OpenID4VCI specifies the message flows and operations used for issuing a credential in [Ope, Section 3.4], and how a credential is created and formatted in [Ope, Section 8.3].

years [CL01, CDHK15, BL13, KLR23, BLNS23, KS25] do not protect the content of the credential, and they do not explicitly formalize the required security against risks due to additional information the issuer may collect to assess the truthfulness of the encoded attributes. Consequently, prior work implicitly trusts the issuer’s decision to refuse issuance based on the (excessive) collected information.

To the best of our knowledge, the only exceptions are the very recent works of [CAHLT25] and [FHLL26] that, following the spirit of predicate blind signatures proposed in [FW24], define the issuing phase so that attributes remain hidden to the (possibly malicious) issuer that however can still assess that, according to some policy, the attributes that will be encoded in the credential are correct. However, those two works only reduce the risk of censorship since the issuer learns that a user got a credential whose attributes satisfy a certain policy. This leak can for example be used to dynamically change the policy and adaptively refuse to issue more credentials for some specific policy⁵. Notice that issuers can always refuse to issue a credential to an identified user that is blacklisted, this is outside our model and unavoidable; what censorship resistance should rule out is refusal based on attributes and any other auxiliary information that the issuer does not inherently need to see.

Summing up, to the best of our knowledge prior security definitions of anonymous credential systems do not prevent an adversarial issuer from obtaining information beyond the minimal one, which, in turn, can be exploited by the issuer to censor a credential request of a user.

Data minimization failure. Without addressing the above potential leaks of confidential information, credential systems could facilitate surveillance. Indeed, in some scenarios, credential issuance could be enforced to pass through a small set of centralized entities that can therefore accumulate significant user auxiliary data. Even setting aside mass surveillance, centralizing such sensitive data creates high-value targets susceptible to data breaches and insider threats.

Device binding in the EUDIW. Besides privacy, an important concern of EUDIW regulation is that credentials be tied to individuals in a way that prevents their cloning or transfer. As a result, the ARF enforces the need for binding credentials to unclonable keys via *device binding*: to realize it, the ARF mandates that the credential be cryptographically tied to a secret key generated and stored in a secure element on the user’s device. The secure element computes a fresh signature during issuance and presentation.

Beyond censorship: the lack of standards-compliant presentations. Existing constructions of anonymous credential systems achieve unlinkability by adopting a credential format and/or a presentation mechanism and/or requirements for secure elements that are not foreseen in government-controlled applications like the EUDIW (for more details see App. B).

EUDIW: the open problem. The discussion so far motivates the following natural requirements for a compliant instantiation of the EUDIW.

1. Issuers should collect only the minimal information needed to check the attributes encoded in a credential.
2. The generation and presentation of credentials should stick with standardized credentials such as ISO mdoc/mDL and SD-JWT VC, instantiated with the EUDI-compliant signature schemes.
3. Device binding should be enforced sticking with current secure elements.

⁵In some applications when censorship-resistance is not considered a danger, such a leak can be benign since it can allow issuers to adapt the policy according to the credentials successfully issued so far.

4. Issue-show unlinkability should be guaranteed, whenever non-inherent, so that an adversary controlling both a verifier and an issuer cannot link an issuance session to a presentation session (where the same credential is used).
5. Show-show unlinkability should be guaranteed, whenever non-inherent, so that an adversary controlling a verifier (and possibly also an issuer) cannot link two presentations based on the same credential.

Interestingly, no known system meets all these requirements: indeed, in previous constructions either 1) issue-show unlinkability is not achieved (this is the case for the main ongoing implementations of EUDIWs provided by Member States), or 2) device binding is not obtained via existing secure elements (e.g., requiring BBS+ signatures) or 3) non-standardized credentials or presentations are required therefore imposing a major and thus problematic update (e.g., Longfellow-ZK). Further details are discussed in App. B.

Our contribution. In this work, we give a new definition for an anonymous credential system that also requires censorship resistance (and thus privacy) against malicious issuers. In our definition of a Censorship-Resistant Anonymous Credential (CRAC) system, the issuer obtains only minimal information about the credential obtained by the user. Moreover, the malicious issuer does not even learn whether the user ended up with a credential or not. These two protections together correspond to what we define as censorship resistance; indeed, we essentially leave to a malicious issuer only inherent information about the user, and thus the only censorship that can be applied is the unavoidable one. The obtained credentials allow the user to selectively disclose the attributes. By restricting the presentation phase to selective disclosure, rather than supporting arbitrary predicates over attributes, our definition admits constructions that are consistent with standardized credential formats and presentation mechanisms.

Next, we give a construction focusing on RSA⁶ signatures since they simplify the issuance phase (just one message per party), making the entire construction easier to read and analyze, while remaining consistent with certain EUDIW deployments (more in App. A).

We then focus on instantiating our credential system by obtaining CRACS (CRAC Standardized), the first credential system in the EUDIW framework (i.e., sticking with the approved credential formats/presentations and signature schemes) that provides issue-show unlinkability.

We provide a basic experimental evaluation of the issuance phase, which is the expensive part. We implemented a toy example where a user possessing some certified data obtains from the issuer an SD-JWT-like credential with a specific attribute that was among the certified data. The sole goal of this experiment was to assess that our system can admit practical instantiations (see App. C).

CRACS as a plug-in for the EUDIW. We stress that CRACS can coexist with current reference implementations of the EUDIW. Verifiers (i.e., relying parties) require no updates, since neither the credential format nor the presentation mechanism changes. Migration is negotiated per issuance session between a wallet and an issuer: an issuer may continue with traditional, non-private issuance while another offers CRACS, and both operate within the same ecosystem. Likewise, since CRACS works with currently deployed secure elements, a user may stay with a basic wallet app (non-private and censorable) or switch to one supporting CRACS without changing their smartphone, provided that the device can meet⁷ the computational and memory requirements of the issuance phase.

⁶See Sec. 4.2 and paragraph “Remark on ECDSA vs RSA” before Sec. 1.1.

⁷Note that our experiments in App. C show that these requirements are modest for the real-world scenarios considered, suggesting that issuance should be feasible on a broad range of devices (i.e., 3 GB of RAM and a couple of seconds).

Remark on revocation. In App. F we also briefly discuss some revocation approaches. Credential revocation is a notorious complication affecting all constructions that achieve issue-show unlinkability. Being a well-studied topic, we do not develop these mechanisms further and focus on the scope of the present work; we refer to [FLS⁺26] and the references therein for a comparison of the revocation mechanisms considered for the EUDIW.

Remark on ECDSA vs RSA. While our construction explicitly uses RSA-PSS (i.e., one of the signature schemes admitted by the EUDIW), there is no technical barrier to using ECDSA instead. Indeed, the blind ECDSA protocol of Maire and Pulval-Dady [MPD25] can take the place of blind RSA-PSS in our construction. We have explicitly used RSA-PSS because its simpler structure makes issuance simpler and round-optimal, with a stateless issuer during the issuance session. Such simplifications make the protocol easier to describe and analyze. Nevertheless, our construction can be instantiated by replacing RSA-PSS with ECDSA. Further discussion can be found in Sec. 4.2.

1.1 Technical Overview

We now give a high-level description of our work, following the structure of the paper: the definition (Sec. 3.1), the construction and its security analysis (Sec. 4), and the instantiation in the EUDIW (Sec. 5). From now on, we will refer to the user as the “client”.

The definition. A censorship-resistant credential system involves an issuer, a client and a verifier, and is parameterized by a policy compiler $\mathbf{P}$ ⁸. $\mathbf{P}$ on input a policy $\mathbf{pol}$ known to the issuer and the client, a private certificate $\mathbf{cert}$ known to the client and a vector of private attributes $\mathbf{attrs}$ known to the client, decides whether the certificate justifies the issuing of a credential to the client with those attributes. The certificate is external evidence (e.g., a passport, or data received over an authenticated channel) and is processed only at issuance; the attributes, together with some secret information, form the content of the credential, and are what the client later (partially) discloses. Each issuance and each presentation⁹ is run with respect to an *attribute-wise* predicate, which specifies, for each field it concerns, a set of admissible attributes. More specifically, in the issuance phase an issuance predicate specifies the possible attributes the issuer is willing to encode in the credential, and in the presentation phase, the verification predicate specifies the attributes allowing for which the presentation succeeds.

For instance, consider an issuer that certifies the nationality of the holder and whether the holder is of age, but issues credentials to adults only. The policy $\mathbf{pol}$ requires a passport carrying a valid signature of a recognized national authority, and checks that the attributes in $\mathbf{attrs}$ are consistent with it (i.e., that the nationality is the one written in the passport and that the attribute in the age field matches its date of birth); the issuance predicate ϕ_{Iss} admits only the attribute **yes** in the age field, and imposes no condition on the nationality field. A client holding the passport of an adult US citizen thus obtains a credential on the attributes $\mathbf{attrs} = (\text{US}, \text{yes})$, whereas a client holding the valid passport of a minor satisfies the policy but not the issuance predicate, since its attribute in the age field is **no**, and so it cannot obtain a credential on those attributes.

We explicitly require three security properties.

(Credential-unlinkability) It concerns the showing phase (i.e., what we informally called issue-show unlinkability earlier): an adversary that chooses the issuer key and all inputs, runs two issuance sessions against honest clients and then sees one showing, cannot tell which session it came from.

⁸We use a policy compiler $\mathbf{P}$ so that the definition is not tied to a specific policy, but applies to arbitrary ones.

⁹We will use the terms “presentation” and “showing” interchangeably.

(Censorship-resistance) It concerns the issuance phase and is our main novelty. This property aims at guaranteeing that the information disclosed to a malicious issuer during issuance should be only the inherent one. The malicious issuer should not get additional information and this can be naturally formalized using the simulation paradigm. We take this path in a strong sense. Indeed we do not want the issuer to even learn whether the client’s input included a valid certificate or not. As a byproduct this property guarantees that the malicious issuer does not get any valid information to convince others about the successful issuance of credentials on some policies/predicates. More technically, in order to allow general-purpose constructions, we require that for every malicious issuer $\mathcal{A}$ there exists a simulator Sim that, given only the policy and the issuance predicate of each session, reproduces the view that $\mathcal{A}$ obtains when interacting with honest clients. One might have a definition allowing a simulator to control some trusted parameters or a random oracle, but it is known that this simplification for the simulator corresponds to a lack of deniability, which is something we do not want to happen (i.e., the malicious issuer should not get evidence that can be shown to others about the successful issuance of a credential). Therefore, crucially, in our definition the simulator has no control over the public setup or the random oracle, in the spirit of [Pas04] (more details in Sec. 3.1), thereby guaranteeing deniability and in turn maximizing the protection for the client.

(Credential-knowledge) It is our soundness notion. Specifically, we need to ensure that a client cannot forge credentials. This means that, from every completed issuance session, a client must either hold a certificate justifying a set of legitimate attributes under the policy and the issuance predicate of that session, or the attributes are $\perp$.

Whatever an honest verifier accepts must trace back to one of the sessions where attributes are not $\perp$. To enforce this behavior, we make use of an extractor that, from every completed issuance session, recovers from the adversary a certificate and the attributes. We also let the adversary act as the verifier when honest clients show (part of) the attributes of a credential, so that credential-knowledge additionally guarantees that an adversary cannot turn a showing it has observed into a different valid one (e.g., by changing a revealed attribute). The only exception is plain replay: re-sending to a verifier attributes that an honest client has already revealed is not a forgery, and is not counted as an attack.

Note that replay attacks are easily prevented in practice by binding each presentation to context data that is freshly generated for every showing, and by checking this binding at verification time. For instance, the EUDIW requires credentials to be bound to the client’s device, via a key pair generated inside its secure element, and requires the client to prove possession of that device during the showing phase by signing, together with the rest of the presentation, a fresh challenge issued by the verifier. This challenge serves as context data and is valid only for that interaction, so a presentation recorded in one interaction cannot be replayed in another (see Sec. 5.2).

On show-show unlinkability. The above explicit security properties do not cover the case in which a credential system allows linking presentation sessions in which the client uses the same credential twice. We do not explicitly model this form of unlinkability because of the inherent impossibility of obtaining this property when the presentation is aligned with standardized credentials (i.e., signed list of committed attributes). The client, on its own, is unable to compute new ECDSA/RSA-PSS signatures that verify under the public key of the issuer. While this might seem quite bad in theory, there is a practical mitigation that is very effective and indeed has been adopted in the context of the EUDIW: clients get batches of independently generated credentials for an issuer so that each credential is spent only once, therefore it is used in a single presentation. Our construction can directly use this batching technique to achieve show-show unlinkability, and therefore we leave it implicit. The fact that this approach has already been adopted in the EUDIW (even though in their

case the adversary is limited to verifiers while in our case security is stronger since the adversary can control both verifiers and issuers) confirms that the burden of having batches of credentials is within the bounds of secure elements and of natural use cases including both online and offline presentations.

The construction. Our construction CRAC_{RSA} is round-optimal and built from three ingredients: a commitment scheme, a NIZK and RSA-PSS. The client commits to each attribute, concatenates the commitments into a message m , PSS-encodes it and blinds the encoding as t , as in blind RSA [Lys23]. It sends t together with a NIZK proof guaranteeing three things: that t is well-formed (i.e., that it is the blinding of the PSS encoding of a message m obtained by concatenating the commitments); that the client knows the attributes **attrs** to which those commitments open, together with the corresponding commitment openings; and that *either* it knows a **cert** justifying **attrs** under the policy **pol** with **attrs** satisfying the issuance predicate, *or* every attribute equals the reserved symbol $\perp$. Note that the issuer learns neither the commitments nor the attributes. That is, both appear only in the witness of the proof, and m is hidden inside t by the blinding. The issuer verifies the proof and, statelessly and using its RSA secret key d , computes $t^d \bmod N$; the client unblinds it and checks that the result is a valid RSA-PSS signature on m . A showing consists of the commitments, the signature and the openings of the disclosed fields, and the verifier performs a standard RSA-PSS verification plus a recomputation of the disclosed commitments.¹⁰

The second branch of the OR, which we call the trapdoor branch, is what makes censorship-resistance provable without any setup power: the simulator is the honest client run on **cert** = $\perp$ and a vector of attributes made of all $\perp$ and the proof is a hybrid argument that moves from the honest branch to the trapdoor branch. Such credentials made of all $\perp$ are harmless, since $\perp$ is admissible for no verification predicate and is thus rejected by every verifier.

The instantiation. The verifier’s checks consist of hash computations and a standard signature verification, which is what allows us to plug CRAC_{RSA} into the EUDIW. SD-JWT VC and mdoc encode attributes as salted hashes, with the salts of the disclosed fields revealed at presentation (App. A), so we are forced to instantiate the commitment with precisely that salted-hash commitment (more details in App. D.3). Device binding is obtained by reserving a field in the credential for the public key of the holder’s secure element. That this key genuinely lives in the secure element is vouched for by a Key Attestation (i.e., a signed statement certifying that the corresponding secret key was generated inside the secure element and cannot be extracted from it). In the ARF issuance flow the wallet hands this Key Attestation to the issuer; we do not, since doing so would defeat censorship-resistance: a Key Attestation is tied to a specific wallet, so handing it over would let the issuer recognize the client and discriminate against it. It rather enters the witness of the NIZK, so that the issuer only learns that *some* valid Key Attestation exists. Proof of possession at showing time is then a device signature, under the secret key matching the public key in that reserved field, on the verifier’s challenge together with the showing message. Note that this is the mirror image of the approach of Frigo and shelat [Fs26], who leave issuance untouched and move all the zero-knowledge machinery to the presentation. Show-show unlinkability is obtained through batch issuance, as the ARF already envisions.

¹⁰Note that the NIZK proves statements about the PSS encoding and hence about the hash functions it use. We therefore treat these hash functions as concrete, publicly known functions that can be expressed as a circuit, and not as random oracles. As a consequence, we assume the security of RSA-PSS instantiated with a fixed hash function, as in [DVVZ26, FW24, DV26].

2 Preliminaries

Notation. Given $n \in \mathbb{N}^+$, we write $[n] := \{1, \dots, n\}$. For an index set I and a vector $\mathbf{a} = (a_i)_{i \in I}$, we write a_i for its i -th component. When x is sampled uniformly at random from a finite set $\mathcal{X}$, we write $x \leftarrow \$ \mathcal{X}$. For an algorithm $\mathcal{A}$, we write $y \leftarrow \mathcal{A}(x)$ to denote an execution of $\mathcal{A}$ on input x with output y ; when its random coins are made explicit, we write $\mathcal{A}(x; \rho)$ for $\rho \in \{0, 1\}^*$. Unless otherwise specified, probabilities are taken over all the randomness of the experiment and of the algorithms involved.

We denote the security parameter by $\lambda \in \mathbb{N}$ and its unary encoding by 1^λ . A function $\text{negl}(\lambda)$ is negligible in λ if, for every constant $c > 0$ and all sufficiently large λ , it holds that $\text{negl}(\lambda) \leq \lambda^{-c}$. A machine is probabilistic polynomial time (PPT), or efficient, if its running time is polynomial in its input size. Unless explicitly stated otherwise, all algorithms considered in this work are efficient.

Cryptographic Primitives. Throughout the paper we use the following primitives, whose formal definitions are deferred to App. D. A digital signature scheme $\text{Sig} = (\text{Sig.KG}, \text{Sig.Sig}, \text{Sig.Vry})$, which is required to be correct and unforgeable; since our construction depends on the specific structure of the standardized RSA-PSS instantiation, we describe it in App. D.2. A non-interactive commitment scheme $\text{Com} = (\text{Com.Cmt}, \text{Com.Opn})$ which is required to be correct, hiding, and binding; in App. D.3 we also describe the salted-hash instantiation Com_H prescribed by the EUDIW standards (see App. A), which we use in Sec. 5.2. A non-interactive zero-knowledge proof system $\text{PS} = (\text{PS.Prv}, \text{PS.Vry})$, which is required to be correct, zero-knowledge, and online knowledge-sound. We refer also to [KL14] for these definitions.

3 Censorship-Resistant Anonymous Credentials

We give the definition of a censorship-resistant anonymous credential (CRAC) system. In each interaction, the two involved parties share a policy and a predicate, both of which are public. The parties involved in the scheme are the following: (1) the issuer $\mathcal{I}$ certifies the attributes of a client if they are compliant with the shared issuance predicate and are justified, under the shared policy, by a certificate held by the client (the role of policies and certificates is discussed in Sec. 5.1), (2) the client $\mathcal{C}$ receives a credential on their attributes, certified by the issuer, such that the certified attributes satisfy the shared issuance predicate, (3) the verifier $\mathcal{V}$ checks that the client holds a credential for attributes satisfying the shared verification predicate.

3.1 Definition of Censorship-Resistant Anonymous Credentials

A censorship-resistant anonymous credential system, supporting $\ell \in \mathbb{N}^+$ attributes, is parameterized by a family of polynomial-time computable policies. Each policy is enforced by a *policy compiler* that is a polynomial-time algorithm P defined as $P(\text{pol}, \text{cert}, \text{attrs}) \rightarrow 0/1$, where $\text{pol} \in \{0, 1\}^*$ is a policy description, $\text{cert} \in \{0, 1\}^*$ is a witness/certificate, and attrs is a vector of attributes. The algorithm outputs 1 if and only if the certificate satisfies the policy and justifies the attributes.

For example, suppose the certificate is a passport, that is, a document signed by a national authority. A policy pol may then require that the passport carry a valid signature of one of the authorities it recognizes, while the fields of the credential are fixed by the deployment, say a field recording the nationality of the holder and one recording whether the holder is of age. The attributes are the values that populate those fields, say **US** in the first and **yes** in the second. On such a pair, P verifies the signature and then checks that the value in the nationality field is the one written in the passport and that the value in the age field agrees with the date of birth.

Attributes and predicates. A credential is issued on a vector of attributes $\mathbf{attrs} = (\mathbf{a}_i)_{i \in [\ell]}$, with $\mathbf{a}_i \in \{0, 1\}^* \setminus \{\perp\}$, where $\perp$ is a reserved string that is never a valid attribute, and the index i identifies a publicly known field of the credential. The fields are public, whereas the attributes are not disclosed to the issuer. Throughout, we reserve the word *attribute* for a value and the word *field* for the position it occupies, so that a credential has ℓ fields and the attributes are the ℓ values that populate them.

We restrict our attention to *attribute-wise* predicates, which we describe field by field through their admissible values. A predicate ϕ consists of a non-empty index set $\mathbf{flds}(\phi) \subseteq [\ell]$, the fields on which ϕ is defined, together with a family of polynomial-time decidable sets $\mathcal{S}_i^\phi \subseteq \{0, 1\}^* \setminus \{\perp\}$, one for each $i \in \mathbf{flds}(\phi)$, and is evaluated as

$$\phi(\mathbf{attrs}) = 1 \iff \mathbf{a}_i \in \mathcal{S}_i^\phi \text{ for all } i \in \mathbf{flds}(\phi).$$

In particular, no predicate admits $\perp$, so $\phi(\mathbf{attrs}) = 0$ whenever $\mathbf{a}_i = \perp$ for some $i \in \mathbf{flds}(\phi)$. We denote by Φ_ℓ the class of such predicates. Note that ϕ ignores every field outside $\mathbf{flds}(\phi)$: if two attribute vectors $\mathbf{attrs}$ and $\mathbf{attrs}'$ coincide on $\mathbf{flds}(\phi)$, i.e., $\mathbf{a}_i = \mathbf{a}'_i$ for every $i \in \mathbf{flds}(\phi)$, then $\phi(\mathbf{attrs}) = \phi(\mathbf{attrs}')$. In particular, when a credential is shown for a predicate that is defined on a subset of the fields, the attributes of the remaining fields do not influence the final output.

Every execution of the issuance protocol is run with respect to an issuance predicate $\phi_{\text{Iss}} \in \Phi_\ell$, and every execution of the verification protocol is run with respect to a verification predicate $\phi_{\text{Vry}} \in \Phi_\ell$. The issuance predicate ϕ_{Iss} declares the values that the issuer is willing to certify in each field, and we assume without loss of generality that $\mathbf{flds}(\phi_{\text{Iss}}) = [\ell]$, setting $\mathcal{S}_i^{\phi_{\text{Iss}}} = \{0, 1\}^* \setminus \{\perp\}$ for the fields on which the issuer imposes no condition. The verification predicate ϕ_{Vry} is in general defined on the same (or fewer) fields, and we require it to admit only values that the issuer certifies:

$$\phi_{\text{Vry}} \preceq \phi_{\text{Iss}} \iff \mathcal{S}_i^{\phi_{\text{Vry}}} \subseteq \mathcal{S}_i^{\phi_{\text{Iss}}} \text{ for all } i \in \mathbf{flds}(\phi_{\text{Vry}}).$$

For example, if field i records the holder's "color", the issuer may certify it with $\mathcal{S}_i^{\phi_{\text{Iss}}}$ equal to $\{\mathbf{red}, \mathbf{blue}\}$, while a verifier interested only in "red" holders sets $\mathcal{S}_i^{\phi_{\text{Vry}}}$ equal to $\{\mathbf{red}\}$ and leaves the remaining fields unconstrained.

On the expressiveness of the predicate class. Our notion targets selective disclosure. Namely, at showing time the client reveals the attributes in the fields of $\mathbf{flds}(\phi_{\text{Vry}})$, which the verifier learns, while the other attributes stay hidden. The admissible sets $\mathcal{S}_i^{\phi_{\text{Vry}}}$ are simply the rule the verifier uses to decide whether to accept the revealed attributes. In the running example, a verifier may accept any nationality, or accept only **yes** in the age field.

We describe predicates field by field so that the relation $\phi_{\text{Vry}} \preceq \phi_{\text{Iss}}$ is easy to check, i.e., it reduces to a set inclusion on each field. Note that this does not prevent a verifier from checking relations between several fields, since it learns their values anyway. Finally, this applies to both issuance and verification predicates, since both belong to Φ_ℓ .

Syntax. A censorship-resistant anonymous credential system for a number of attributes $\ell \in \mathbb{N}^+$ and a policy compiler $\mathbf{P}$ is a tuple of efficient algorithms $\text{CRAC} = (\text{CRAC.KG}, \text{CRAC.Issue}, \text{CRAC.Vry})$ where:

- $\text{CRAC.KG}(1^\lambda) \rightarrow (\mathbf{vk}, \mathbf{ik})$, on input the security parameter, outputs a (secret) issuer key $\mathbf{ik}$ and a (public) verification key $\mathbf{vk}$. Without loss of generality, we assume that $\mathbf{ik}$ includes $\mathbf{vk}$.
- $\text{CRAC.Issue}(\mathcal{I}(\mathbf{ik}, \mathbf{pol}, \phi_{\text{Iss}}), \mathcal{C}(\mathbf{vk}, \mathbf{pol}, \phi_{\text{Iss}}, \mathbf{cert}, \mathbf{attrs})) \rightarrow \mathbf{cred}$ is an *interactive protocol*, run between the issuer $\mathcal{I}$ and the client $\mathcal{C}$, with shared input the verification key $\mathbf{vk}$ (implicit in $\mathbf{ik}$), a policy $\mathbf{pol} \in \{0, 1\}^*$, and a predicate $\phi_{\text{Iss}} \in \Phi_\ell$. The issuer's private input is the issuer key $\mathbf{ik}$; the client's private inputs are a certificate $\mathbf{cert} \in \{0, 1\}^*$ and a vector of attributes $\mathbf{attrs}$.

If the interaction completes successfully, the client obtains a credential **cred** on the attributes **attrs**, which is used at showing time; otherwise, the client outputs $\perp$. The issuer has no output. We use the issuer $\mathcal{I}$ and the client $\mathcal{C}$ as a shorthand for the several algorithms that are executed depending on the input received during the interaction, namely $\mathcal{I}$ represents $(\text{CRAC.Issuer}_0, \dots, \text{CRAC.Issuer}_{k-1})$ and $\mathcal{C}$ respectively $(\text{CRAC.IClient}_0, \dots, \text{CRAC.IClient}_k)$, which depend on the round of interaction of the protocol. For a generic k -round protocol, with $k \in \mathbb{N}^+$, the interaction is defined as follows:

$$\begin{aligned} (\text{pm}_{\mathcal{C}}^0, \text{st}_{\mathcal{C}}^0) &\leftarrow \text{CRAC.IClient}_0(\text{vk}, \text{pol}, \phi_{\text{Iss}}, \text{cert}, \mathbf{attrs}), \\ (\text{pm}_{\mathcal{I}}^0, \text{st}_{\mathcal{I}}^0) &\leftarrow \text{CRAC.Issuer}_0(\text{ik}, \text{pol}, \phi_{\text{Iss}}, \text{pm}_{\mathcal{C}}^0), \\ (\text{pm}_{\mathcal{C}}^1, \text{st}_{\mathcal{C}}^1) &\leftarrow \text{CRAC.IClient}_1(\text{st}_{\mathcal{C}}^0, \text{pm}_{\mathcal{C}}^0), \\ (\text{pm}_{\mathcal{I}}^1, \text{st}_{\mathcal{I}}^1) &\leftarrow \text{CRAC.Issuer}_1(\text{st}_{\mathcal{I}}^0, \text{pm}_{\mathcal{C}}^1), \\ &\dots \\ (\text{pm}_{\mathcal{I}}^{k-1}, \text{st}_{\mathcal{I}}^{k-1}) &\leftarrow \text{CRAC.Issuer}_{k-1}(\text{st}_{\mathcal{I}}^{k-2}, \text{pm}_{\mathcal{C}}^{k-1}), \\ \text{cred} &\leftarrow \text{CRAC.IClient}_k(\text{st}_{\mathcal{C}}^{k-1}, \text{pm}_{\mathcal{I}}^{k-1}). \end{aligned}$$

We write $\text{cred} \leftarrow \text{CRAC.Issue}(\mathcal{I}(\text{ik}, \text{pol}, \phi_{\text{Iss}}), \mathcal{C}(\text{vk}, \text{pol}, \phi_{\text{Iss}}, \text{cert}, \mathbf{attrs}))$ as shorthand for the above sequence.

- $\text{CRAC.Vry}(\mathcal{V}(\text{vk}, \phi_{\text{Vry}}), \mathcal{C}(\text{vk}, \phi_{\text{Vry}}, \mathbf{attrs}, \text{cred})) \rightarrow 0/1$ is an *interactive protocol*, run between the verifier $\mathcal{V}$ and the client $\mathcal{C}$, with shared input the verification key vk and a predicate $\phi_{\text{Vry}} \in \Phi_{\ell}$. We require that $\phi_{\text{Vry}} \preceq \phi_{\text{Iss}}$, where ϕ_{Iss} is the issuance predicate of the session in which **cred** was obtained. The verifier has no private input; the client's private inputs are the attributes **attrs** and the credential **cred**. In the course of the interaction the client reveals its attributes on the fields of ϕ_{Vry} ; we write $\mathbf{opn} = (\mathbf{opn}_i)_{i \in \text{flds}(\phi_{\text{Vry}})}$ for the values so revealed, which are therefore part of the transcript. The verifier outputs a bit, which is 1 if it accepts that the client holds a credential for attributes satisfying ϕ_{Vry} , and 0 otherwise. The client has no output.

As above, we use the verifier $\mathcal{V}$ and the client $\mathcal{C}$ as a shorthand for the several algorithms that are executed depending on the input received during the interaction, namely, for a k' -round protocol, $\mathcal{V}$ represents $(\text{CRAC.Verifier}_0, \dots, \text{CRAC.Verifier}_{k'})$ and $\mathcal{C}$ respectively $(\text{CRAC.VClient}_0, \dots, \text{CRAC.VClient}_{k'-1})$.

We write $b \leftarrow \text{CRAC.Vry}(\mathcal{V}(\text{vk}, \phi_{\text{Vry}}), \mathcal{C}(\text{vk}, \phi_{\text{Vry}}, \mathbf{attrs}, \text{cred}))$ as shorthand for the above sequence.

We require a censorship-resistant anonymous credential system to satisfy correctness, credential-unlinkability, censorship-resistance and credential-knowledge, as defined below.

Note that the certificate and the attributes are defined as two separate inputs because they play different roles. The certificate is the evidence that the client is qualified for a credential and is consumed at issuance, appearing in no later algorithm, whereas the attributes are the content of the credential and persist beyond the session, since they are what the client (selectively) reveals to the verifier at showing time. Moreover, since $\mathbf{P}$ is a relation and not a function, the attributes on which a credential is issued are not uniquely determined by the certificate that justifies them. That is, the same certificate may justify several attribute vectors, so the client chooses one of them and supplies it explicitly, as its private input **attrs** to the issuance protocol.

Note that **cred** (i.e., the output of the interactive issuance) is the private state the client keeps for later showings, not a value sent to the verifier. That is, it may contain secrets and only the showing protocol decides which part of **cred** the verifier sees. The EUDIW standards (i.e., App. A) instead call *credential* the issuer-signed object that is sent to the verifier, which contains no secret; the correspondence between **cred** and that object is made explicit in Sec. 5.2.

Finally, the verifier holds no key of its own, and a showing is checked under the public key vk alone, as is customary for anonymous credentials with publicly verifiable showings. Authenticating the verifier to the client, which the EUDIW requires, is orthogonal to the credential system and is layered on the messages of the showing protocol; we discuss it in Sec. 5.2.

Correctness. Correctness requires that, for every policy pol , an honest client whose certificate and attributes are accepted by the policy compiler, i.e., $P(\text{pol}, \text{cert}, \text{attrs}) = 1$, and whose attributes satisfy the issuance predicate, i.e., $\phi_{\text{Iss}}(\text{attrs}) = 1$, always obtains a credential from an honest issuer, and that an honest verifier's decision on that credential always agrees with the predicate it is shown for. Note that a single credential can be shown for every verification predicate ϕ_{Vry} that refines the issuance predicate, i.e., such that $\phi_{\text{Vry}} \preceq \phi_{\text{Iss}}$. When the certificate does not justify the attributes under the policy, or when the attributes do not satisfy the issuance predicate, the honest issuer is expected to abort, and the definition requires nothing.

Definition 1 (Correctness). *A censorship-resistant anonymous credential system CRAC for a number of attributes $\ell \in \mathbb{N}^+$ and a policy compiler P is perfectly correct if for any $\lambda \in \mathbb{N}$, any $\text{pol} \in \{0, 1\}^*$, $\text{cert} \in \{0, 1\}^*$ and $\text{attrs} \in (\{0, 1\}^*)^\ell$, and any predicates $\phi_{\text{Iss}}, \phi_{\text{Vry}} \in \Phi_\ell$ such that $\phi_{\text{Vry}} \preceq \phi_{\text{Iss}}$, $P(\text{pol}, \text{cert}, \text{attrs}) = 1$ and $\phi_{\text{Iss}}(\text{attrs}) = 1$, the following game returns 1 with probability 1:*

$(\text{vk}, \text{ik}) \leftarrow \text{CRAC.KG}(1^\lambda),$
 $\text{cred} \leftarrow \text{CRAC.Issue}(\mathcal{I}(\text{ik}, \text{pol}, \phi_{\text{Iss}}), \mathcal{C}(\text{vk}, \text{pol}, \phi_{\text{Iss}}, \text{cert}, \text{attrs})),$
 $b \leftarrow \text{CRAC.Vry}(\mathcal{V}(\text{vk}, \phi_{\text{Vry}}), \mathcal{C}(\text{vk}, \phi_{\text{Vry}}, \text{attrs}, \text{cred})),$
Return 1 iff $\text{cred} \neq \perp \wedge \phi_{\text{Vry}}(\text{attrs}) = b$

Credential-Unlinkability. Credential-unlinkability captures the requirement that a shown credential cannot be traced back to the issuing session that produced it. The adversary is defined in the (strong) malicious-issuer model [Fis06]: it chooses the verification key vk itself, so that no honestly generated keys are assumed, and it chooses the issuance predicate, both policies, both certificates and both attribute vectors. Two issuing sessions are then run against honest clients, the id -th on the triple $(\text{pol}_{\text{id}}, \text{cert}_{\text{id}}, \text{attrs}_{\text{id}})$, and the adversary is told which session is which. A hidden bit b selects one of the two resulting credentials, which is then shown to the adversary, now acting as a malicious verifier, and the adversary must decide which session that credential came from.

Since the adversary controls all of the inputs, the game must rule out trivial wins, and this is reflected in two places in the definition. First, each certificate is required to justify its own attributes under its own policy, i.e., $P(\text{pol}_{\text{id}}, \text{cert}_{\text{id}}, \text{attrs}_{\text{id}}) = 1$, and each attribute vector to satisfy the issuance predicate, i.e., $\phi_{\text{Iss}}(\text{attrs}_{\text{id}}) = 1$, for $\text{id} \in \{0, 1\}$. Second, the showing oracle aborts unless both attribute vectors satisfy the verification predicate and agree on its fields. The first of these conditions is needed because, by correctness, the verifier's decision is $\phi_{\text{Vry}}(\text{attrs}_b)$, which is then 1 regardless of b ; the second because a showing reveals in the clear the attributes in the fields of ϕ_{Vry} , so two vectors differing there would be told apart from the revealed attributes alone. On every other field the two vectors are unconstrained.

The definition is a single-show one. That is, the adversary may obtain at most one showing, and every later query to the showing oracle aborts. This is enforced inside the oracle, which sets both credentials to $\perp$ *before* starting the interaction rather than after it ends. This matters because the adversary plays the verifier in that interaction and therefore decides when it terminates. Note that, in case the credentials are discarded at the end, the adversary could leave the first showing open and query the oracle again in the meantime, obtaining two showings of the same credential.

Definition 2 (Credential-Unlinkability). *Consider the ULK game, described in Fig. 1. A censorship-resistant anonymous credential system CRAC for a number of attributes $\ell \in \mathbb{N}^+$ and a policy compiler P satisfies computational (single-show) credential-unlinkability if, for every PPT adversary $\mathcal{A}$, the advantage $\text{Adv}_{\text{CRAC}}^{\text{ulk}}(\mathcal{A}, \lambda)$ is negligible in λ , where:*
 $\text{Adv}_{\text{CRAC}}^{\text{ulk}}(\mathcal{A}, \lambda) := \left| \frac{1}{2} - \Pr[\text{ULK}_{\text{CRAC}}^{\mathcal{A}}(\lambda) = 1] \right|.$

Game $\text{ULK}_{\text{CRAC}}^{\mathcal{A}}(\lambda)$

1. On input 1^λ , the adversary $\mathcal{A}$ outputs $(\text{vk}, \phi_{\text{Iss}}, \text{pol}_0, \text{pol}_1, \text{cert}_0, \text{cert}_1, \text{attrs}_0, \text{attrs}_1)$, where $\phi_{\text{Iss}} \in \Phi_\ell$.
2. If there exists $\text{id} \in \{0, 1\}$ such that $\text{P}(\text{pol}_{\text{id}}, \text{cert}_{\text{id}}, \text{attrs}_{\text{id}}) \neq 1$ or $\phi_{\text{Iss}}(\text{attrs}_{\text{id}}) \neq 1$, the game halts and outputs a uniformly random bit.
3. The values $\text{cred}_0, \text{cred}_1$ are initialized to $\perp$.
4. A uniformly random bit $b \leftarrow_{\$} \{0, 1\}$ is sampled.
5. An issuance oracle is initialized with access to vk , and:

On input a session identifier $\text{id} \in \{0, 1\}$ from $\mathcal{A}$, if session id has already been run, the oracle aborts. Otherwise, it runs the honest client algorithms CRAC.IClient on inputs $(\text{vk}, \text{pol}_{\text{id}}, \phi_{\text{Iss}}, \text{cert}_{\text{id}}, \text{attrs}_{\text{id}})$, interacting with the adversary acting as a malicious issuer $\mathcal{I}$, that is, it runs $\text{CRAC.Issue}(\mathcal{A}, \mathcal{C}(\text{vk}, \text{pol}_{\text{id}}, \phi_{\text{Iss}}, \text{cert}_{\text{id}}, \text{attrs}_{\text{id}}))$. It records the client's output as cred_{id} , which is $\perp$ if the client aborts. The output cred_{id} is not given to $\mathcal{A}$.
6. A showing oracle is initialized with access to $(\text{cred}_{\text{id}})_{\text{id} \in \{0, 1\}}$ and:

On input a verification predicate $\phi_{\text{Vry}} \in \Phi_\ell$ from $\mathcal{A}$, such that $\phi_{\text{Vry}} \preceq \phi_{\text{Iss}}$, the oracle first checks whether cred_0 or cred_1 equals $\perp$; if so, the oracle aborts. It then checks whether $\phi_{\text{Vry}}(\text{attrs}_0) = \phi_{\text{Vry}}(\text{attrs}_1) = 1$, and whether $\text{a}_{0,i} = \text{a}_{1,i}$ for every field $i \in \text{flds}(\phi_{\text{Vry}})$, where $\text{attrs}_{\text{id}} = (\text{a}_{\text{id},i})_{i \in [\ell]}$; if not, the oracle aborts. Otherwise, the oracle sets $(\text{attrs}^*, \text{cred}^*) \leftarrow (\text{attrs}_b, \text{cred}_b)$ and then $\text{cred}_0 \leftarrow \perp$ and $\text{cred}_1 \leftarrow \perp$. It then runs the honest client algorithms CRAC.VClient on inputs $(\text{vk}, \phi_{\text{Vry}}, \text{attrs}^*, \text{cred}^*)$, interacting with the adversary acting as a malicious verifier, that is, it runs $\text{CRAC.Vry}(\mathcal{A}, \mathcal{C}(\text{vk}, \phi_{\text{Vry}}, \text{attrs}^*, \text{cred}^*))$.
7. The adversary is given access to both the issuance and the showing oracle. After interacting with them, it outputs a bit b' . The game outputs 1 if $b = b'$, and 0 otherwise.

Fig. 1: The game ULK. The credentials produced by the issuance oracle are never given to the adversary, which learns only what it observes as the malicious verifier of the showing. We implicitly assume that the adversary $\mathcal{A}$ may maintain an internal state.

Note that the restriction to a single showing reflects the fact that show-show unlinkability of a single credential is unattainable with standardized credentials (see Secs. 1.1 and 5.2); the definition nonetheless extends naturally to show-show unlinkability. To this end, it suffices to let the showing oracle answer polynomially many queries and to let the adversary request, besides challenge showings of cred_b , showings of cred_0 or cred_1 of its choice, whose origin is therefore known to it; the adversary still wins by guessing b . An adversary able to tell whether two showings stem from the same credential would then link a challenge showing to one of the known showings and recover b , so the extended game captures issue-show and show-show unlinkability at once.

Censorship-Resistance. Censorship-resistance rules out a malicious issuer that learns *any information* about the certificate used by a client and then selectively refuses service to censor individual clients. The definition requires the issuer's view to reveal no information about the certificate at all, including information that does not, by itself, allow the issuer to distinguish between two specific certificates (i.e., sessions), such as the message subspaces to which they belong. We therefore adopt a simulation-based formulation. In the real world, a malicious issuer interacts with honest clients, each running the issuance protocol on its own policy-certificate pair, and outputs its view. In the ideal world, a simulator with black-box access to the same adversary must reproduce that view given only the policy and the issuance predicate of each session. Since the simulator succeeds without ever

seeing a certificate, no efficient issuer strategy (i.e., no decision to abort or to deviate) can depend on which certificate a client holds.

Crucially, the simulator is not allowed to exploit any additional setup power to reproduce the issuer's view. Any public setup required by CRAC, as well as any random oracle, is initialized independently of $\mathcal{A}$, Sim , and the challenge bit. In particular, Sim may observe random-oracle queries and their answers, but cannot program or otherwise modify the random oracle, nor can it choose or alter the public setup. This restriction is essential to the deniability-like guarantee captured by our definition. That is, the simulated view should be reproducible without the certificate using only the same public resources available in a real execution, rather than by relying on special control over the setup or the random oracle. This follows the same idea of [Pas04] in the context of zero-knowledge.

Definition 3 (Censorship-Resistance). *Consider the CR game, described in Fig. 2. A censorship-resistant anonymous credential system CRAC for a number of attributes $\ell \in \mathbb{N}^+$ and a policy compiler P satisfies computational (adaptive) censorship-resistance if, for every PPT adversary $\mathcal{A}$, there exists a PPT simulator Sim such that for every PPT distinguisher D the advantage $\text{Adv}_{\text{CRAC}}^{\text{cr}}(\mathcal{A}, D, \lambda)$ is negligible in λ , where:*

$$\text{Adv}_{\text{CRAC}}^{\text{cr}}(\mathcal{A}, D, \lambda) := \left| \Pr \left[\text{CR}_{\text{CRAC}}^{\mathcal{A}, \text{Sim}, D, 0}(\lambda) = 1 \right] - \Pr \left[\text{CR}_{\text{CRAC}}^{\mathcal{A}, \text{Sim}, D, 1}(\lambda) = 1 \right] \right|.$$

We further require that Sim has no control over any public setup or random oracle used in CRAC.

Game $\text{CR}_{\text{CRAC}}^{\mathcal{A}, \text{Sim}, D, b}(\lambda)$

The distinguisher D is run on input 1^λ and outputs a verification key $\text{vk} \in \{0, 1\}^*$. It may then exchange messages with the issuer via the issuance oracle only, and finally outputs a bit, which the game outputs.

- **REAL-WORLD**_($b=0$): the issuer is $\mathcal{A}$, run on input $(1^\lambda, \text{vk})$.
 1. An issuance oracle is initialized with access to vk and to a session counter $\text{id} \leftarrow 0$, and is defined as follows:

On input a tuple $(\text{pol}, \phi_{\text{Iss}}, \text{cert}, \text{attrs})$ from D , the oracle returns $\perp$ if $P(\text{pol}, \text{cert}, \text{attrs}) \neq 1$ or $\phi_{\text{Iss}}(\text{attrs}) \neq 1$. Otherwise it increments id , and runs the honest client algorithms $\text{CRAC}.\text{IClient}$ on inputs $(\text{vk}, \text{pol}, \phi_{\text{Iss}}, \text{cert}, \text{attrs})$ in a new session with identifier id , interacting with $\mathcal{A}$ acting as a malicious issuer $\mathcal{I}$, that is, it runs $\text{CRAC}.\text{Issue}(\mathcal{A}, \mathcal{C}(\text{vk}, \text{pol}, \phi_{\text{Iss}}, \text{cert}, \text{attrs}))$. The client's output is discarded.
 2. D is given access to the issuance oracle. Sessions may be interleaved, and each query may depend on all messages D has received so far.
- **IDEAL-WORLD**_($b=1$): the issuer is Sim , run on input $(1^\lambda, \text{vk})$ with access to $\mathcal{A}$.
 1. An issuance oracle is initialized with access to a session counter $\text{id} \leftarrow 0$ and is defined as follows:

On input a tuple $(\text{pol}, \phi_{\text{Iss}}, \text{cert}, \text{attrs})$ from D , the oracle returns $\perp$ if $P(\text{pol}, \text{cert}, \text{attrs}) \neq 1$ or $\phi_{\text{Iss}}(\text{attrs}) \neq 1$. Otherwise it increments id and gives $(\text{id}, \text{pol}, \phi_{\text{Iss}})$ to Sim , discarding cert and attrs .
 2. D is given access to the issuance oracle, under the same conditions as above.

Fig. 2: The game CR. The verification key is chosen by the distinguisher, so neither $\mathcal{A}$ nor Sim controls it. The simulator never receives any certificate: it learns only the policy and the issuance predicate of each session. We implicitly assume that $\mathcal{A}$, Sim and D may each maintain an internal state.

On the requirement of simulatability. We adopt a simulation-based definition rather than an indistinguishability-based one. The reason is that an indistinguishability notion bounds what the issuer can *learn* during a session, but says nothing about what the transcript of that session *is*. A protocol may be indistinguishable in this sense and still be such that every accepting transcript constitutes a proof that the client held a certificate satisfying the policy. An issuer that archives such transcripts can then defer the censorship decision: it may hand them to a party holding side information that it does not itself possess, and act on the outcome in later sessions. Requiring the transcript to be reproducible from the policy alone rules this out, since a transcript that anyone can produce without a certificate attests to nothing.

Moreover, an indistinguishability notion gives no guarantee at all when a policy is satisfied by essentially one certificate. Such a notion would ask the adversary to submit two pairs $(\text{cert}_{\text{id}}, \text{attrs}_{\text{id}})_{\text{id} \in \{0,1\}}$ and to tell which of them an honest client is running on. The two pairs must be valid under the same policy, since policies are public and a difference between them is visible to the issuer; the adversary must therefore exhibit two distinct certificates that same policy accepts. For some applications (e.g., when a policy is typically satisfied by the identity document of a particular holder) no such pair need exist, and the definition guarantees nothing.

Credential-Unlinkability vs Censorship-Resistance. Both properties concern what a malicious issuer can learn about the certificate of an honest client, and it is worth making precise why we separate them. They differ in the phase at which the guarantee applies, and in what is being hidden. Credential-unlinkability is a property of the showing phase. The adversary knows both certificates, since it chose them, and it also runs both issuance sessions; the only thing it must not learn is which of the two sessions produced the credential it is later shown. Censorship-resistance is a property of the issuance phase, and no credential is ever shown: the adversary interacts only with clients running `CRAC.Issue`, and the requirement is that the transcripts of those interactions carry no information about the certificates at all.

Credential-Knowledge. Credential-knowledge imposes two main requirements. First, that an issuer completes an issuance session only if the client knows a certificate and a vector of attributes such that the certificate justifies those attributes under the policy of that session and the attributes satisfy its issuance predicate, and the credential is issued on them. Second, that a client convinces a verifier only through values that are part of a credential issued in some such session. Together, the two requirements say that whatever a verifier accepts can be traced back to a certificate and attributes that the client held at issuance time.

We therefore state the property with respect to an *extractor* `Ext`, which recovers, from the transcript of an issuance session, the certificate and the attributes underlying it. The extractor is run by the issuance oracle itself, on every session that the issuer answered and as soon as that session completes, and what it returns is used only to decide the winning condition.

The difficulty in giving such a definition is that the issuer learns neither the certificate nor the attributes of a session, since issuance happens in a *blind* fashion, i.e., the certificate and the attributes are private inputs of the client, and censorship-resistance requires the issuer’s view to be independent of them. Thus, the transcript of an issuance session hides the certificate and the attributes that were used. Even by inspecting all exchanged messages with the honest issuer, one cannot tell whether the client used a valid tuple of policy, certificate, and attributes, so the winning condition must be stated differently.

The game runs an honest issuer and an honest verifier, and lets the adversary play the client against both. The adversary may open polynomially many sessions with each of them, interleaved as it wishes. There are two ways in which the adversary wins, one for each phase, and we describe them in turn.

ILLEGITIMATE ISSUANCE. The adversary wins if, in some completed issuance session, the extracted certificate does not justify the extracted attributes under the policy of that session, or those attributes do not satisfy its issuance predicate, provided that at least one attribute is different from $\perp$; a session in which every attribute equals $\perp$ yields a credential that no verifier accepts, and is therefore not counted as an attack. The attributes extracted from the remaining sessions are collected in a set $\mathcal{Q}$, which is the record of what the adversary has legitimately been issued.

ILLEGITIMATE SHOWING. Suppose that an honest verifier accepts a verification session; then the values the adversary revealed in that session must match one of the attribute vectors in $\mathcal{Q}$, on the fields of the predicate the session was run on. If they do not, the adversary has convinced the verifier of values that no issuer ever certified to it, and it wins.

The second condition, as stated, would be satisfied by a scheme in which a showing can be modified into a different valid showing, since the adversary of the game never sees a valid showing and could not attempt such an attack in the first place. To rule out this form of malleability, the game gives the adversary two further oracles: one that creates honest clients, and one that lets it act as the verifier in their credential verifications.

Once the adversary may observe a showing, however, one behaviour has to be excluded from the second condition, because it is not an attack. That is, an adversary that repeats to the honest verifier what an honest client already revealed to it is not forging anything, since those values were disclosed to it legitimately. The game therefore records, for each honest client, the pairs of a field and the value revealed in that field during the credential verifications that the adversary observed, and it does not count a showing whose revealed values are among the ones recorded for a single honest client. Everything else is counted. In particular, the adversary wins if it alters a revealed value, if it reveals a field that no honest client ever opened to it, or if it combines values revealed by two different honest clients into a single showing.

Definition 4 (Credential-Knowledge). Consider the KNW game, described in Fig. 3. A censorship-resistant anonymous credential system CRAC for a number of attributes $\ell \in \mathbb{N}^+$ and a policy compiler P satisfies credential-knowledge if, for every PPT adversary $\mathcal{A}$, there exists a PPT extractor Ext such that the advantage $\text{Adv}_{\text{CRAC}}^{\text{knw}}(\mathcal{A}, \text{Ext}, \lambda)$ is negligible in λ , where:

$$\text{Adv}_{\text{CRAC}}^{\text{knw}}(\mathcal{A}, \text{Ext}, \lambda) := \Pr \left[\text{KNW}_{\text{CRAC}}^{\mathcal{A}, \text{Ext}}(\lambda) = 1 \right].$$

Game $\text{KNW}_{\text{CRAC}}^{\mathcal{A}, \text{Ext}}(\lambda)$

1. The game samples $(\text{vk}, \text{ik}) \leftarrow \text{CRAC.KG}(1^\lambda)$, sets $\mathcal{W}, \mathcal{Q} \leftarrow \emptyset$ and $\text{win} \leftarrow 0$.
Each oracle below maintains its own counter, initialized to 0. Whenever an oracle opens a session, it increments its counter and uses the new value as the session identifier, which it returns to $\mathcal{A}$; the adversary carries on that session by addressing it through this identifier. Sessions may be interleaved, and each message may depend on all messages $\mathcal{A}$ has received so far. A party aborts by sending $\perp$, and a session *completes* if neither party aborted.
2. An issuance oracle is initialized with access to ik and to the extractor Ext , and:
On input a policy $\text{pol} \in \{0, 1\}^*$ and an issuance predicate $\phi_{\text{Iss}} \in \Phi_\ell$ from $\mathcal{A}$, the oracle opens a session in which it runs the honest issuer algorithms CRAC.Issuer on inputs $(\text{ik}, \text{pol}, \phi_{\text{Iss}})$, interacting with $\mathcal{A}$ acting as a malicious client.
If the session completes, the oracle runs Ext on the transcript tr of the session, obtaining $(\text{cert}, \text{attrs}) \leftarrow \text{Ext}(\text{vk}, \text{pol}, \phi_{\text{Iss}}, \text{tr})$ with $\text{attrs} = (\mathbf{a}_i)_{i \in [\ell]}$, and sets:
 $\text{win} \leftarrow 1$ if $(P(\text{pol}, \text{cert}, \text{attrs}) = 0 \vee \phi_{\text{Iss}}(\text{attrs}) = 0) \wedge \exists i \in [\ell] : (\mathbf{a}_i \neq \perp)$,
and otherwise sets $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\text{attrs}\}$.
3. An honest-issuance oracle is initialized with access to ik and:

On input a policy $\text{pol} \in \{0,1\}^*$, an issuance predicate $\phi_{\text{Iss}} \in \Phi_\ell$, a certificate $\text{cert} \in \{0,1\}^*$ and a vector of attributes $\mathbf{attrs}$ from $\mathcal{A}$, the oracle aborts if $P(\text{pol}, \text{cert}, \mathbf{attrs}) \neq 1$ or $\phi_{\text{Iss}}(\mathbf{attrs}) \neq 1$. Otherwise, it opens a session and runs an honest issuance $\text{cred} \leftarrow \text{CRAC.Issue}(\mathcal{I}(\text{ik}, \text{pol}, \phi_{\text{Iss}}), \mathcal{C}(\text{vk}, \text{pol}, \phi_{\text{Iss}}, \text{cert}, \mathbf{attrs}))$, in which it plays both parties.

It stores the tuple $(\mathbf{attrs}, \text{cred}, \phi_{\text{Iss}})$ under the identifier cid of the session, writing it $(\mathbf{attrs}_{\text{cid}}, \text{cred}_{\text{cid}}, \phi_{\text{Iss}_{\text{cid}}})$ with $\mathbf{attrs}_{\text{cid}} = (a_{\text{cid},i})_{i \in [\ell]}$, sets $\mathcal{R}_{\text{cid}} \leftarrow \emptyset$, and returns cid to $\mathcal{A}$.

4. An honest-verification oracle is initialized with access to vk and to the entries stored by the honest-issuance oracle, and:

On input a session identifier cid of the honest-issuance oracle and a verification predicate $\phi_{\text{Vry}} \in \Phi_\ell$ from $\mathcal{A}$, the oracle aborts if no entry $(\mathbf{attrs}_{\text{cid}}, \text{cred}_{\text{cid}}, \phi_{\text{Iss}_{\text{cid}}})$ is stored under cid , or if $\phi_{\text{Vry}}(\mathbf{attrs}_{\text{cid}}) \neq 1$, or if $\phi_{\text{Vry}} \not\leq \phi_{\text{Iss}_{\text{cid}}}$. Otherwise it opens a session in which it runs the honest client algorithms CRAC.VClient on inputs $(\text{vk}, \phi_{\text{Vry}}, \mathbf{attrs}_{\text{cid}}, \text{cred}_{\text{cid}})$, interacting with $\mathcal{A}$ acting as a malicious verifier.

If the session completes, the oracle sets $\mathcal{R}_{\text{cid}} \leftarrow \mathcal{R}_{\text{cid}} \cup \{(i, \mathbf{opn}_i) : i \in \text{flds}(\phi_{\text{Vry}})\}$, where $\mathbf{opn} = (\mathbf{opn}_i)_{i \in \text{flds}(\phi_{\text{Vry}})}$ are the values revealed by the honest client during the session.

5. A verification oracle is initialized with access to vk and:

On input a verification predicate $\phi_{\text{Vry}} \in \Phi_\ell$ from $\mathcal{A}$, the oracle opens a session in which it runs the honest verifier algorithms CRAC.Verifier on inputs $(\text{vk}, \phi_{\text{Vry}})$, interacting with $\mathcal{A}$ acting as a malicious client.

If the session completes with output 1, the oracle sets $\mathcal{W} \leftarrow \mathcal{W} \cup \{(\phi_{\text{Vry}}, \mathbf{opn})\}$, where $\mathbf{opn} = (\mathbf{opn}_i)_{i \in \text{flds}(\phi_{\text{Vry}})}$ are the values revealed by $\mathcal{A}$ during the session.

6. $\mathcal{A}$ is run on input $(1^\lambda, \text{vk})$ and is given access to the four oracles above. When $\mathcal{A}$ halts, the game sets $\text{win} \leftarrow 1$ if $\exists (\phi_{\text{Vry}}, \mathbf{opn}) \in \mathcal{W}$ such that both:
 $\nexists \mathbf{attrs} = (a_i)_{i \in [\ell]} \in \mathcal{Q} : \forall i \in \text{flds}(\phi_{\text{Vry}}), (\mathbf{opn}_i = a_i)$ and $\nexists \text{cid} : \{(i, \mathbf{opn}_i) : i \in \text{flds}(\phi_{\text{Vry}})\} \subseteq \mathcal{R}_{\text{cid}}$, and outputs win.

Fig. 3: The game KNW. The issuer and the verifier are honest, and the adversary plays the client against both, so the verification key is generated by the game. We implicitly assume that $\mathcal{A}$ may maintain an internal state.

On the oracles of KNW. We give a verbal description of the oracles used in our security game in Fig. 3.

ISSUANCE ORACLE. The adversary requests credentials for itself, choosing the policy and the issuance predicate freely. This oracle enforces the first requirement: on every session in which the honest issuer did not abort, the extractor recovers a certificate and a vector of attributes from the transcript, and these must be legitimate. When they are, the attributes are recorded in $\mathcal{Q}$, which is the game's record of what the adversary has been issued.

HONEST-ISSUANCE ORACLE. This oracle creates clients whose credentials are legitimate, by running both the honest issuer and the honest client. The adversary chooses their policy, predicate, certificate, and attributes, and therefore knows exactly what they hold, but it receives only the handle cid and never the credential itself. The oracle produces no attack on its own; it exists to make the next one possible.

HONEST-VERIFICATION ORACLE. Here the roles are reversed: the game plays the client, and the adversary plays the verifier. Nothing in this oracle can make the adversary win. Its purpose is to hand the adversary genuine showings, which is what a malleability attack starts from. Without it, modifying someone else’s showing would not be an attack the adversary could even attempt, and the definition would be silent on malleability. The oracle also records, in $\mathcal{R}_{\text{cid}}$, which values of that client the adversary has seen.

VERIFICATION ORACLE. The adversary chooses the predicate and tries to make the honest verifier accept. Every accepted session is recorded in $\mathcal{W}$, together with the values the adversary revealed, and the adversary wins if, in one of them, those values can be traced neither to a credential it was issued nor to what an honest client revealed to it.

We make the following observations. First, the attributes of the honest clients are *not* recorded in $\mathcal{Q}$, which is reserved for the credentials the adversary obtained itself; only the values they actually revealed are recorded, and in a separate set per client. An adversary that mauls a showing to reveal a value that was never disclosed to it therefore wins, since that value belongs to no $\mathcal{R}_{\text{cid}}$. Second, the sets $\mathcal{R}_{\text{cid}}$ are kept separate for each honest client, rather than merged into one: an adversary that takes some fields from the showing of one honest client and some from that of another, and presents them as a single credential, is not repeating what it was shown, and is counted as an attack. Third, the values of a given client accumulate across the credential verifications the adversary observes, since a client that discloses one field in one session and another field in a later one has legitimately revealed both. Finally, the condition is not an equality, so an adversary that repeats only part of what it was shown is not counted.

4 An RSA-based CRAC Construction

We present a construction of a round-optimal CRAC system whose showing is compatible with verifying a standard RSA-PSS signature [MKJR16, BR98]. More specifically, we give a construction CRAC_{RSA} in which the verifier’s checks fully comply with the standard verification required by EUDIW credentials. The construction uses, as a black box, a commitment scheme Com_{vry} , whose instantiation is discussed separately. We describe CRAC_{RSA} in Fig. 4, and the policy compiler $\mathbf{P}$ and the relation $\mathbf{R}_{\text{CRAC}_{\text{RSA}}}$ in the following.

Recall from Sec. 3.1 that $\perp$ is not a valid attribute and is admitted by no predicate; our construction uses it as the attribute of every field in the Trapdoor Branch below.

Description of the policy compiler $\mathbf{P}$. We define the policy compiler $\mathbf{P}$ as follows. According to the definition in Sec. 3.1, it takes as input a policy $\text{pol} \in \{0, 1\}^*$, a certificate $\text{cert} \in \{0, 1\}^*$ and a vector of attributes $\text{attrs} = (a_i)_{i \in [\ell]}$. The compiler first interprets the policy as a pair of polynomial-size circuits

$$\text{pol} := (\text{Pol}, \text{AttrChk}) : \text{Pol} : \{0, 1\}^* \rightarrow \{0, 1\}, \text{AttrChk} : \{0, 1\}^* \times (\{0, 1\}^*)^\ell \rightarrow \{0, 1\},$$

where Pol decides whether a certificate satisfies the policy, and AttrChk decides whether a certificate justifies a vector of attributes. Then $\mathbf{P}(\text{pol}, \text{cert}, \text{attrs})$ outputs 1 if $\text{Pol}(\text{cert}) = 1$ and $\text{AttrChk}(\text{cert}, \text{attrs}) = 1$, and outputs 0 otherwise.

On the commitment scheme. The construction uses a single commitment scheme, and requires of it hiding and binding, as in App. D.3. The commitment c_{vry} is computed by the client at the very beginning of the issuance protocol and is never seen by the issuer: it enters the protocol only inside the blinded value t and in the witness of the NIZK. It is the object on which the credential is

signed, and the one the verifier receives and checks at showing time, so it is the component that a deployment must align with whatever credential format it has to comply with. The only structural requirement is that the opening relation $\text{Com}_{\text{vry}}.\text{Opn}$ be efficiently provable inside the NIZK, since the relation $\text{R}_{\text{CRAC}_{\text{RSA}}}$ below asserts that $\mathbf{c}_{\text{vry}}$ is opened to the attributes that the certificate justifies. In particular, when the showing has to be compatible with the EUDIW, Com_{vry} can be instantiated with the folklore salted-hash scheme Com_{H} of App. D.3, whose security is stated in Asm. 2.

Description of the relation $\text{R}_{\text{CRAC}_{\text{RSA}}}$. In the first round of the issuance protocol in Fig. 4, the client sends a NIZK proof of knowledge for an OR relation with two branches. Intuitively, the prover shows that the blinded message is well-formed and that the commitment hidden inside it opens to a vector of attributes; in addition, either the client knows a certificate that justifies those attributes under the policy and they satisfy the issuance predicate, or all attributes are equal to $\perp$.

We define the relation $\text{R}_{\text{CRAC}_{\text{RSA}}}$ used by the NIZK as follows. Let $\mathbf{x} := (\text{vk}, t, \text{pol}, \phi_{\text{Iss}})$ and $\mathbf{w} := (\text{cert}, \mathbf{attrs}, \mathbf{c}_{\text{vry}}, \mathbf{d}_{\text{vry}}, \kappa)$ with $\text{vk} := (N, e, \text{PSS})$, $\mathbf{c}_{\text{vry}} = (c'_i)_{i \in [\ell]}$, $\mathbf{d}_{\text{vry}} = (d'_i)_{i \in [\ell]}$ and $\mathbf{attrs} = (a_i)_{i \in [\ell]}$. Then $(\mathbf{x}, \mathbf{w}) \in \text{R}_{\text{CRAC}_{\text{RSA}}}$ if the two common conditions below hold and, in addition, at least one of the two branches is satisfied. Note that the commitment $\mathbf{c}_{\text{vry}}$ appears in the witness and not in the statement, so that it is never revealed to the issuer.

Common conditions. (1) $\text{Com}_{\text{vry}}.\text{Opn}(a_i, c'_i, d'_i) = 1$ for every $i \in [\ell]$; (2) letting $m := c'_1 \parallel \dots \parallel c'_\ell$ and $\mu := \text{PSS}.\text{Enc}(m)$, it holds that $\kappa \in \mathbb{Z}_N^*$ and $t = \mu \kappa^e \pmod{N}$.

Honest Branch (policy branch). $\text{P}(\text{pol}, \text{cert}, \mathbf{attrs}) = 1$ and $\phi_{\text{Iss}}(\mathbf{attrs}) = 1$.

Trapdoor Branch (zero-attribute branch). For every $i \in [\ell]$, $a_i = \perp$.

The relation outputs 1 if the common conditions hold and either the Honest Branch or the Trapdoor Branch is satisfied, and outputs 0 otherwise. Condition (1) ties the attributes to the commitment $\mathbf{c}_{\text{vry}}$, and condition (2) ties that commitment to the value t that the issuer signs; together with the Honest Branch, they are what forces the signature to be produced on a commitment to attributes that a certificate justifies under the policy. Condition (2) is well defined because, in our treatment of RSA-PSS, the encoding $\text{PSS}.\text{Enc}$ is deterministic, with the salt derived from the message (see App. D.2). The Trapdoor Branch lets a client obtain a credential on the all- $\perp$ vector without exhibiting any certificate. It serves two purposes. First, since the proof does not reveal which branch the client satisfied, the issuer cannot tell whether a session has produced a valid credential or a useless one on the all- $\perp$ vector. Second, it is what makes censorship-resistance provable under the restriction of Sec. 3.1 that the simulator has no control over the public setup or the random oracle: the simulator in the proof of Thm. 3 is the honest client itself, run on the all- $\perp$ inputs, i.e., a legitimate run of the protocol that involves no certificate. Note that $\perp$ is not admissible for any verification predicate, so such a credential is rejected by every honest verifier. Note also that the branch is all-or-nothing: a client that puts a genuine value in even one field must satisfy the Honest Branch, and hence must exhibit a certificate justifying *all* of its attributes.

Protocol $\text{CRAC}_{\text{RSA}}[\text{Com}_{\text{vry}}, \text{PS}, \text{R}_{\text{CRAC}_{\text{RSA}}}]$

- **Key Generation** $\text{CRAC}_{\text{RSA}}.\text{KG}(1^\lambda) \rightarrow (\text{vk}, \text{ik})$:
 1. Generate an RSA key pair $(N, e, d) \leftarrow \text{RSAGen}(1^\lambda)$.
 2. Generate the PSS parameters $\text{PSS} \leftarrow \text{PSSGen}(1^\lambda, N)$.
 3. Output (vk, ik) where $\text{ik} := (N, e, d, \text{PSS})$ and $\text{vk} := (N, e, \text{PSS})$.
- **Issuance** $\text{CRAC}_{\text{RSA}}.\text{Issue}(\mathcal{I}(\text{ik}, \text{pol}, \phi_{\text{Iss}}), \mathcal{C}(\text{vk}, \text{pol}, \phi_{\text{Iss}}, \text{cert}, \mathbf{attrs}))$: The construction is round-optimal: the client sends one message, the issuer replies (statelessly), and the client computes the credential.
 - **(1st round)** The algorithm $\text{CRAC}_{\text{RSA}}.\text{IClient}_0$ on input $(\text{vk}, \text{pol}, \phi_{\text{Iss}}, \text{cert}, \mathbf{attrs})$ outputs $(\text{pm}_c^0, \text{st}_c^0)$ and works as follows:

1. $\forall i \in [\ell]$, compute $(c'_i, d'_i) \leftarrow \text{Com}_{\text{vry}}.\text{Cmt}(a_i)$, and set $\mathbf{c}_{\text{vry}} := (c'_i)_{i \in [\ell]}$ and accordingly $\mathbf{d}_{\text{vry}} := (d'_i)_{i \in [\ell]}$ and $m := c'_1 \parallel \dots \parallel c'_\ell$.
2. Compute $\mu := \text{PSS}.\text{Enc}(m)$.
3. If $\gcd(\mu, N) \neq 1$, abort.
4. Sample $\kappa \leftarrow \mathbb{Z}_N^*$, then compute $t := \mu \kappa^e \pmod{N}$.
5. Compute $\pi \leftarrow \text{PS}.\text{Prv}(\mathbf{x}, \mathbf{w})$ for the relation $\text{R}_{\text{CRAC}_{\text{RSA}}}$, where $\mathbf{x} := (\text{vk}, t, \text{pol}, \phi_{\text{Iss}})$ and $\mathbf{w} := (\text{cert}, \mathbf{attrs}, \mathbf{c}_{\text{vry}}, \mathbf{d}_{\text{vry}}, \kappa)$.
6. Return $\text{pm}_{\mathcal{C}}^0 := (t, \pi)$ and store $\text{st}_{\mathcal{C}}^0 := (\text{vk}, \mu, \kappa, \mathbf{c}_{\text{vry}}, \mathbf{d}_{\text{vry}})$.
- **(2nd round)** The algorithm $\text{CRAC}_{\text{RSA}}.\text{Issuer}_0$ on input $(\text{ik}, \text{pol}, \phi_{\text{Iss}}, \text{pm}_{\mathcal{C}}^0)$ outputs $(\text{pm}_{\mathcal{T}}^0, \perp)$ and works as follows:
 1. If $\text{pm}_{\mathcal{C}}^0$ cannot be parsed as (t, π) , abort.
 2. Let $\text{vk} := (N, e, \text{PSS})$ and compute $\mathbf{x} := (\text{vk}, t, \text{pol}, \phi_{\text{Iss}})$. If $\text{PS}.\text{Vry}(\mathbf{x}, \pi) = 0$, abort.
 3. Otherwise, compute $\text{pm}_{\mathcal{T}}^0 := t^d \pmod{N}$.
- **(credential computation)** The algorithm $\text{CRAC}_{\text{RSA}}.\text{IClient}_1$ on input $(\text{st}_{\mathcal{C}}^0, \text{pm}_{\mathcal{T}}^0)$ outputs cred and works as follows:
 1. Compute $\sigma := \text{pm}_{\mathcal{T}}^0 \kappa^{-1} \pmod{N}$.
 2. If $\sigma^e \neq \mu \pmod{N}$, output $\text{cred} := \perp$.
 3. Otherwise, output $\text{cred} := (\sigma, \mathbf{c}_{\text{vry}}, \mathbf{d}_{\text{vry}})$.
- **Verification** $\text{CRAC}_{\text{RSA}}.\text{Vry}(\mathcal{V}(\text{vk}, \phi_{\text{Vry}}), \mathcal{C}(\text{vk}, \phi_{\text{Vry}}, \mathbf{attrs}, \text{cred}))$: The protocol consists of two messages: the verifier sends a context string, the client replies with the showing, and the verifier outputs a bit.
 - **(1st round)** The algorithm $\text{CRAC}_{\text{RSA}}.\text{Verifier}_0$ on input $(\text{vk}, \phi_{\text{Vry}})$ outputs $(\text{pm}_{\mathcal{V}}^0, \text{st}_{\mathcal{V}}^0)$, where $\text{pm}_{\mathcal{V}}^0 := \text{ctx}$ for an application-defined context string $\text{ctx} \in \{0, 1\}^*$, and stores $\text{st}_{\mathcal{V}}^0 := (\text{vk}, \phi_{\text{Vry}}, \text{ctx})$.
 - **(2nd round)** The algorithm $\text{CRAC}_{\text{RSA}}.\text{VClient}_0$ on input $(\text{vk}, \phi_{\text{Vry}}, \mathbf{attrs}, \text{cred}, \text{pm}_{\mathcal{V}}^0)$ outputs $\text{pm}_{\mathcal{C}}^0$ and works as follows:
 1. Interpret cred as $(\sigma, \mathbf{c}_{\text{vry}}, \mathbf{d}_{\text{vry}})$ with $\mathbf{d}_{\text{vry}} = (d'_i)_{i \in [\ell]}$, and $\text{pm}_{\mathcal{V}}^0$ as ctx .
 2. If $\phi_{\text{Vry}}(\mathbf{attrs}) = 0$, abort.
 3. Set $\mathbf{opn} := (a_i)_{i \in \text{flds}(\phi_{\text{Vry}})}$ and compute an application-defined auxiliary string $\mathbf{aux} \in \{0, 1\}^*$, possibly depending on ctx .
 4. Return $\text{pm}_{\mathcal{C}}^0 := (\mathbf{c}_{\text{vry}}, \sigma, \mathbf{opn}, (d'_i)_{i \in \text{flds}(\phi_{\text{Vry}})}, \mathbf{aux})$.
 - **(decision)** The algorithm $\text{CRAC}_{\text{RSA}}.\text{Verifier}_1$ on input $(\text{st}_{\mathcal{V}}^0, \text{pm}_{\mathcal{C}}^0)$ outputs a bit and works as follows:
 1. If $\text{pm}_{\mathcal{C}}^0$ cannot be parsed as $(\mathbf{c}_{\text{vry}}, \sigma, \mathbf{opn}, (d'_i)_{i \in \text{flds}(\phi_{\text{Vry}})}, \mathbf{aux})$, return 0.
 2. If $\text{flds}(\phi_{\text{Vry}}) = \emptyset$, return 0.
 3. If the application-defined check on $(\text{ctx}, \mathbf{aux})$ fails, return 0.^a
 4. Interpret $\mathbf{c}_{\text{vry}}$ as $(c'_i)_{i \in [\ell]}$ and set $m := c'_1 \parallel \dots \parallel c'_\ell$.
 5. If $\text{RSA}.\text{Vry}(\text{vk}, m, \sigma) = 0$, return 0.
 6. If there exists $i \in \text{flds}(\phi_{\text{Vry}})$ such that $\text{Com}_{\text{vry}}.\text{Opn}(\mathbf{opn}_i, c'_i, d'_i) = 0$, return 0.
 7. If there exists $i \in \text{flds}(\phi_{\text{Vry}})$ such that $\mathbf{opn}_i \notin \mathcal{S}_i^{\phi_{\text{Vry}}}$, return 0.
 8. Return 1.

^aWe leave the relation between ctx and $\mathbf{aux}$ unspecified, since it depends on the application: in the generic construction this check is vacuous, i.e., it always succeeds, and the two strings are otherwise ignored.

Fig. 4: Construction of $\text{CRAC}_{\text{RSA}}[\text{Com}_{\text{vry}}, \text{PS}, \text{R}_{\text{CRAC}_{\text{RSA}}}]$. The commitment $\mathbf{c}_{\text{vry}}$ is produced by the client at the beginning of the issuance protocol, is never seen by the issuer, and is sent to the verifier at showing time. The strings ctx and aux are placeholders: they are exchanged by the two parties, and the only check involving them is application-defined and vacuous in the generic construction (an example of a possible instantiation is in Sec. 5.2).

Description of the Construction. We next provide a verbal description of the two interactive protocols. We omit the key-generation procedure, since it coincides with the standard RSA-PSS key generation RSA.KG in Fig. 6.

Issuance protocol. Suppose a client holds a certificate cert that justifies, under a policy pol , a vector of attributes attrs satisfying an issuance predicate ϕ_{Iss} , and wishes to obtain a credential on attrs by interacting with an issuer holding RSA secret key $\text{ik} = (N, e, d, \text{PSS})$ and public key $\text{vk} = (N, e, \text{PSS})$. The protocol is round-optimal and consists of one client message, one issuer response, and a final client post-processing step.

In the first round, the client commits to each of its attributes with Com_{vry} , obtaining the commitment $\mathbf{c}_{\text{vry}}$ and its openings $\mathbf{d}_{\text{vry}}$, and it sets $m := c'_1 \parallel \dots \parallel c'_\ell$. It computes $\mu := \text{PSS.Enc}(m)$, aborts if μ is not coprime to N , as in the blinding step of RSA-BSSA [DJW23], samples a blinding factor $\kappa \leftarrow \mathbb{Z}_N^*$ and defines $t := \mu \kappa^e \pmod{N}$. The client proves in zero-knowledge that t is well-formed, that the commitments hidden inside m open to a vector of attributes attrs , and that *either* it knows a certificate that justifies those attributes under the policy, with the attributes satisfying the issuance predicate, *or* every attribute equals $\perp$, i.e., the Trapdoor Branch of $\text{R}_{\text{CRAC}_{\text{RSA}}}$. It sends (t, π) to the issuer and stores $(\text{vk}, \mu, \kappa, \mathbf{c}_{\text{vry}}, \mathbf{d}_{\text{vry}})$.

Upon receiving (t, π) , the issuer reconstructs $\mathbf{x} = (\text{vk}, t, \text{pol}, \phi_{\text{Iss}})$ and verifies the proof. If verification fails, it aborts. Otherwise, it computes the RSA response $t^d \pmod{N}$ and returns it to the client. Observe that the issuer sees neither $\mathbf{c}_{\text{vry}}$ nor m : the former appears only in the witness, and the latter only inside the blinded value t . This is what allows a showing to be performed against $\mathbf{c}_{\text{vry}}$ in the clear without linking it to the issuance session.

Finally, the client unblinds the issuer's response. Upon receiving pm_T^0 , it computes $\sigma := \text{pm}_T^0 \kappa^{-1} \pmod{N}$ and checks that $\sigma^e \equiv \mu \pmod{N}$, i.e., that σ is an RSA inverse of the very encoding μ it blinded; if the check fails it outputs $\perp$, otherwise it outputs the credential $\text{cred} := (\sigma, \mathbf{c}_{\text{vry}}, \mathbf{d}_{\text{vry}})$. When the issuer answers honestly with $\text{pm}_T^0 = t^d \pmod{N}$, the check always passes, since $\sigma^e = t^{de} \kappa^{-e} = t \kappa^{-e} = \mu \pmod{N}$, and the value σ is an RSA-PSS signature on m that passes standard RSA-PSS verification. The check is the same performed by the finalization step of blind RSA-PSS [Lys23, BR98], and it is necessary: a malicious issuer that replies with an arbitrary value in one session would otherwise obtain a credential whose signature is correlated with that session, which is what allows the credential to be linked to the session at showing time (see the proof of Thm. 2).

Verification protocol. The verifier holds vk and the verification predicate ϕ_{vry} , and the client additionally holds attrs and cred . The verifier speaks first, sending a context string ctx whose content is left to the application; in the EUDIW setting it may carry, e.g., a challenge for the device-binding compliance (see Sec. 5.2 and App. A).

The client checks that its attributes satisfy ϕ_{vry} and, if so, sends the showing commitment together with the signature, the values of the disclosed fields, their openings and an auxiliary string, that is $(\mathbf{c}_{\text{vry}}, \sigma, \text{opn}, (\mathbf{d}'_i)_{i \in \text{flds}(\phi_{\text{vry}})}, \text{aux})$ with $\text{opn} = (\mathbf{a}_i)_{i \in \text{flds}(\phi_{\text{vry}})}$. The entire commitment vector

$\mathbf{c}_{\text{vry}}$ is sent, since the verifier needs all of its components to recompute the signed message, whereas openings are sent only for the disclosed fields.¹¹

The verifier then proceeds as follows. It enforces non-trivial disclosure, returning 0 if $\text{flds}(\phi_{\text{vry}})$ is empty. It recomputes m from $\mathbf{c}_{\text{vry}}$ and checks the RSA-PSS signature by running $\text{RSA.Vry}(\text{vk}, m, \sigma)$; the signature is therefore what certifies that the commitment vector received from the client was signed in some issuance session, i.e., one in which the issuer verified the proof for $\text{R}_{\text{CRAC}_{\text{RSA}}}$, without identifying that session. It checks that each revealed value is a valid opening of the corresponding component of $\mathbf{c}_{\text{vry}}$, and finally that each revealed value is admissible for the predicate, i.e., that $\text{opn}_i \in \mathcal{S}_i^{\phi_{\text{vry}}}$ for every $i \in \text{flds}(\phi_{\text{vry}})$. If all checks pass it returns 1.

4.1 Security Analysis

We give a concrete security analysis of the scheme in Fig. 4. In particular, we show correctness, credential-unlinkability, censorship-resistance, and credential-knowledge according to the definitions given in Sec. 3.1.

Correctness. Correctness follows immediately from the perfect correctness of the scheme PS for the relation $\text{R}_{\text{CRAC}_{\text{RSA}}}$, of the commitment scheme Com_{vry} and of the signature scheme RSA. We state the formal theorem below and give the full proof in App. E.1.

Theorem 1 (Correctness). *Let PS be perfectly correct for the relation $\text{R}_{\text{CRAC}_{\text{RSA}}}$, let Com_{vry} be a perfectly correct commitment scheme, and let RSA be the perfectly correct RSA-PSS signature scheme of Fig. 6. Then the CRAC system $\text{CRAC}_{\text{RSA}}[\text{Com}_{\text{vry}}, \text{PS}, \text{R}_{\text{CRAC}_{\text{RSA}}}]$ of Fig. 4 is perfectly correct.*

Credential-Unlinkability. The proof is a sequence of three games G_0, G_1, G_2 . The first game G_0 is the real one, and the final step shows that the view of $\mathcal{A}$ in the last game is independent of b , so that $\Pr[G_2 = 1] = 1/2$.

The proof mainly follows this idea. The only way a credential can be linked to its issuing session is by linking the values the client sent during issuance, namely the blinded encoding t and the proof π , together with the issuer's response pm_T^0 , to the values that the client later shows, namely $\mathbf{c}_{\text{vry}}$, σ and the openings of the disclosed fields. The games remove the dependencies on t , π and $\mathbf{c}_{\text{vry}}$ one at a time. The game G_1 replaces the proofs of both sessions with simulated ones, so that π no longer depends on the witness and hence on $\mathbf{c}_{\text{vry}}$; otherwise there is a direct reduction to the zero-knowledge of PS. The game G_2 replaces the components of $\mathbf{c}_{\text{vry}}$ on which the two attribute vectors differ with commitments to a fixed value, so that the two showing commitments become identically distributed; otherwise there is a direct reduction to the hiding of Com_{vry} , and it is possible because the showing oracle never reveals the openings of those components.

The final step is perfect and needs no computational assumption: it follows the argument of [Lys23, Claim 3] for the blindness of plain blind RSA. For a malicious key (N, e) , the blinded value $\mu\kappa^e$ is uniform among the e -th residues and independent of μ , and the unblinded signature $\text{pm}_T^0\kappa^{-1}$ is uniform among the e -th roots of μ given the transcript, by [Lys23, Lemmas 5–7]. The check $\sigma^e \equiv \mu$ performed by the client guarantees that a session yields a credential only if the issuer's response is a valid root of t , so that a malicious response cannot correlate the credential with the session. In G_2 the issuance transcript of each session is then independent of the tuple $(\mathbf{c}_{\text{vry}}, \mathbf{d}_{\text{vry}}, \sigma)$ produced in that session, and these tuples are i.i.d. across the two sessions, so the showing carries no information about b .

¹¹The strings ctx and aux carry no meaning in the generic construction: the check the verifier performs on them is vacuous, and the client may set aux to the empty string. We keep them in the syntax of the protocol because they are used by the instantiation of Sec. 5.2, where aux is computed as a function of ctx .

Note that no assumption on the honest generation of (N, e) is needed, in line with the malicious-issuer model of Fig. 1. That is, the case in which the value μ computed in a session is not an e -th residue, which may happen when e is not coprime to $\varphi(N)$, only causes that session to output $\perp$, after which the showing aborts and the view is trivially independent of b . The same holds when μ is not coprime to N , in which case the client aborts before sending t .

We state the formal theorem below and give the full proof in App. E.2.

Theorem 2 (Credential-Unlinkability). *Let PS be a NIZK proof system for the relation $\text{R}_{\text{CRAC}_{\text{RSA}}}$ satisfying zero-knowledge, and let Com_{vry} be a computationally hiding commitment scheme. Then the CRAC system $\text{CRAC}_{\text{RSA}}[\text{Com}_{\text{vry}}, \text{PS}, \text{R}_{\text{CRAC}_{\text{RSA}}}]$ of Fig. 4 satisfies computational (single-show) credential-unlinkability. More precisely, for every PPT adversary $\mathcal{A}$ there exist a PPT adversary $\mathcal{A}_{\text{ZK}}$ against the zero-knowledge of PS and a PPT adversary $\mathcal{A}_{\text{HDN}}$ against the hiding of Com_{vry} , such that:*

$$\text{Adv}_{\text{CRAC}_{\text{RSA}}}^{\text{ulk}}(\mathcal{A}, \lambda) \leq 2 \text{Adv}_{\text{PS}}^{\text{zk}}(\mathcal{A}_{\text{ZK}}, \lambda) + 2 \text{Adv}_{\text{Com}_{\text{vry}}}^{\text{hdn}}(\mathcal{A}_{\text{HDN}}, \lambda).$$

Censorship-Resistance. In CR the adversary sees, in each session, only the inputs $(\text{pol}, \phi_{\text{Iss}})$ and the single message (t, π) that the honest client sends during the issuance protocol: the output of the client is discarded, and no showing ever takes place. The role of the issuance oracle in Fig. 2 is precisely to produce these messages, which D observes through $\mathcal{A}$.

The simulator we exhibit is the honest client itself, run on dummy inputs. It sets $\text{cert} := \perp$ and $\text{attrs} := (\perp)_{i \in [\ell]}$ and executes $\text{CRAC}_{\text{RSA}}.\text{IClient}_0$: with these inputs the client satisfies the Trapdoor Branch of $\text{R}_{\text{CRAC}_{\text{RSA}}}$, and hence completes the protocol without a meaningfully valid certificate. The issuance predicate does not constrain it, since ϕ_{Iss} is checked only by the Honest Branch, and indeed no predicate admits $\perp$. Hence Sim is a legitimate client that anybody can run without a certificate, and the theorem states that an issuer cannot tell such a client from one holding a certificate. The sequence of games then moves from the real game, in which the client proves the Honest Branch, to this simulator: G_1 simulates π , G_2 replaces the commitments with commitments to $\perp$, and G_3 produces π from $\text{cert} = \perp$ and the all- $\perp$ attribute vector, which is the simulator.

The message hidden by the simulator differs from that of a client in two places: the commitment $\mathbf{c}_{\text{vry}}$ hidden inside t , which commits to $\perp$ instead of attrs ; and the witness used to compute π . The games change them one at a time.

Finally, we need to state two remarks. First, no game is devoted to t . The blinded encoding is a public function of $(\mathbf{c}_{\text{vry}}, \kappa)$, the attributes enter it only through the components of $\mathbf{c}_{\text{vry}}$, and the reduction of G_2 recomputes t , together with the condition $\text{gcd}(\mu, N) = 1$ under which the client sends it, from the commitments it receives from its oracle. In particular, censorship-resistance does not rely on the blinding factor κ , whose role is unlinkability (Thm. 2). Second, as in the malicious-issuer model of Fig. 2, nothing is assumed on (N, e) : if μ is not an e -th residue the honest client outputs $\perp$, but the output of the client is discarded by the game, so this case needs no analysis.

We state the formal theorem below and give the full proof in App. E.3.

Theorem 3 (Censorship-Resistance). *Let PS be a NIZK proof system for the relation $\text{R}_{\text{CRAC}_{\text{RSA}}}$ satisfying zero-knowledge with a universal simulator PS.Sim , and let Com_{vry} be a computationally hiding commitment scheme. Then the CRAC system $\text{CRAC}_{\text{RSA}}[\text{Com}_{\text{vry}}, \text{PS}, \text{R}_{\text{CRAC}_{\text{RSA}}}]$ of Fig. 4 satisfies computational censorship-resistance. More precisely, for every PPT adversary $\mathcal{A}$ there exists a PPT simulator Sim such that for every PPT distinguisher D , there exist PPT adversaries $\mathcal{A}_{\text{ZK},1}, \mathcal{A}_{\text{ZK},2}$ against the zero-knowledge of PS and a PPT adversary $\mathcal{A}_{\text{HDN}}$ against the hiding of Com_{vry} , such that:*

$$\text{Adv}_{\text{CRAC}_{\text{RSA}}}^{\text{cr}}(\mathcal{A}, \text{D}, \lambda) \leq 2 \text{Adv}_{\text{PS}}^{\text{zk}}(\mathcal{A}_{\text{ZK},1}, \lambda) + 2 \text{Adv}_{\text{Com}_{\text{vry}}}^{\text{hdn}}(\mathcal{A}_{\text{HDN}}, \lambda) + 2 \text{Adv}_{\text{PS}}^{\text{zk}}(\mathcal{A}_{\text{ZK},2}, \lambda).$$

Credential-Knowledge. In KNW the issuer and the verifier are honest and the adversary plays the client against both. It wins in one of two ways: at issuance, by completing a session in which the certificate and the attributes it used, as recovered by the extractor, are not legitimate for the policy and the issuance predicate of that session (in the issuance oracle of Fig. 3); at showing, by making the verifier accept values that neither agree with an attribute vector recorded in $\mathcal{Q}$ nor are among the values that a single honest client revealed to it (in the final step of Fig. 3).

Our strategy separates the two. The first is handled by the knowledge-soundness of PS: the extractor Ext we exhibit runs the extractor PS.Ext of PS on the proof π of every completed issuance session, and it retrieves the witness that contains the certificate, the attributes, the openings of $\mathbf{c}_{\text{Vry}}$ and the value κ that blinds the encoding; since the witness satisfies $\text{R}_{\text{CRAC}_{\text{RSA}}}$, either the Honest Branch holds, and the session is legitimate, or every attribute equals $\perp$, and the session is not counted.

The second is handled in two steps. The issuer’s response in a completed session is $t^d = \text{PSS.Enc}(m)^d \kappa$, i.e., the blinding of an RSA-PSS signature on the message m formed by the showing commitment $\mathbf{c}_{\text{Vry}}$ of the extracted witness: hence the messages on which the adversary holds a signature are known to the game and a showing on any other commitment vector yields a forgery against RSA. A showing on one of these vectors, finally, opens its components to the extracted attributes, or else breaks the binding of Com_{Vry} .

Concretely, the proof is a sequence of five games $G_0, \dots, G_4$. We bound the advantage of $\mathcal{A}$ in KNW by the advantages of reductions against the building blocks of the construction, namely the knowledge-soundness of PS, the unforgeability of RSA, and the binding and hiding of Com_{Vry} . The games proceed as follows. The game G_0 is the real game KNW of Fig. 3 run against CRAC_{RSA} , with the extractor Ext described above, i.e., the one that runs PS.Ext on the proof π of every completed issuance session; its output is the bit win , so that the probability that G_0 outputs 1 is exactly the advantage of $\mathcal{A}$. The game G_1 is identical to G_0 , except for its output: it runs the extractor of PS on the proof π of every completed issuance session and, if it finds a session whose extracted value is not a witness for its statement, it outputs 0 instead of computing win . The two games can differ only when such a session exists, and there is a direct reduction to the knowledge-soundness of PS. From this point on every completed session carries a witness satisfying $\text{R}_{\text{CRAC}_{\text{RSA}}}$, so the issuance oracle never sets win , the set $\mathcal{Q}$ consists of the extracted attributes, the game knows the openings $\mathbf{d}_{\text{Vry}}$ of every $\mathbf{c}_{\text{Vry}}$ on which it has issued a signature, and it knows the set of the messages it has signed, i.e., those of the completed sessions of $\mathcal{A}$ and those of the honest clients.

The game G_2 differs from G_1 only for its output: once $\mathcal{A}$ has halted, if the game finds in $\mathcal{W}$ a tuple accepted by the verifier whose commitment vector was never signed, it outputs 0 instead of computing win . The two games can differ only when such a tuple exists, and there is a direct reduction to the unforgeability of RSA, in which the reduction answers the issuance sessions of $\mathcal{A}$ and the honest issuances through its signing oracle, blinding the signature with the extracted κ in the former, and outputs the signature of the accepted showing as a forgery. This is where a showing built on a commitment vector that was never signed, i.e., on attributes for which no policy was ever checked at issuance, is excluded.

The game G_3 differs from G_2 only for its output: once $\mathcal{A}$ has halted, if the game finds in $\mathcal{W}$ a tuple accepted by the verifier that is built on the commitment vector of an issuing session of $\mathcal{A}$ and discloses some value different from the attribute extracted from that session, it outputs 0 instead of computing win . The two games can differ only when such a tuple exists, and there is a direct reduction to the binding of Com_{Vry} , since the reduction holds the opening extracted from the witness and the opening sent to the verifier, and they open the same commitment to different values. After G_3 , every accepted showing on a commitment vector of $\mathcal{A}$ discloses exactly the extracted attributes,

which are in $\mathcal{Q}$, and does not set `win`; it remains a showing built on the commitment vector of an honest client that reveals, in a field, a value that this client never revealed to $\mathcal{A}$.

The game G_4 is identical to G_3 , except for its output: once $\mathcal{A}$ has halted, if the game finds in $\mathcal{W}$ a tuple accepted by the verifier that is built on the commitment vector of an honest client and reveals, in some field, a value such that the pair of that field and value is not recorded in the set $\mathcal{R}_{\text{cid}}$ of that client, it outputs 0 instead of computing `win`. Two cases arise. Either the revealed value differs from the attribute of the honest client in that field, and there is a direct reduction to the binding of Com_{vry} , as in G_3 , with the honest opening in place of the extracted one; or it is equal to it, and $\mathcal{A}$ has opened a commitment of the honest client to the committed value without ever having seen its opening, since the field was never revealed to it. In the latter case there is a reduction to the hiding of Com_{vry} : the reduction plants a challenge commitment, to either that attribute or $\perp$, in the credential of the honest client, and an opening to the attribute tells the two apart, unless the commitment was to $\perp$, in which case that opening is a binding break. After G_4 , every accepted showing either discloses the extracted attributes of a credential of $\mathcal{A}$, which are in $\mathcal{Q}$, or repeats values that a single honest client revealed to $\mathcal{A}$, which are in $\mathcal{R}_{\text{cid}}$; hence `win` is never set.

We state the formal theorem below and give the full proof in App. E.4.

Theorem 4 (Credential-Knowledge). *Let PS be a NIZK proof system for the relation $\mathcal{R}_{\text{CRAC}_{\text{RSA}}}$ satisfying knowledge-soundness (Def. 12), let RSA be unforgeable (Asm. 1), and let Com_{vry} be a computationally binding and hiding commitment scheme. Then the CRAC system $\text{CRAC}_{\text{RSA}}[\text{Com}_{\text{vry}}, \text{PS}, \mathcal{R}_{\text{CRAC}_{\text{RSA}}}]$ of Fig. 4 for a number of attributes ℓ satisfies computational credential-knowledge. More precisely, for every PPT adversary $\mathcal{A}$ that opens at most q_I issuance sessions and creates at most q_C honest clients, there exist a PPT extractor Ext and PPT adversaries $\mathcal{A}_{\text{KS},j}$ for $j \in [q_I]$, $\mathcal{A}_{\text{EUF-CMA}}$, $\mathcal{A}_{\text{BND},j}$ for $j \in [3]$ and $\mathcal{A}_{\text{HDN}}$, against the knowledge-soundness of PS, the unforgeability of RSA, and the binding and the hiding of Com_{vry} respectively, such that:*

$$\begin{aligned} \text{Adv}_{\text{CRAC}_{\text{RSA}}}^{\text{knw}}(\mathcal{A}, \text{Ext}, \lambda) &\leq \sum_{j=1}^{q_I} \text{Adv}_{\text{PS}}^{\text{ks}}(\mathcal{A}_{\text{KS},j}, \lambda) + \text{Adv}_{\text{RSA}}^{\text{euf-cma}}(\mathcal{A}_{\text{EUF-CMA}}, \lambda) \\ &\quad + \sum_{j=1}^2 \text{Adv}_{\text{Com}_{\text{vry}}}^{\text{bnd}}(\mathcal{A}_{\text{BND},j}, \lambda) + q_C \cdot \ell \left(2 \text{Adv}_{\text{Com}_{\text{vry}}}^{\text{hdn}}(\mathcal{A}_{\text{HDN}}, \lambda) + \text{Adv}_{\text{Com}_{\text{vry}}}^{\text{bnd}}(\mathcal{A}_{\text{BND},3}, \lambda) \right). \end{aligned}$$

Everlasting privacy against malicious issuers. The theorems above also implicitly show how to strengthen CRAC_{RSA} against long-term privacy attacks on an honest client. In particular, the only computational steps in the proofs of credential-unlinkability and censorship-resistance rely on the zero-knowledge of PS and the hiding of Com_{vry} ; all remaining steps are information-theoretic. Thus, by instantiating Com_{vry} with a statistically hiding commitment scheme and PS with a statistically zero-knowledge proof system, the same arguments yield statistical credential-unlinkability and statistical censorship-resistance. This provides an everlasting privacy guarantee against a malicious issuer. In particular, an issuer cannot store transcripts today and later use them to mount a “harvest now, link later” attack. Both requirements admit efficient instantiations. For example, Com_{vry} can be instantiated with Pedersen commitments, which are perfectly hiding. For PS, statistical zero-knowledge can be obtained from Ligerio [AHIV17, AHIV23] together with the BCS transformation [BCS16], yielding a non-interactive statistically zero-knowledge argument in the random-oracle model.

4.2 On the Choice of RSA-PSS vs ECDSA

Our construction uses RSA-PSS, but nothing in Sec. 3.1 depends on this choice. A similar construction, using the same “blind signature” paradigm, can be carried out over ECDSA. Maire and Pulval-Dady [MPD25] give the first blind signing protocol producing standard ECDSA signatures whose security relies only on the ECDSA assumption,¹² and their protocol can play the role of blind RSA-PSS in Fig. 4. That is, the client would commit to its attributes as before and prove the policy over the message whose blind signature it requests. The composition is in fact natural, because their one-more unforgeability proof already extracts the message from the client’s proofs by *straight-line* extraction, which is exactly the property that our Def. 12 requires and on which the proof of Thm. 4 depends. That is, censorship-resistance would still follow from the Trapdoor Branch of the relation, whose simulator is the honest client run on the all- $\perp$ inputs, and credential-knowledge would still combine the knowledge-soundness of the client’s proof with the unforgeability of the blind signature.

We nevertheless do not develop this variant formally, for reasons that are practical rather than definitional. Their protocol runs in two rounds, i.e., four messages instead of two, and the signer must maintain a state between its two messages, including its share of the ECDSA nonce; an issuer serving many concurrent clients would therefore have to maintain per-session state, which is precisely what we avoid. Moreover, the proving effort is not confined to the client, and is already larger than in the RSA variant developed here. Indeed, besides the three proofs that the client produces with a generic NIZK, the signer itself must compute a NIZK attesting that it has evaluated the signing equation honestly, so that the issuer pays a proving cost at every session rather than a single proof verification. Finally, the overall protocol is considerably heavier. Namely, it relies on two linearly homomorphic encryption schemes in different moduli, one for scalars and one for elliptic curve points, and on commitments in both groups, apart from the several NIZKs mentioned above.

We stress that this is a statement about efficiency and deployability, not about security. An ECDSA instantiation is entirely possible following [MPD25], and would be equally sound; the choice of RSA-PSS is what makes our issuance protocol round-optimal, its issuer stateless, and its client’s work a single proof.

5 Certificates, Policies and EUDI-Compatible Credentials

The scheme of Fig. 4 leaves three choices to a deployment: the policy compiler P , the commitment scheme Com_{vry} , and the strings ctx and aux . The two subsections below discuss how to fix them. We first describe, informally, where certificates come from and what a policy has to be able to check; we then explain CRACS, namely the instantiation whose showings are compatible with the EUDIW.

5.1 On Certificates and Policies

Nothing in our definition produces a certificate. A certificate is external to the scheme. That is, it is whatever evidence an existing authority makes available to a client, and the policy compiler is simply the interface through which an issuer declares which pieces of evidence it is willing to accept.

The simplest case is a document signed by an authority, as in the running example of Sec. 3.1. That is, a passport carries a name, a date of birth and a nationality, signed by a national authority whose verification key is public. Here Pol verifies that signature and checks whatever the deployment requires of the document, such as that it has not expired, and AttrChk checks that the attributes the client asks to have certified are the ones the document carries. This is also the setting considered

¹²Such an assumption is of the same nature as the one we make on the security of RSA-PSS when the hash function is fixed in the plain model, and as those found in the literature on blind Schnorr signatures.

by [RWGM23], where the evidence is a digitally signed document, and the client proves that it holds one.

Evidence need not come in so convenient a form. A client may wish to be certified on the basis of data that is not signed for this purpose at all, but merely delivered to it over an authenticated channel, such as the contents of a page on a web server that the client is logged into. Proving to a third party what a server said during a TLS session is the subject of a line of work on TLS oracles and on zero-knowledge proofs over TLS-protected data [ZMM⁺20, XYWY24, CDH⁺25, DVVZ26], and any such mechanism can play the role of a certificate here: the policy circuit verifies the transcript-level evidence instead of a document signature, and the attributes are the values the client extracted from the session.

5.2 CRACS: CRAC_{RSA} for EUDI-Compatible Credentials

We now fix the three choices left open by Fig. 4, namely the commitment scheme, the strings `ctx` and `aux` and the policy compiler, so as to obtain CRACS¹³, a scheme that meets the requirements that the ARF [EUD] imposes on the presentation of a credential, as summarized in App. A. Two requirements drive the instantiation: the encoding of the attributes, which fixes Com_{vry} , and *device binding*, which fixes the remaining degrees of freedom.

Device binding splits into two requirements, which we treat separately and refer to as *key binding* and *proof of possession*: key binding is enforced at issuance and ties the credential to a key that resides in the holder’s secure device; proof of possession is enforced at presentation and establishes that whoever is showing the credential has that device at that moment. The first is a property of what is certified, the second of what is exchanged during a showing, and the two are enforced by different mechanisms. As we show at the end of the section, proof of possession is also what makes a single-show notion mandatory in this setting.

Instantiating the showing commitment. A credential conforming to the ARF encapsulates its attributes according to SD-JWT VC [FYC] or to the ISO/IEC 18013-5 mdoc format [Int21], and in both the attributes appear as salted hashes, with the salts of the disclosed fields revealed at presentation time. We are therefore forced to instantiate Com_{vry} with the scheme Com_{H} of App. D.3, whose security is stated in Asm. 2. With this choice the showing message of Fig. 4 consists of the vector of salted hashes, the values of the fields in $\text{flds}(\phi_{\text{vry}})$ together with their salts, a signature σ that the verifier checks with standard RSA-PSS verification over $m = \text{c}'_1 \parallel \dots \parallel \text{c}'_\ell$, and the string `aux` fixed below. This matches the structure of a presentation in the two ARF formats, and the verifier requires no operations beyond a hash function and signature verification.

The credential of the ARF and the string cred. The standards call *credential* the object that the verifier receives, whereas *cred* is the private output of issuance (see the remark following the syntax in Sec. 3.1); we make the correspondence explicit. In CRACS, $\text{cred} = (\sigma, \text{c}_{\text{vry}}, \text{d}_{\text{vry}})$, where $\text{d}_{\text{vry}} = (\text{d}'_i)_{i \in [\ell]}$ collects the ℓ salts. The pair $(\sigma, \text{c}_{\text{vry}})$, i.e., the list of salted hashes with the issuer’s RSA-PSS signature on it, is the part of the credential signed by the issuer, namely the JWT of SD-JWT VC and the Mobile Security Object of mdoc [FYC, Int21]: it contains no secret, and it is sent to the verifier, in full, at every presentation. The salts are the secret half of the *disclosures* of SD-JWT VC and of the issuer-signed items of mdoc, each of which pairs a salt with a field and the attribute stored in it. The wallet stores all of them alongside the credential, exactly as an EUDI

¹³CRACS stands for CRAC Standardized. That is, the CRAC of Sec. 3.1, realized by the construction of Sec. 4 and instantiated in this section with a standardized commitment scheme, a standardized signature scheme, and device binding via widely deployed secure elements, so that the resulting credentials and their presentation are fully compliant with the EUDIW.

wallet does, and sends only those of the fields in $\text{flds}(\phi_{\text{Vry}})$, each with the value **opn**_{*i*} it opens; the others never leave the wallet, which is what keeps the undisclosed attributes hidden.

Device binding at issuance: key binding and the key attribute. The ARF mandates that a credential be cryptographically bound to a key that resides in the holder’s Wallet Secure Cryptographic Device (WSCD), so the credential cannot be transferred to another holder. We realize key binding by reserving the last field of the credential for the device key: we require $\mathbf{a}_\ell = \mathbf{pk}_{\text{dev}} \in \mathcal{K}$, where $\mathcal{K}$ is the set of well-formed public keys of a signature scheme $\text{DS} = (\text{DS.KG}, \text{DS.Sig}, \text{DS.Vry})$ supported by the WSCD, and we set $\mathcal{S}_\ell^{\phi_{\text{iss}}} = \mathcal{K}$ accordingly. As discussed in App. A, secure elements support ECDSA and RSA [Fs26], so DS is one of the two; both are admitted by the JOSE and COSE encapsulations.

It remains to guarantee that the key in this field really is a WSCD key, and not one the client generated in software. In the ARF this guarantee is provided by a *Key Attestation* (KA), issued by the Wallet Provider when the Wallet Unit is activated [EUD, Section 6.5.3.4]. A KA is a signed object that describes the certification and the properties of the WSCD, contains one or more public keys, and attests that the corresponding private keys were generated inside that WSCD and cannot be extracted from it. The Wallet Provider is publicly disclosed. That is, its trust anchors are published in the Wallet Provider LoTE [EUD, Section 6.2.2], so the key under which a KA is verified is public information.

We therefore fix the policy as follows. A certificate is a tuple $\text{cert} = (\mathbf{ka}, \mathbf{dom})$, where $\mathbf{ka} \in \{0, 1\}^*$ is a Key Attestation and $\mathbf{dom} \in \{0, 1\}^*$ collects the domain-specific material the policy is about, that is, the evidence for the attributes in the fields $1, \dots, \ell - 1$, of the kind discussed in Sec. 5.1. The two circuits of Sec. 4 are then instantiated so that each of them performs a check on each component:

- (i) $\text{Pol}(\text{cert})$ verifies the signature on $\mathbf{ka}$ under a Wallet Provider trust anchor taken from the LoTE, checks that the WSCD it describes meets the required certification level, and checks its revocation reference¹⁴; and it runs the domain-specific validity check on $\mathbf{dom}$, whatever the application requires of the evidence backing the attributes.
- (ii) $\text{AttrChk}(\text{cert}, \mathbf{attrs})$ checks that $\mathbf{a}_\ell$ is one of the public keys contained in $\mathbf{ka}$; and it checks that $\mathbf{a}_1, \dots, \mathbf{a}_{\ell-1}$ are justified by $\mathbf{dom}$, again as the application requires.

Only the first half of each check is fixed by the ARF, and it is the same for every application: it is what realizes key binding. The second half is what the policy is *about*, and it varies from one deployment to another.

The point to stress is *where* the Key Attestation goes. In the ARF issuance flow the Wallet Unit sends the KA to the issuer [EUD, Section 6.6.2.4], which is precisely what enables the issuer to discriminate among clients. In our issuance protocol, the KA is part of the witness of the NIZK. This is the sense in which our scheme departs from the ARF at issuance in order to comply with it at presentation, and it is where censorship-resistance comes from.

Device binding at showing: proof of possession. Key binding, on its own, guarantees only that the credential was issued against a key held in a secure device; it says nothing about who is presenting it. A client that obtains a credential legitimately may hand it to someone else, who would then be able to show it. Proof of possession closes this gap by requiring, at presentation time, evidence that the private key matching the certified public key is available in that very session. Following the ARF presentation flow, the verifier draws a fresh challenge and requires a signature

¹⁴If the two components are checked separately, two colluding clients may obtain a credential that carries the evidence of one and is bound to the WSCD of the other. This is not specific to CRACS but a matter of the issuer’s policy, which can rule it out by requiring $\mathbf{dom}$ to be itself device-bound to a key attested in $\mathbf{ka}$; we do not pursue it further.

on it under the device key. We instantiate the two placeholders of Fig. 4 accordingly:

$$\text{ctx} \leftarrow \$ \{0, 1\}^\lambda, \quad \text{aux} \leftarrow \text{DS.Sig}(\text{sk}_{\text{dev}}, \text{ctx} \parallel \text{c}_{\text{vry}} \parallel \text{opn}),$$

and the application-defined check on (ctx, aux) of Fig. 4 is instantiated as follows: $\text{CRAC}_{\text{RSA}}.\text{Verifier}_1$ returns 0 if $\ell \notin \text{flds}(\phi_{\text{vry}})$, and it returns 0 if $\text{DS.Vry}(\text{opn}_\ell, \text{ctx} \parallel \text{c}_{\text{vry}} \parallel \text{opn}, \text{aux}) = 0$, where $\text{opn}_\ell = \text{pk}_{\text{dev}}$ and $(\text{pk}_{\text{dev}}, \text{sk}_{\text{dev}})$ have been generated inside the WSCD according to DS.KG . The first condition restricts the admissible verification predicates to those that disclose the key field, and is what makes the second one meaningful, since the key under which the response is verified is the one opened from the credential; a showing that does not disclose the key field carries no device binding at all, and is rejected.

Note that disclosing pk_{dev} at every showing does not weaken credential-unlinkability. The issuer never learns the device key: at issuance, pk_{dev} enters the protocol only inside the commitment c'_ℓ , and it is tied to the Key Attestation only within the witness of the NIZK. The key a verifier sees therefore cannot be matched to any issuance session. What the disclosed key does affect is unlinkability across showings of the same credential, which we discuss at the end of the section.

It is worth being precise about *what* is signed, since signing the challenge alone would be insufficient. Suppose aux were $\text{DS.Sig}(\text{sk}_{\text{dev}}, \text{ctx})$. A malicious client in possession of a credential it is not entitled to show, but not of the corresponding WSCD, could then prepare the showing itself, obtain from the legitimate holder of the device a signature on the string ctx alone, and attach it to its own showing. The holder of the device would have no way to tell what it was authorizing, since ctx is a uniformly random string carrying no information about the credential, the disclosed fields, or even the identity of the verifier. Signing the showing message together with the challenge removes this: the device signs a specific disclosure of a specific credential in a specific session, so a signature produced for one showing is useless for any other.

Note that the verifier now performs two signature verifications, one with RSA-PSS under the issuer key and one with DS under the device key, and both are among the algorithms recommended by the ECCG [Eur25, Section 5.2] and admitted by JOSE and COSE; compliance is therefore preserved.

Relying Party authentication. In the model of Sec. 3.1 the verifier holds no key, and a showing is checked under vk alone. The EUDIW instead equips every verifier, i.e., every Relying Party Instance, with a key pair $(\text{pk}_\mathcal{V}, \text{sk}_\mathcal{V})$ and an *access certificate* on $\text{pk}_\mathcal{V}$, issued upon registration by an Access Certificate Authority whose trust anchors are published [EUD, Section 6.4.2]. The certificate serves verifier authentication: the verifier includes it in the presentation request and signs the request with $\text{sk}_\mathcal{V}$, and the wallet verifies the signature and the certificate chain before disclosing anything, stopping the transaction otherwise, since a forged request may point to a man-in-the-middle [EUD, Section 6.6.3.2]. Both presentation protocols of the ARF, ISO/IEC 18013-5 in proximity and OpenID4VP remotely [EUD, Sections 5.7.2 and 5.7.4], implement it, and integrating it into CRACS requires no change to the scheme, since it fits in the placeholder ctx fixed above. Precisely, the first message of the verifier becomes $\text{ctx} := (\text{nonce}, \text{req}, \text{cert}_\mathcal{V}, \sigma_\mathcal{V})$, where $\text{nonce} \leftarrow \$ \{0, 1\}^\lambda$ plays the role of the challenge drawn above, req is the presentation request, which carries ϕ_{vry} , $\text{cert}_\mathcal{V}$ is the access certificate chain of the verifier and $\sigma_\mathcal{V} \leftarrow \text{Sig.Sig}(\text{sk}_\mathcal{V}, \text{nonce} \parallel \text{req})$. Two tweaks follow. First, $\text{CRAC}_{\text{RSA}}.\text{Verifier}_0$ takes $(\text{sk}_\mathcal{V}, \text{cert}_\mathcal{V})$ as an additional private input, which the syntax of Sec. 3.1 does not foresee; it is used only to authenticate the verifier, and the decision of $\text{CRAC}_{\text{RSA}}.\text{Verifier}_1$ remains a function of vk and of the transcript alone, so that showings stay publicly verifiable. Second, $\text{CRAC}_{\text{RSA}}.\text{VClient}_0$ verifies $\sigma_\mathcal{V}$ under the key in $\text{cert}_\mathcal{V}$ and validates the chain against the trust anchors, and aborts otherwise. This is a check performed by the honest client only: it restricts the requests the client answers and touches none of the games of Sec. 3.1, in which an adversarial verifier may in any case be a registered Relying Party holding a valid access certificate.

Nothing changes in **aux**: since **aux** signs **ctx** in full, the device signature covers the identity of the verifier and its fresh nonce, in addition to the showing message, and this is what makes a showing useless outside the session for which it was produced. In particular, an adversary sitting between an honest verifier and an honest client, and relaying the request of the former to the latter, obtains a showing bound to the identity and to the nonce of that verifier, which it can neither present to a different verifier nor replay later. This is the role of the audience and nonce claims of the key-binding JWT in SD-JWT VC [FYC] and of the session transcript in the device authentication of mdoc [Int21], which **aux** reproduces.

On the single-show limitation. Credential-unlinkability is stated in Sec. 3.1 as a single-show notion, and it is natural to ask whether this is an artifact of CRACS. It is not; that is, under the constraints of this section, no scheme can do better. The argument is immediate. Proof of possession requires the verifier to check a signature under the device key, and the EUDI verifier is *required* by the ARF to do so with one of the standard verification algorithms (i.e., with no verifier-side zero-knowledge system). That is, the public key under which that signature is verified must be given to the verifier in the clear. Two showings performed with the same device key are therefore linkable by inspection. Unlinkability across showings of the same credential would require either a fresh key per showing, which contradicts the key being attested inside the WSCD at issuance time, or a proof that the signature verifies under *some* attested key without revealing which, which is exactly the verifier-side system that the ARF does not contemplate. Note that this is a limitation that the ARF itself acknowledges.

Batch issuance. The natural way to obtain show-show unlinkability is then to obtain several single-show credentials, which is what the ARF calls batch issuance and suggests in [EUD, Discussion Topic B]. A Key Attestation contains one or more public keys, so a client holding a KA with n keys may run n issuance sessions, the j -th on an attribute vector whose last field is the j -th key, and spend the resulting credentials one at a time. Each showing then discloses a different device key, and the showings are unlinkable.

Running CRACS for batch issuance naively costs n NIZK proofs, but the expensive part of each of them, the verification of the certificate demanded by the policy, is the same in all n sessions; only the last attribute and the commitments change. Standard commit-and-prove techniques [CLOS02, AGM18, CFQ19] can therefore be used.¹⁵ The cost drops to one policy circuit plus n linking proofs, and the shared proof can be reused later for further credentials.

Acknowledgments

We thank Leonardo Rufini for valuable feedback on this work. This work was partially supported by the research project funded by INDAM-GNCS in 2026 (CUP E53C25002010001). All authors are members of the Gruppo Nazionale Calcolo Scientifico–Istituto Nazionale di Alta Matematica (GNCS-INDAM).

References

- AGM18. Shashank Agrawal, Chaya Ganesh, and Payman Mohassel. Non-interactive zero-knowledge proofs for composite statements. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 643–673. Springer, Cham, August 2018. doi:10.1007/978-3-319-96878-0_22.

¹⁵That is, prove the shared part once, committing inside that proof to the witness the sessions have in common, and attach to each session a cheap proof that its own values are consistent with that same commitment.

- AHIV17. Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkatasubramanian. Ligerio: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2087–2104. ACM Press, October / November 2017. doi:10.1145/3133956.3134104.
- AHIV23. Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkatasubramanian. Ligerio: lightweight sublinear arguments without a trusted setup. *DCC*, 91(11):3379–3424, 2023. doi:10.1007/s10623-023-01222-8.
- BBL⁺25. Jan Bobolz, Emad Heydari Beni, Anja Lehmann, Omid Mirzamohammadi, Cavit Özbay, and Mahdi Sedaghat. Multi-verifier keyed-verification anonymous credentials. Cryptology ePrint Archive, Paper 2025/2156, 2025. Accepted at ASIACRYPT 2026. URL: <https://eprint.iacr.org/2025/2156>.
- BCS16. Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 31–60. Springer, Berlin, Heidelberg, October / November 2016. doi:10.1007/978-3-662-53644-5_2.
- BFF⁺26. Anna-Birgitta Burmeister, Anna Fennig, Andreas Franke, Karla Friedrichs, Anja Lehmann, Kurt-Kester Leißering, Konrad Letz, and Cavit Özbay. Invisible traces: Subversion attacks on batch-issued credentials. Cryptology ePrint Archive, Paper 2026/1229, 2026. URL: <https://eprint.iacr.org/2026/1229>.
- BL13. Foteini Baldimtsi and Anna Lysyanskaya. Anonymous credentials light. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 1087–1098. ACM Press, November 2013. doi:10.1145/2508859.2516687.
- BL26a. Magdalena Bertram and Anja Lehmann. Deniability for signed credentials: Revisiting the authenticated channel vs. signed data debate in the EUDI wallet. Cryptology ePrint Archive, Paper 2026/1648, 2026. Accepted at ARES 2026 Workshops. URL: <https://eprint.iacr.org/2026/1648>.
- BL26b. Christian Bormann and Anja Lehmann. SoK: Anonymous credentials for digital identity wallets. In Heinrich C. Pöhls and Chris J. Mitchell, editors, *Security Standardisation Research – SSR 2025*, Lecture Notes in Computer Science, pages 3–25. Springer, Cham, Switzerland, 2026. doi:10.1007/978-3-032-19567-8_1.
- BLNS23. Jonathan Bootle, Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Alessandro Sorniotti. A framework for practical anonymous credentials from lattices. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part II*, volume 14082 of *LNCS*, pages 384–417. Springer, Cham, August 2023. doi:10.1007/978-3-031-38545-2_13.
- BR96. Mihir Bellare and Phillip Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In Ueli M. Maurer, editor, *EUROCRYPT’96*, volume 1070 of *LNCS*, pages 399–416. Springer, Berlin, Heidelberg, May 1996. doi:10.1007/3-540-68339-9_34.
- BR98. Mihir Bellare and Phillip Rogaway. Pss: Provably secure encoding method for digital signatures, 1998. Submission to IEEE P1363, August 1998. URL: <https://www.cs.ucdavis.edu/~rogaway/papers/exact.html>.
- CAHLT25. Rutchathon Chairattana-Apirom, Franklin Harding, Anna Lysyanskaya, and Stefano Tessaro. Server-aided anonymous credentials. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VI*, volume 16005 of *LNCS*, pages 291–324. Springer, Cham, August 2025. doi:10.1007/978-3-032-01887-8_10.
- CDH⁺25. Sofia Celi, Alex Davidson, Hamed Haddadi, Gonçalo Pestana, and Joe Rowell. DiStefano: Decentralized infrastructure for sharing trusted encrypted facts and nothing more. In *NDSS 2025*. The Internet Society, February 2025.
- CDHK15. Jan Camenisch, Maria Dubovitskaya, Kristiyan Haralambiev, and Markulf Kohlweiss. Composable and modular anonymous credentials: Definitions and practical constructions. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 262–288. Springer, Berlin, Heidelberg, November / December 2015. doi:10.1007/978-3-662-48800-3_11.
- CFQ19. Matteo Campanelli, Dario Fiore, and Anaïs Querol. LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2075–2092. ACM Press, November 2019. doi:10.1145/3319535.3339820.
- Cha85. David Chaum. Security without identification: transaction systems to make big brother obsolete. *Commun. ACM*, 28(10):1030–1044, October 1985. doi:10.1145/4372.4373.
- CL01. Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer, Berlin, Heidelberg, May 2001. doi:10.1007/3-540-44987-6_7.
- CLLZ26. Sofia Celi, Anja Lehmann, Shai Levin, and Alexandros Zacharakis. Device binding for anonymous credentials on legacy phones. Cryptology ePrint Archive, Paper 2026/965, 2026. Accepted at IEEE S&P’27. URL: <https://eprint.iacr.org/2026/965>.

- CLOS02. Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002. doi:10.1145/509907.509980.
- Den21. Frank Denis. Blind RSA signatures in pure Rust. <https://github.com/jedisct1/rust-blind-rsa-signatures>, 2021. Accessed on 2026-09-17.
- DGC. EU Digital COVID Certificate: test data repository for test automation. <https://github.com/eu-digital-green-certificates/dgc-testdata>. Accessed on 2026-09-17.
- DJW23. Frank Denis, Frederic Jacobs, and Christopher A. Wood. RSA Blind Signatures. RFC 9474, October 2023. URL: <https://www.rfc-editor.org/info/rfc9474>, doi:10.17487/RFC9474.
- DV26. Pierpaolo Della Monica and Ivan Visconti. Improved concurrent-secure blind schnorr signatures. In *Advances in Cryptology – CRYPTO 2026*, page 457–489. Springer Nature Switzerland, 2026. doi:10.1007/978-3-032-35415-0_15.
- DVVZ26. Pierpaolo Della Monica, Ivan Visconti, Andrea Vitaletti, and Marco Zecchini. Acts: Attestations of contents in tls sessions. In *Proceedings 2026 Network and Distributed System Security Symposium*, NDSS 2026. Internet Society, 2026. doi:10.14722/ndss.2026.241861.
- eHe. eHealth Network. Electronic Health Certificate Specification. <https://github.com/ehn-dcc-development/hcert-spec>. Accessed on 2026-09-17.
- ENR⁺26. Liam Eagen, Hy Ngo, Vikas Rushi, Ying Tong, Moven Tsai, and Janabel Xia. OpenAC: Open design for transparent and lightweight anonymous credentials. Cryptology ePrint Archive, Paper 2026/251, 2026. URL: <https://eprint.iacr.org/2026/251>.
- EUD. European digital identity - architecture and reference framework. Accessed on 2026-09-16. URL: <https://eu-digital-identity-wallet.github.io/eudi-doc-architecture-and-reference-framework/3.0.0/>.
- Eur24. European Parliament and Council of the European Union. Regulation (eu) 2024/1183 of the european parliament and of the council of 11 april 2024 amending regulation (eu) no 910/2014 as regards establishing the european digital identity framework. Official Journal of the European Union, L 1183, 2024. URL: <https://eur-lex.europa.eu/eli/reg/2024/1183/oj/eng>.
- Eur25. European Cybersecurity Certification Group (ECCG) Sub-group on Cryptography. Agreed Cryptographic Mechanisms. ECCG Technical Report, April 2025. URL: https://certification.enisa.europa.eu/document/download/a845662b-ae0-484e-9191-890c4cfa7aaa_en?filename=ECCG+Agreed+Cryptographic+Mechanisms+version+2.pdf.
- FFK⁺26. Andrea Flamini, Karla Friedrichs, Jonathan Katz, Watson Ladd, Anja Lehmann, and Marek Sefranek. Issuer-hiding BBS-based anonymous credentials without policy keys. Cryptology ePrint Archive, Paper 2026/870, 2026. URL: <https://eprint.iacr.org/2026/870>.
- FFL26. Andrea Flamini, Karla Friedrichs, and Anja Lehmann. Constant-size issuer hiding for BBS credentials via randomizable keys. Cryptology ePrint Archive, Paper 2026/369, 2026. Accepted at ASIACRYPT 2026. URL: <https://eprint.iacr.org/2026/369>.
- FHLL26. Karla Friedrichs, Franklin Harding, Anja Lehmann, and Anna Lysyanskaya. Device-bound anonymous credentials with(out) trusted hardware. In Joan Daemen and Emmanuel Thomé, editors, *Advances in Cryptology – EUROCRYPT 2026, Part II*, volume 16542 of *Lecture Notes in Computer Science*, pages 345–375, Rome, Italy, May 10–14, 2026. Springer, Cham, Switzerland. doi:10.1007/978-3-032-25317-0_12.
- Fis06. Marc Fischlin. Round-optimal composable blind signatures in the common reference string model. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 60–77. Springer, Berlin, Heidelberg, August 2006. doi:10.1007/11818175_4.
- FLS⁺26. Andrea Flamini, Anja Lehmann, Giada Sciarretta, Mario Scuro, Nicola Smaniotto, Alessandro Tomasi, and Silvio Ranise. Comparing privacy-preserving revocation for the EUDI wallet. Cryptology ePrint Archive, Paper 2026/1824, 2026. Accepted at WPES 2026. URL: <https://eprint.iacr.org/2026/1824>.
- Fs26. Matteo Frigo and abhi shelat. Anonymous credentials from ECDSA. *IACR Communications in Cryptology*, 3(1), 2026. doi:10.62056/a3qjmgpxq.
- FW24. Georg Fuchsbaauer and Mathias Wolf. Concurrently secure blind Schnorr signatures. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part II*, volume 14652 of *LNCS*, pages 124–160. Springer, Cham, May 2024. doi:10.1007/978-3-031-58723-8_5.
- FYC. Daniel Fett, Kristina Yasuda, and Brian Campbell. Selective Disclosure for JWTs (SD-JWT). Internet-Draft draft-ietf-oauth-selective-disclosure-jwt, IETF OAuth Working Group. Accessed on 2026-09-17. URL: <https://datatracker.ietf.org/doc/draft-ietf-oauth-selective-disclosure-jwt/>.
- Git25. Github project. <https://anonymous.4open.science/r/CRACS>, 2025.
- Gro16. Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Berlin, Heidelberg, May 2016. doi:10.1007/978-3-662-49896-5_11.

- GRSB19. Sharon Goldberg, Leonid Reyzin, Omar Sagga, and Foteini Baldimtsi. Efficient noninteractive certification of RSA moduli and beyond. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 700–727. Springer, Cham, December 2019. doi:10.1007/978-3-030-34618-8_24.
- HSS23. Julia Hesse, Nitin Singh, and Alessandro Sorniotti. How to bind anonymous credentials to humans. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023*, pages 3047–3064. USENIX Association, August 2023. URL: <https://www.usenix.org/conference/usenixsecurity23/presentation/hesse>.
- idea. iden3. circom: Circuit Compiler for Zero-Knowledge Proofs. <https://github.com/iden3/circom>. Version 2.2.3. Accessed on 2026-09-17.
- ideb. iden3. rapidsnark: Fast zkSNARK prover. <https://github.com/iden3/rapidsnark>. Accessed on 2026-09-17.
- idec. iden3. snarkjs: zkSNARK implementation in JavaScript and WASM. <https://github.com/iden3/snarkjs>. Version 0.7.6. Accessed on 2026-09-17.
- Int21. International Organization for Standardization. ISO/IEC 18013-5:2021 – Personal identification – ISO-compliant driving licence – Part 5: Mobile driving licence (mDL) application. International Standard, 2021. Accessed on 2026-09-17. URL: <https://www.iso.org/standard/69084.html>.
- JL17. Simon Josefsson and Ilari Liusvaara. Edwards-Curve Digital Signature Algorithm (EdDSA). RFC 8032, IRTF, January 2017. §5.1.6, p. 13. URL: <https://www.rfc-editor.org/info/rfc8032>, doi:10.17487/RFC8032.
- Jon15. Michael B. Jones. JSON Web Algorithms (JWA). RFC 7518, May 2015. URL: <https://www.rfc-editor.org/info/rfc7518>, doi:10.17487/RFC7518.
- JWET18. Michael B. Jones, Erik Wahlstroem, Samuel Erdtman, and Hannes Tschofenig. CBOR Web Token (CWT). RFC 8392, May 2018. URL: <https://www.rfc-editor.org/info/rfc8392>, doi:10.17487/RFC8392.
- KL14. Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall, CRC Press, third edition, 2014.
- KLR23. Julia Kastner, Julian Loss, and Omar Renawi. Concurrent security of anonymous credentials light, revisited. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 45–59. ACM Press, November 2023. doi:10.1145/3576915.3623184.
- KS24. Abhiram Kothapalli and Srinath Setty. NeutronNova: Folding everything that reduces to zero-check. Cryptology ePrint Archive, Paper 2024/1606, 2024. Accepted at CRYPTO 2026. URL: <https://eprint.iacr.org/2024/1606>.
- KS25. Jonathan Katz and Marek Sefranek. Issuer hiding for BBS-based anonymous credentials. Cryptology ePrint Archive, Report 2025/2080, 2025. Accepted at IACR PKC 2026. URL: <https://eprint.iacr.org/2025/2080>.
- KS26. Darya Kaviani and Srinath Setty. Vega: Low-latency zero-knowledge proofs over existing credentials. In *2026 IEEE Symposium on Security and Privacy*, pages 1951–1969, San Francisco, CA, USA, May 18–20, 2026. IEEE Computer Society Press. doi:10.1109/SP63933.2026.00182.
- KST22. Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 359–388. Springer, Cham, August 2022. doi:10.1007/978-3-031-15985-5_13.
- Leh25. Anja Lehmann. Eu digital identity and anonymous credentials - a happy end? Talk given at RWC 2025, 2025. Video at <https://youtu.be/UpQHWObCx4I>.
- LSZ26. Anja Lehmann, Andrey Sidorenko, and Alexandros Zacharakis. Vision: A modular framework for anonymous credential systems. In Henrich C. Pöhls and Chris J. Mitchell, editors, *Security Standardisation Research – SSR 2025*, Lecture Notes in Computer Science, pages 46–68. Springer, Cham, Switzerland, 2026. doi:10.1007/978-3-032-19567-8_3.
- Lys23. Anna Lysyanskaya. Security analysis of RSA-BSSA. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part I*, volume 13940 of *LNCS*, pages 251–280. Springer, Cham, May 2023. doi:10.1007/978-3-031-31368-4_10.
- MKJR16. Kathleen Moriarty, Burt Kaliski, Jakob Jonsson, and Andreas Rusch. PKCS #1: RSA Cryptography Specifications Version 2.2. RFC 8017, November 2016. URL: <https://www.rfc-editor.org/info/rfc8017>, doi:10.17487/RFC8017.
- MLs25. René Mayrhofer, Anja Lehmann, and abhi shelat. A brief note on cryptographic pseudonyms for anonymous credentials. arXiv preprint arXiv:2510.05419, 2025. URL: <https://arxiv.org/abs/2510.05419>.
- MPD25. Jules Maire and Alan Pulval-Dady. Blind ECDSA from the ECDSA assumption. *IACR Communications in Cryptology*, 2(3), 2025. doi:10.62056/ahjbhee-3.
- Ope. Openid for verifiable credential issuance 1.0. Accessed on 2026-06-22. URL: https://openid.net/specs/openid-4-verifiable-credential-issuance-1_0.html.

- Pas04. Rafael Pass. *Alternative Variants of Zero-Knowledge Proofs*. Licentiate thesis, KTH Royal Institute of Technology, Stockholm, Sweden, 2004. URL: <https://kth.diva-portal.org/smash/record.jsf?pid=diva2:9472>.
- RWGM23. Michael Rosenberg, Jacob D. White, Christina Garman, and Ian Miers. zk-creds: Flexible anonymous credentials from zkSNARKs and existing identity infrastructure. In *2023 IEEE Symposium on Security and Privacy*, pages 790–808. IEEE Computer Society Press, May 2023. doi:10.1109/SP46215.2023.10179430.
- Sch22. Jim Schaad. CBOR Object Signing and Encryption (COSE): Structures and Process. RFC 9052, August 2022. URL: <https://www.rfc-editor.org/info/rfc9052>, doi:10.17487/RFC9052.
- Set20. Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 704–737. Springer, Cham, August 2020. doi:10.1007/978-3-030-56877-1_25.
- XYWY24. Xiang Xie, Kang Yang, Xiao Wang, and Yu Yu. Lightweight authentication of web data via garble-then-prove. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024*. USENIX Association, August 2024. URL: <https://www.usenix.org/conference/usenixsecurity24/presentation/xie-xiang>.
- ZMM⁺20. Fan Zhang, Deepak Maram, Harjasleen Malvai, Steven Goldfeder, and Ari Juels. Deco: Liberating web data using decentralized oracles for tls. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, CCS '20*, page 1919–1938, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3372297.3417239.

A EUDI Cryptographic Signatures for Credentials

The Architecture and Reference Framework [EUD] defines the technical requirements for implementing an EUDI Wallet and the solutions that interact with it. A credential is the machine-processable representation of an attestation of attributes. It contains the attested attributes, issuer and validity metadata, and cryptographic material used to authenticate the credential and to bind it to a particular wallet or device.

The ARF identifies three standardised formats for such attestations: (i) the mdoc format, encoded in CBOR and specified by ISO/IEC 18013-5; (ii) SD-JWT VC, encoded using JSON-based JWT/JWS structures; and (iii) W3C Verifiable Credentials Data Model v2.0, normally represented using JSON-LD. However, mdoc and SD-JWT VC support all the EUDI Wallet’s attestation categories, whereas W3C VCDM v2.0 supports only one.

We therefore focus on mdoc and SD-JWT VC. These formats use different signing frameworks: COSE for mdoc and JOSE for SD-JWT VC. They support several public-key signature families, among which ECDSA, EdDSA and RSA.

ECDSA is commonly selected as the interoperability baseline in EUDIW profiles, while support for EdDSA and RSA may depend on the applicable credential format, trust framework, national rulebook, and WSCD implementation. However, ECDSA is known to be inefficient or relying on strong assumption when used in conjunction with more advanced cryptographic tools, such as blind signature schemes [MPD25] or presentations in anonymous credentials [Leh25]. In EdDSA (Section 5.1.6 of [JL17]), the signature challenge is obtained deterministically from a prefix computed on the signer’s secret key and the message to be signed; this makes it impossible to obtain a blind issuance of credentials with this signature scheme.

Therefore, for the subsequent cryptographic analysis, we select RSA as the reference signature scheme. This is a research choice consistent with certain EUDI Wallet deployments. We select RSA because it is a standardised, well-studied signature scheme; it is included in JOSE; it is recognised in the ECCG cryptographic guidance [Eur25]; and it suits the anonymous-credential constructions presented in this work.

B Related Work

In the last decades, a large body of work has been dedicated to anonymous credentials. In this section we restrict our attention to two families of works. The first (i) adds privacy at presentation time to credentials targeting the EUDIW, either building on the ECDSA-signed formats it adopts or proposing alternatives to them. The second (ii) consists of anonymous credential schemes, not necessarily tied to the EUDIW, in which issuance limits what the issuer learns about the attributes. Our construction sits at the intersection: it acts at issuance time, against a censoring issuer, while leaving the presentation of EUDIW credentials untouched.

For (i), Frigo and shelat [Fs26] build anonymous credentials directly on top of legacy ECDSA-signed credentials, so that neither the issuer nor the secure element of the device needs to change. Contrary to our approach, they leave issuance untouched and move all the complexity to the presentation, whereas we leave the presentation untouched and act at issuance. During the presentation, the holder produces a zero-knowledge proof of knowledge of an ECDSA signature by the issuer on the credential, together with proofs of the SHA-256 hashing and of the CBOR parsing of the ISO mDL format, and of the ECDSA device-binding signature under the committed device key. To make this feasible, they design a new proof system that combines sumcheck with the Ligerio [AHIV17] argument, relying only on SHA-256 as assumption and requiring no trusted setup, and they contribute several optimizations for P-256 arithmetic, whose field is not FFT-friendly. The resulting proofs for a full mDL presentation take a few hundred milliseconds on a phone and are a few hundred kilobytes in size. However, their approach is performed with a monolithic circuit-based proof system that is not standardized and cannot be verified by the verifiers deployed today according to the ARF. We instead keep the showing standard for the verifier and act at issuance time, which moreover allows us to protect the user from a censoring issuer, a threat that is outside their model.

Along the same line, Kaviani and Setty [KS26] propose Vega, a transparent proof system for existing ECDSA-signed credentials that, like Longfellow, leaves the issuer untouched and supports device binding. Building on Spartan [Set20] and on folding schemes [KST22, KS24], they split the prover’s work into a precomputation phase, run once per credential on the part of the witness that does not depend on the verifier, and an online phase that at each presentation only handles the verifier-dependent part, with the precomputed commitments rerandomized at every reuse. This cuts the proving and verification time of Longfellow by about three and six times, respectively, but the presentation remains a non-standard zero-knowledge proof that the verifiers deployed according to the ARF cannot check, and the issuer is trusted.

The same split between a per-credential precomputation and a lightweight online phase underlies OpenAC [ENR⁺26], which targets the SD-JWT and mDL credentials of the EUDIW: a Spartan [Set20] proof that the issuer’s signature verifies and that a Pedersen commitment opens to the attributes is computed once and rerandomized, and each presentation only proves the predicates and the device signature against the rerandomized commitment. Despite the authors’ mapping onto the ARF requirements, the presentation remains a non-standard zero-knowledge proof for the verifier, and the issuer is assumed honest.

Bertram and Lehmann [BL26a] address a different privacy property of the ECDSA-signed credentials adopted by the EUDIW, namely plausible deniability: a verifier holding a publicly verifiable issuer signature can prove to third parties that a presentation took place. They observe that the ECDSA verification equation directly yields a discrete-logarithm relation, and turn the presentation into a universal designated-verifier signature through an OR proof of knowledge of either the ECDSA witness or the verifier’s secret key, which the verifier could have simulated by itself. The construction requires no change to the signature scheme, the credential format, or the issuer, and adds only two Schnorr proofs to the presentation. The problem is orthogonal to ours, but the work

shares our design philosophy of adding privacy on top of standard ECDSA credentials without touching the issuer.

Mayrhofer, Lehmann, and shelat [MLs25] discuss verifier-specific pseudonyms for the EUDIW, derived through a pseudo random function keyed with a secret seed committed in the credential. Both instantiations they propose, a circuit-based proof over mdoc or SD-JWT credentials and a BBS-based one, require a zero-knowledge presentation layer that is, once again, non-standard for the verifier.

A second line of work in (i) replaces the ECDSA-signed credential altogether with a BBS-based one. Celi et al. [CLLZ26] address device binding for pairing-based anonymous credentials on secure elements that only support ECDSA over P-256. During the showing, the holder provides a zero-knowledge proof derived from a BBS signature for the attributes and a separate proof of possession of the device key, obtained either with a circuit-based proof system (Bulletproofs or PLONK) or with a Σ -protocol; moreover, two of their three instantiations run over the Tom curve T-256, which is not standardized by NIST, ISO, or W3C. Hence, while the resulting showing is privacy-preserving, it relies on non-standard tools, is not compliant with the current ARF, and cannot be verified by the verifiers deployed in production today. Our construction instead preserves privacy while keeping the showing fully standard for the verifier. This construction is in fact the concrete instantiation used by Lehmann et al. [LSZ26] to demonstrate their vision of building and standardizing anonymous credential systems in a modular manner, by identifying re-usable components and designing them in a way that allows for flexible extension: BBS is used for the core credential and ECDSA, proven via the same Σ -protocols over T-256, for device binding. Their security treatment is only informal, and even their BBS presentation deviates from the IETF draft, which supports selective but not committed disclosure. Friedrichs et al. [FHLL26] also study device binding for BBS credentials, focusing on the trust placed in the secure element: they formalize privacy against subverted or fully malicious secure elements, including the remote HSM setting considered for the EUDIW, and give three constructions in which the secure element only computes a Schnorr or (blind) BLS signature. The core credential is still BBS with its zero-knowledge presentation, so the showing is again non-standard for the verifier.

Issuer hiding for BBS credentials, i.e., presentations that only reveal that the credential was issued by some issuer within a policy set rather than by a specific one, was first achieved by Katz and Sefranek [KS25]: presentations are constant-size, but each verifier must generate a policy key with a secret verification counterpart, so that presentations are no longer publicly verifiable, and security holds only in the generic group model. Two recent works remove these drawbacks. Flamini et al. [FFK⁺26] randomize the BBS issuer public key, adapt the signature accordingly, and prove in zero knowledge that the randomized key derives from one of the keys in the policy through an OR proof, whose size is linear, or logarithmic when using Stacking Sigmas or Groth–Kohlweiss proofs, in the policy size; no policy key is needed and security reduces to the unforgeability of BBS. Flamini, Friedrichs, and Lehmann [FFL26] instead avoid the OR proof altogether: a trusted party publishes, for every issuer key in the policy, a tag computed with a secret exponent; the holder raises its issuer key and the corresponding tag to the same random exponent and sends both in the clear, and the verifier checks their consistency with a pairing. This yields constant-size presentations, at the price of a trusted policy key and of a security analysis in the algebraic group model. All these works target a privacy goal orthogonal to ours and inherently rely on BBS signatures for the credential and on the corresponding zero-knowledge presentation: as discussed in the introduction, this makes the showing non-standard and thus not verifiable by the verifiers currently deployed according to the ARF. Moreover, in all these works issuance is the plain BBS signing procedure, run by an issuer that sees the attributes in the clear and decides whether to sign them, and a corrupt issuer

is explicitly excluded from their security models: issuer hiding protects the holder from the verifier, not from a censoring issuer.

Bobolz et al. [BBL⁺25] extend keyed-verification anonymous credentials, where issuer and verifier share a secret key, to multiple verifiers: each verifier registers with the issuer and receives an issuer-derived secret key, and the holder must blindly obtain a separate credential for every verifier it wants to interact with. Besides the non-standard presentation, this setting is incompatible with the EUDIW, where verifiers form an open set that neither registers with the issuer nor holds issuer-derived secrets, and a single credential must be presentable to any of them; moreover, issuance is still run by the issuer, which can refuse it.

For (ii), the only work we are aware of that considers a malicious issuer of EUDIW credentials is that of Burmeister et al. [BFF⁺26], which analyzes a problem orthogonal to the one analyzed in this paper. Specifically, they studied unlinkability in the case of batch issuing of EUDIW credentials under subversion attacks: indeed, a malicious issuer could trace users through the salted hashes it signs and the signature value itself. In their analysis, they study various and elaborated techniques to realize such attacks and suggest lightweight mechanisms to provably mediate them; the issuer, however, still learns all the attributes it signs.

Hesse et al. [HSS23] focus on a problem orthogonal to the one studied in this paper, namely how to guarantee that a digital credential has been issued to the person standing in front of the verifier in the physical world. Their paper explicitly considers “blind issuance of attributes” (in Section 5) and models a functionality in which part of the attributes of the credential is chosen by the user and part by the issuer, with the issuer learning nothing about the attributes chosen by the user. This functionality is however limited: the issuer has no guarantee that the user-chosen attributes satisfy any property, which devalues such attributes since they are not checked in any way. Bootle et al. [BLNS23] provide a post-quantum construction of anonymous credentials. Once again the privacy guarantees analyzed in their paper are limited: unlinkability holds only for those attributes that are disclosed during the issuance process to the issuer.

Chairattana-Apirom et al. [CAHLT25] propose an anonymous credential system where, in the process of showing a credential, the holder is helped by additional auxiliary information generated by an additional (anonymous) interaction with the issuer. They also propose a formal definition of such a system. However, their definition enables constructions (e.g., a NIZK-based construction proving that the user holds an attribute vector m such that a predicate over m is satisfied) in which, at the end of the signing session, the issuer learns whether or not it has issued a credential, which opens the door to censorship. In our notion, instead, the issuer does not learn whether the user holds a vector m that satisfied a shared predicate. Indeed, in some use cases a malicious issuer may, for example, keep count of how many valid credentials it has issued for a given predicate and, after some threshold, decide to stop issuing them. Even worse, such an issuer may obtain evidence of the issuance (e.g., in the NIZK-based example above, the proof itself together with the rest of the transcript) that it has issued a valid credential for a given predicate to a user, and may later show this evidence to a third party, acting in that party’s interest and thereby determining the issuance or non-issuance of specific credentials.

Finally, for a complete analysis of the ongoing standardization effort we refer the reader to this comprehensive systematization of knowledge [BL26b].

C Experiments

In this section, we describe the implementation of a use case in which a user requests the issuer to issue an SD-JWT VC credential (one of the credential formats supported by the ARF), derived from

a credential that is not compliant with the ARF (specifically, CWT). The goal of the experiment is to test whether the statement that the client proves at issuance, which involves RSA operations (the PSS encoding and RSA blinding of the committed attributes), can be expressed and proved with a SNARK at a practical cost.

Scenario. The input credential is a real EU Digital COVID Certificate (EUDCC, commonly known as Green Pass) [eHe, DGC]. In our implementation, the EUDCC is encoded as a CWT claims set [JWET18] wrapped in a COSE_Sign1 object [Sch22].

We chose this document because it is a concrete example of a credential that carries personal attributes and is *not* an ARF credential [EUD]. As a test vector, we use the certificate C01 of the official interoperability corpus [DGC], signed with PS256, that is, RSA-PSS with a 2048-bit modulus and SHA-256 [MKJR16, Jon15].

Mapping to CRACS. Here we show how the objects of our implementation instantiate those of Fig. 4.

The certificate `cert` is the raw byte string representing the EUDCC. The policy `pol = (Pol, AttrChk)` is fixed as follows: `Pol(cert)` parses the CBOR structure, rebuilds the COSE signature input (the `Sig_structure` with context `Signature1`) and verifies the RSA-PSS signature of the Document Signer under its public key, which is part of `pol` and hence public. `AttrChk(cert, attrs)` selects the subject map by a public path of map keys and checks that `attrs` is exactly the list of its entries; in our instance $\ell = 4$.

The showing commitment `Comvy` is the salted-hash scheme `ComH` of App. D.3, with `H` instantiated with SHA-256: c'_i is the SHA-256 digest of a fixed-width preimage that contains a_i and a random 16-byte salt, and d'_i is the salt. The signed message m is a salted hash as in SD-JWT VC [FYC]. It also includes the namespace string preceded by its length, the number of fields, and the pairs $(\text{digestID}_i, c'_i)$ for $i \in [\ell]$; that is, m is the concatenation $c'_1 \parallel \dots \parallel c'_\ell$ of Fig. 4 interleaved with public framing bytes, and has length $2 + |\text{ns}| + 33\ell$ bytes, where `ns` is the namespace. The encoding $\mu = \text{PSS.Enc}(m)$ and the blinded value $t = \mu\kappa^e \bmod N$ are computed inside the circuit, with κ a blinding factor chosen by the client.

The issuer's reply $t^d \bmod N$ is produced by the issuer; the client unblinds it and checks that $\sigma^e \equiv \mu \pmod{N}$, exactly as in `CRACRSA.IClient1`.

In a showing, the client sends the tuple $(\text{c}_{\text{vy}}, \sigma, \text{opn}, (d'_i)_{i \in \text{flds}(\phi_{\text{vy}})}, \text{aux})$ with `aux` empty, and the verifier runs the checks of `CRACRSA.Verifier1`: it recomputes the disclosed digests, matches them against the signed list, and verifies the RSA-PSS signature. In our runs, ϕ_{vy} discloses two of the four fields.

Inputs and outputs of the circuit. The circuit implements the Honest Branch of `RCRACRSA`. Its private inputs, i.e., the witness w , are the raw bytes of the EUDCC, the ℓ salts d'_i , the PSS salt¹⁶ and the blinding factor κ . Its public inputs, i.e., the statement $\mathfrak{x}$, are the Document Signer public key and the path of map keys that locates the attributes, both part of `pol`, and the issuer key $\text{vk} = (N, e)$. Its output is the blinded value t , which the client sends to the issuer together with π .

¹⁶Our construction and its security analysis use a deterministic encoding `PSS.Enc`, in which the salt is derived from the message (see App. D.2), so that no salt appears in the witness of `RCRACRSA`. Our implementation instead lets the client sample the salt at random and supplies it as an additional private input of the circuit. This departs from Fig. 4, but does not affect the measured costs. Deriving the salt from m would add at most one hash evaluation to a circuit of about 2.64M constraints.

Party	Step	Time	Peak memory	Output
Client (C)	Witness generation (wasm)	≈ 22 s	–	80 MB
	Groth16 proving (rapidsnark)	5.0–6.5 s	2.77 GB	706 B

Table 1: Client-side costs to generate a valid witness and a Groth16 proof with snarkjs and rapidsnark. Proving times are the range over three runs; memory is the maximum resident set size.

C.1 Main Technical Choices

We wrote the circuit in circom [idea] (version 2.2.3) with the support of Claude Code and compiled it into a Rank-1 Constraint System (R1CS).

To make the in-circuit encoding and the off-circuit blind-signature library agree byte for byte, we use a fork of `rust-blind-rsa-signatures` [Den21]. We generate proofs with Groth16 [Gro16], using rapidsnark [ideb] as prover and snarkjs [idec] (version 0.7.6) for the setup and for cross-checking verification. The issuer verifies π and blindly signs t with a 2048-bit key; the client unblinds the reply with κ^{-1} and verifies the resulting RSA-PSS signature on the digest list, which we re-verify with an independent library.¹⁷ Showings are then built by revealing the salted preimages of any subset of the entries, two out of four in our runs; the verifier recomputes the digests, matches them against the signed list and verifies the issuer’s RSA-PSS signature, which are exactly the checks of `CRACRSA.Verifier1` once `ctx` and `aux` are ignored. All components used to evaluate the performance of our CRACS are available on GitHub [Git25].

C.2 Experimental Results

All measurements were taken on a single laptop, a MacBook with an Apple M4 SoC (10 cores) and 16 GB of RAM. The circuit has about 2.64M non-linear constraints, of which about 992k are due to the in-circuit RSA-PSS verification of the Document Signer signature, the rest being due to the verified CBOR parse, the four SHA-256 digests, the PSS encoding and the blinding. Compiling it takes under a minute, whereas the Groth16 setup takes about 25 minutes and 5.8 GB of memory and produces a 1.5 GB proving key.

Tab. 1 reports the results of the experiments and shows the feasibility of our approach. On the client side, preparing the inputs and blinding takes about one second, generating the witness about 22 seconds, and proving between 5.0 and 6.5 seconds with a peak of 2.77 GB of memory. On the issuer side, verifying the proof takes 10 ms with the native verifier (190 ms with snarkjs), and the blind-signature round, including the unblinding and the check of the client, takes less than one second; the proof is 706 bytes long.

The small running times and the limited resources needed to obtain the derived credentials prove the viability of our approach in the considered scenario.

D Additional Preliminaries

D.1 Signature Scheme

A signature scheme is a tuple of efficient algorithms $\text{Sig} = (\text{Sig.KG}, \text{Sig.Sig}, \text{Sig.Vry})$ where:

¹⁷The Python `cryptography` library, <https://cryptography.io>.

- $\text{Sig.KG}(1^\lambda) \rightarrow (\text{sk}, \text{vk})$ on input the security parameter, outputs a signing key sk and a verification key vk .
- $\text{Sig.Sig}(\text{sk}, m) \rightarrow \sigma$ on input a signing key sk and a message $m \in \{0, 1\}^*$, outputs a signature σ .
- $\text{Sig.Vry}(\text{vk}, m, \sigma) \rightarrow 0/1$ is deterministic and on input a verification key vk , a message m and a signature σ , outputs 1 if σ is valid and 0 otherwise.

and such that correctness and unforgeability as defined below hold.

Definition 5 (Correctness). *A signature scheme Sig is perfectly correct if for all $\lambda \in \mathbb{N}$ and $m \in \{0, 1\}^*$:*

$$\Pr \left[1 = \text{Sig.Vry}(\text{vk}, m, \sigma) \mid \begin{array}{l} (\text{sk}, \text{vk}) \leftarrow \text{Sig.KG}(1^\lambda), \\ \sigma \leftarrow \text{Sig.Sig}(\text{sk}, m) \end{array} \right] = 1$$

Definition 6 (Unforgeability). *Consider the game EUF-CMA, described in Fig. 5. A signature scheme Sig satisfies existential unforgeability under chosen-message attacks (EUF-CMA) if for every PPT adversary $\mathcal{A}$ the advantage $\text{Adv}_{\text{Sig}}^{\text{euf-cma}}(\mathcal{A}, \lambda)$ is negligible in λ where*

$$\text{Adv}_{\text{Sig}}^{\text{euf-cma}}(\mathcal{A}, \lambda) := \Pr[\text{EUF-CMA}_{\text{Sig}}^{\mathcal{A}}(\lambda) = 1].$$

Game $\text{EUF-CMA}_{\text{Sig}}^{\mathcal{A}}(\lambda)$

1. The key generation algorithm Sig.KG is run on input the security parameter, outputting a key pair (sk, pk) . A set $\mathcal{Q}$ is initialized to $\emptyset$.
2. A signing oracle Sign is initialized with access to the signing key sk and defined as follows:
On input a message m , the oracle runs the signing algorithm Sig.Sig on input (sk, m) . It obtains a signature σ , adds m to the set $\mathcal{Q}$, and returns σ .
3. The adversary $\mathcal{A}$ is given pk and access to the oracle Sign . After interacting with the oracle, it outputs a message-signature pair (m^*, σ^*) .
4. The game outputs 1 if: (a) $m^* \notin \mathcal{Q}$, and (b) $\text{Sig.Vry}(\text{pk}, m^*, \sigma^*) = 1$. Otherwise 0.

Fig. 5: The EUF-CMA game for the signature scheme Sig .

D.2 RSA Signature

We work with RSA-PSS as standardized in PKCS#1 v2.2 [MKJR16], rather than with the original PSS proposal of Bellare and Rogaway [BR96]. The two differ in ways that matter for a precise statement: the standard hashes the message before encoding, admits a variable salt length and inserts fixed padding strings at several stages (see [Lys23, App. B] for a discussion of these deviations). Since our construction is intended to work with deployed verifiers, we fix the standardized variant throughout and denote the resulting scheme by RSA , as described in Fig. 6.

The scheme is built from two fixed components. The first is the standard RSA key generator RSAGen , which on input 1^λ outputs (N, e, d) where N is an RSA modulus of the λ -bit length, e is coprime to $\varphi(N)$, and $ed \equiv 1 \pmod{\varphi(N)}$. The second is the PSS function generator PSSGen of Def. 7, instantiated with the parameter set of [MKJR16]. The PSS encoding employs two cryptographic hash functions, which are modeled as random oracles in the original analysis of PSS [BR96, Lys23]; we instead assume standard-model security of the concrete standardized instantiation, as stated in Asm. 1. We refer to [DVVZ26, MKJR16, BR98] for a detailed description of the standard, and follow [DVVZ26] in our treatment of the security of RSA-PSS in practical deployments.

Signature RSA

- **Key Generation** $\text{RSA.KG}(1^\lambda)$:
 1. Generate an RSA key pair $(N, e, d) \leftarrow \text{RSAGen}(1^\lambda)$.
 2. Generate the PSS parameters $\text{PSS} \leftarrow \text{PSSGen}(1^\lambda, N)$.
 3. Output the key pair (sk, vk) , where the signing key is defined as $\text{sk} := (N, e, d, \text{PSS})$ and the verification key as $\text{vk} := (N, e, \text{PSS})$.
- **Signature Generation** $\text{RSA.Sig}(\text{sk}, m)$:
 1. Compute the PSS encoding of the message as $\mu \leftarrow \text{PSS.Enc}(m)$.
 2. Compute the RSA signature $\sigma := \mu^d \pmod{N}$ and output σ .
- **Signature Verification** $\text{RSA.Vry}(\text{vk}, m, \sigma)$:
 1. Compute the RSA inverse $\mu' := \sigma^e \pmod{N}$.
 2. Output $\text{PSS.Vry}(m, \mu')$.

Fig. 6: The RSA-PSS signature scheme RSA, built from the RSA key generator RSAGen and the PSS function generator PSSGen.

Definition 7. A PSS function generator PSSGen is a PPT algorithm that, on input 1^λ and a modulus $N \in \mathbb{N}^+$, outputs the description of a pair of deterministic algorithms $\text{PSS} := (\text{PSS.Enc}, \text{PSS.Vry})$, where $\text{PSS.Enc} : \{0, 1\}^* \rightarrow \mathbb{Z}_N$ and $\text{PSS.Vry} : \{0, 1\}^* \times \mathbb{Z}_N \rightarrow \{0, 1\}$, such that $\text{PSS.Vry}(m, \text{PSS.Enc}(m)) = 1$ for every $m \in \{0, 1\}^*$. Both algorithms internally make black-box use of two hash functions treated as random oracles (with proper domain separation) in the security analysis of the RSA signature scheme.

We consider the PSS encoding as a function of the message. Note that the standard leaves the choice of explicitly introduce a salt to the function as input to the signer, and deriving it deterministically from the message preserves the salt length defined by [Jon15], so a verifier performing the standard verification of [MKJR16] is unaffected.

Assumption 1 (Security of RSA-PSS) The signature scheme RSA of Fig. 6, instantiated with RSAGen and PSSGen as fixed above, is unforgeable according to Def. 6. That is, for every PPT adversary $\mathcal{A}$, the advantage $\text{Adv}_{\text{RSA}}^{\text{euf-cma}}(\mathcal{A}, \lambda)$ is negligible in λ .

In Asm. 1, we assume the security of RSA-PSS as it is used in practice. Specifically, we work in the standard model without random oracles, assuming that RSA-PSS is secure when instantiated with a fixed hash function. This assumption was previously used in [DVVZ26], and very similar ones appear in [FW24, DV26]. Following their argumentation, although this assumption lacks a formal theoretical proof, it is considered uncontroversial in practice given the extensive deployment of RSA-PSS over the years.

D.3 Commitment Scheme

A (non-interactive) commitment scheme for a message space $\mathcal{M}_{\text{Com}}$ is a tuple of efficient algorithms $\text{Com} = (\text{Com.Cmt}, \text{Com.Opn})$ where:

- $\text{Com.Cmt}(m) \rightarrow (c, d)$ on input a message $m \in \mathcal{M}_{\text{Com}}$, outputs a pair of values (c, d) , consisting of the commitment c and the decommitment/opening information d .
- $\text{Com.Opn}(m, c, d) \rightarrow 0/1$ is deterministic and on input a message m a commitment c , an opening information d outputs 1 if, using d , it accepts m as the decommitment of c , otherwise 0.

and such that correctness, hiding and binding, as defined below, hold. Hiding and binding each come in a computational and in a perfect flavour; we define both, since the two commitment schemes used in Sec. 3.1 require different combinations of them.

Definition 8 (Correctness). *A commitment scheme Com is perfectly correct if for any $\lambda \in \mathbb{N}$ and message $m \in \mathcal{M}_{\text{Com}}$, the probability $\Pr[\text{Com.Opn}(m, c, d) = 1 \mid (c, d) \leftarrow \text{Com.Cmt}(m)]$ is equal to 1.*

Definition 9 (Hiding). *Consider the HDN game, described in Fig. 7. A commitment scheme Com is computationally (resp. statistically) hiding if, for every PPT (resp. possibly unbounded) adversary $\mathcal{A}$, the advantage $\text{Adv}_{\text{Com}}^{\text{hdn}}(\mathcal{A}, \lambda)$ is negligible in λ , where:*

$$\text{Adv}_{\text{Com}}^{\text{hdn}}(\mathcal{A}, \lambda) := \left| \frac{1}{2} - \Pr[\text{HDN}_{\text{Com}}^{\mathcal{A}}(\lambda) = 1] \right|.$$

Game $\text{HDN}_{\text{Com}}^{\mathcal{A}}(\lambda)$

1. Sample a uniformly random bit $b \leftarrow \{0, 1\}$.
2. Define a commitment oracle Cmt as follows:
On input a pair of messages $(m_0, m_1) \in \mathcal{M}_{\text{Com}}$, compute $(c, d) \leftarrow \text{Com.Cmt}(m_b)$ and output c .
3. The adversary $\mathcal{A}$ is given as input the security parameter 1^λ and oracle access to Cmt . After interacting with the oracle, it outputs a bit b' . The game outputs 1 if $b = b'$, and 0 otherwise.

Fig. 7: The HDN game.

Definition 10 (Binding). *Consider the BND game, described in Fig. 8. A commitment scheme Com is computationally binding if, for every PPT adversary $\mathcal{A}$, the advantage $\text{Adv}_{\text{Com}}^{\text{bnd}}(\mathcal{A}, \lambda)$ is negligible in λ , where:*

$$\text{Adv}_{\text{Com}}^{\text{bnd}}(\mathcal{A}, \lambda) := \left| \Pr[\text{BND}_{\text{Com}}^{\mathcal{A}}(\lambda) = 1] \right|.$$

Game $\text{BND}_{\text{Com}}^{\mathcal{A}}(\lambda)$

1. On input 1^λ , the adversary $\mathcal{A}$ outputs a tuple (c, m_0, m_1, d_0, d_1) .
2. The game outputs 1 if: **(a)** for every $b \in \{0, 1\}$ the algorithm Com.Opn on input (m_b, c, d_b) outputs 1, and **(b)** $m_0 \neq m_1$. Otherwise, 0.

Fig. 8: The BND game.

Standardized EUDI Wallet Commitment Instantiation. We now describe the commitment instantiation prescribed by the EUDI Wallet standard. Specifically, as discussed in App. A, EUDI Wallet credentials follow the JOSE/COSE encoding format, which represents the attributes contained in a verifiable credential through salted-hash commitments.

Accordingly, we consider the salted-hash commitment scheme used by the standard. We briefly define the scheme below and state the security properties that we require from it.

That is, we define such a commitment as $\text{Com}_H = (\text{Com}_H.\text{Cmt}, \text{Com}_H.\text{Opn})$, and it works as follows. On input a message $m \in \mathcal{M}_{\text{Com}}$ with $\mathcal{M}_{\text{Com}} = \{0, 1\}^*$, the algorithm $\text{Com}_H.\text{Cmt}$ first samples $r \leftarrow \{0, 1\}^\lambda$ and then it computes $H(m\|r)$. Finally, it outputs as c the tuple $H(m\|r)$ and as d the r . The algorithm $\text{Com}_H.\text{Opn}$ on input a message m and c, d as just defined simply checks if $H(m\|d)$ is equal to c ; if so it outputs 1, otherwise it outputs 0.

Assumption 2 (Security of Com_H) Let $H : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$ be modeled as a CR hash function. Let $\text{Com}_H = (\text{Com}_H.\text{Cmt}, \text{Com}_H.\text{Opn})$ be the commitment scheme defined above for the fixed hash H . Such a scheme is both binding according to Def. 10 and hiding according to Def. 9. In particular, for every PPT adversary $\mathcal{A}$, the advantages $\text{Adv}_{\text{Com}_H}^{\text{bnd}}(\mathcal{A}, \lambda)$ and $\text{Adv}_{\text{Com}_H}^{\text{hdn}}(\mathcal{A}, \lambda)$ are both negligible in λ .

D.4 Non-Interactive Proof Systems

Given a relation $R \subseteq \{0, 1\}^* \times \{0, 1\}^*$ for which membership can be tested efficiently, we call w a *witness* for a statement x if $(x, w) \in R$, and define the corresponding language $\mathcal{L}_R := \{x \mid \exists w : (x, w) \in R\}$. We focus on the notion of a non-interactive zero-knowledge (NIZK) system that is also knowledge sound. A NIZK system for a relation R is a tuple of PPT algorithms $\text{PS} = (\text{PS.Priv}, \text{PS.Vry})$, and works as follows:

- $\text{PS.Priv}(x, w) \rightarrow \pi$ on input a statement x and a witness w , outputs a proof π .
- $\text{PS.Vry}(x, \pi) \rightarrow 0/1$ on input a statement x and a proof π , outputs 1 if it accepts and 0 otherwise.

The scheme PS satisfies the correctness, zero-knowledge (ZK) and knowledge-soundness (KS) as defined below.

Definition 11 (Correctness). A NIZK argument system $\text{PS} = (\text{PS.Priv}, \text{PS.Vry})$ for a relation R is perfectly correct if for every $\lambda \in \mathbb{N}$ and $(x, w) \in R$ the following holds:

$$\Pr[\text{PS.Vry}(x, \pi) = 1 \mid \pi \leftarrow \text{PS.Priv}(x, w)] = 1.$$

Definition 12 (Online Knowledge-Soundness). Consider the game KS , described in Fig. 9. A NIZK system $\text{PS} = (\text{PS.Priv}, \text{PS.Vry})$ for a relation R satisfies the KS property if for every $\lambda \in \mathbb{N}$ and every PPT adversary $\mathcal{A}$ there exists a PPT extractor PS.Ext such that the advantage $\text{Adv}_{\text{PS}}^{\text{ks}}(\mathcal{A}, \lambda)$ is negligible in λ where: $\text{Adv}_{\text{PS}}^{\text{ks}}(\mathcal{A}, \lambda) := \Pr[\text{KS}_{\text{PS}}^{\mathcal{A}}(\lambda) = 1]$.

Game $\text{KS}_{\text{PS}}^{\mathcal{A}}(\lambda)$ —

1. The adversary $\mathcal{A}$, on input 1^λ , outputs a pair (x, π) .
2. The extractor PS.Ext , on input (x, π) outputs a value $w \in \{0, 1\}^*$.
3. The game outputs 1 if a) $\text{PS.Vry}(x, \pi) = 1$ and b) $(x, w) \notin R$. Otherwise, it outputs 0.

Fig. 9: The KS game.

Note that we keep the definition black-box in the setup, since the side information that the extractor uses depends on how the proof system is realized, and none of our proofs relies on its form. That is, in the common reference string model it is a trapdoor produced together with the string, while in the random oracle model it is the list of the queries that the prover made to the oracle.

Definition 13 (Zero-Knowledge). Consider the game ZK , described in Fig. 10. A NIZK system $\text{PS} = (\text{PS.Priv}, \text{PS.Vry})$ for a relation R satisfies the ZK property if for every PPT adversary $\mathcal{A}$ a) there exists a PPT simulator PS.Sim and b) the advantage $\text{Adv}_{\text{PS}}^{\text{zk}}(\mathcal{A}, \lambda)$ is negligible in λ where: $\text{Adv}_{\text{PS}}^{\text{zk}}(\mathcal{A}, \lambda) := \left| \frac{1}{2} - \Pr[\text{ZK}_{\text{PS}}^{\mathcal{A}}(\lambda) = 1] \right|$.

Game $\text{ZK}_{\text{PS}}^{\mathcal{A}}(\lambda)$ —

1. Sample a uniformly random bit $b \leftarrow \{0, 1\}$.
2. Define a proving oracle Prove as follows:

- On input a tuple $(\mathbf{x}, \mathbf{w})$, if $(\mathbf{x}, \mathbf{w}) \notin \mathbf{R}$ then the oracle return $\perp$.
 Compute $\pi_0 \leftarrow \text{PS.Priv}(\mathbf{x}, \mathbf{w})$ if $b = 0$, otherwise $\pi_1 \leftarrow \text{PS.Sim}(\mathbf{x})$, and output π_b .
3. The adversary $\mathcal{A}$ is given as input the security parameter 1^λ and oracle access to **Prove**. After interacting with the oracle, it outputs a bit b' . The game outputs 1 if $b = b'$, and 0 otherwise.

Fig. 10: The ZK game.

E Security Proofs

E.1 Proof of Thm. 1

Fix inputs as in Def. 1, so that $\text{P}(\text{pol}, \text{cert}, \mathbf{attrs}) = 1$ and $\phi_{\text{Iss}}(\mathbf{attrs}) = 1$. Two things have to be shown: that the honest client obtains $\text{cred} \neq \perp$, and that the honest verifier outputs $b = \phi_{\text{Vry}}(\mathbf{attrs})$. For the first, the tuple $\mathbf{w} = (\text{cert}, \mathbf{attrs}, \mathbf{c}_{\text{Vry}}, \mathbf{d}_{\text{Vry}}, \kappa)$ computed by $\text{CRAC}_{\text{RSA}}.\text{IClient}_0$ is a valid witness for $\mathbf{x} = (\text{vk}, t, \text{pol}, \phi_{\text{Iss}})$: condition (1) of $\text{R}_{\text{CRAC}_{\text{RSA}}}$ holds by the perfect correctness of Com_{Vry} , condition (2) holds because the client itself sets μ as the output of $\text{PSS.Enc}(m)$ and $t := \mu \kappa^e$ with $\kappa \leftarrow \mathbb{Z}_N^*$, and the Honest Branch holds by hypothesis. By the perfect correctness of **PS** the proof π verifies, so the issuer does not abort and replies with $t^d \bmod N$. Since (N, e, d) is output by RSAGen we have $ed \equiv 1 \pmod{\varphi(N)}$, whence $\sigma^e = t^{de} \kappa^{-e} = t \kappa^{-e} = \mu \pmod{N}$: the client's check succeeds and $\text{cred} = (\sigma, \mathbf{c}_{\text{Vry}}, \mathbf{d}_{\text{Vry}}) \neq \perp$.

For the second, if $\phi_{\text{Vry}}(\mathbf{attrs}) = 0$ then $\text{CRAC}_{\text{RSA}}.\text{VClient}_0$ aborts, the session does not complete, and $b = 0$ as required. Suppose then $\phi_{\text{Vry}}(\mathbf{attrs}) = 1$, and consider the checks of $\text{CRAC}_{\text{RSA}}.\text{Verifier}_1$. The message parses by construction, $\text{flds}(\phi_{\text{Vry}}) \neq \emptyset$ since every predicate of Φ_ℓ is defined on a non-empty set of fields, and the check relating ctx and aux succeeds by definition in the generic construction. The verifier recomputes the same m from $\mathbf{c}_{\text{Vry}}$, and $\sigma^e = \mu$ that is the output of the encoding $\text{PSS.Enc}(m)$, so $\text{RSA.Vry}(\text{vk}, m, \sigma) = 1$ by the perfect correctness of **RSA**. Each revealed value satisfies $\text{Com}_{\text{Vry}}.\text{Opn}(\mathbf{opn}_i, \mathbf{c}'_i, \mathbf{d}'_i) = 1$ by the perfect correctness of Com_{Vry} , since $\mathbf{opn}_i = \mathbf{a}_i$ and $(\mathbf{c}'_i, \mathbf{d}'_i) \leftarrow \text{Com}_{\text{Vry}}.\text{Cmt}(\mathbf{a}_i)$. Finally $\mathbf{opn}_i \in \mathcal{S}_i^{\phi_{\text{Vry}}}$ for every $i \in \text{flds}(\phi_{\text{Vry}})$, which is exactly $\phi_{\text{Vry}}(\mathbf{attrs}) = 1$. The verifier therefore returns $b = 1$.

E.2 Proof of Thm. 2

We proceed by hybrid arguments creating a sequence of indistinguishable games G_0, G_1, G_2 . Every game samples the bit b uniformly at random and has the same output rule as **ULK**, i.e., it outputs 1 iff the bit b' of $\mathcal{A}$ equals b (or a uniformly random bit whenever step 2 of **ULK** halts the game, as **ULK** does), and we write $p_i := \Pr[G_i = 1]$. Since G_0 is the real game, $\text{Adv}_{\text{CRAC}_{\text{RSA}}}^{\text{ulk}}(\mathcal{A}, \lambda) = |\frac{1}{2} - p_0|$. As in Fig. 1, $\text{id} \in \{0, 1\}$ is the session identifier and the values of session id carry the subscript id ; we extend this convention to the values computed by the client and by the issuer in that session, e.g., t_{id} , π_{id} and $\mathbf{c}_{\text{Vry}, \text{id}}$; in particular, we write $u_{\text{id}} := \text{pm}_{\mathcal{I}}^0$ for the issuer's response in session id , so that the client computes $\sigma_{\text{id}} = u_{\text{id}} \kappa_{\text{id}}^{-1}$ and accepts iff $\sigma_{\text{id}}^e \equiv \mu_{\text{id}}$. Moreover, $D := \{i \in [\ell] : \mathbf{a}_{0,i} \neq \mathbf{a}_{1,i}\}$ is the set of fields on which the two attribute vectors differ. Every modification below is applied to both sessions at once, and one reduction per game suffices, since the oracles of **ZK** and **HDN** answer a polynomial number of queries. Each reduction simulates the whole game for $\mathcal{A}$, including the sampling of b , and outputs 1 to its challenger iff $b' = b$; its own challenge bit then selects which of the two games being compared it simulates.

Two facts are used throughout: (a) the bit b is read only by the showing oracle, and only after all of its checks have passed; hence, in every game below, on the event that the showing does not take place (because $\mathcal{A}$ never queries the oracle, or because the oracle aborts, e.g., since some $\text{cred}_{\text{id}} = \perp$ or the checks on ϕ_{Vry} fail) the view of $\mathcal{A}$ is independent of b ; (b) the showing oracle aborts unless $\text{flds}(\phi_{\text{Vry}}) \cap D = \emptyset$, and the client sends openings only for $i \in \text{flds}(\phi_{\text{Vry}})$, so the openings $\mathbf{d}'_{\text{id},i}$ with $i \in D$ are never sent to $\mathcal{A}$.

Game G_0 . This is the real game $\text{ULK}_{\text{CRAC}_{\text{RSA}}}^{\mathcal{A}}(\lambda)$: both sessions run the honest client on the inputs $(\text{pol}_{\text{id}}, \phi_{\text{Iss}}, \text{cert}_{\text{id}}, \mathbf{attrs}_{\text{id}})$ chosen by $\mathcal{A}$, and the client proves the Honest Branch of $\text{R}_{\text{CRAC}_{\text{RSA}}}$ with the witness $\mathbf{w}_{\text{id}} = (\text{cert}_{\text{id}}, \mathbf{attrs}_{\text{id}}, \mathbf{c}_{\text{Vryid}}, \mathbf{d}_{\text{Vryid}}, \kappa_{\text{id}})$. The witness is valid because step 2 of ULK guarantees $\text{P}(\text{pol}_{\text{id}}, \text{cert}_{\text{id}}, \mathbf{attrs}_{\text{id}}) = 1$ and $\phi_{\text{Iss}}(\mathbf{attrs}_{\text{id}}) = 1$.

Game G_1 . In this game, we make the following change: in both issuance sessions, instead of creating the proof via $\text{PS.Priv}(\mathbf{x}_{\text{id}}, \mathbf{w}_{\text{id}})$, the client uses the simulator PS.Sim to simulate a proof for the statement $\mathbf{x}_{\text{id}}$. We show that this change is not efficiently noticeable by defining an adversary $\mathcal{A}_{\text{ZK}}$ that plays in game ZK against PS.

We describe the strategy of the adversary $\mathcal{A}_{\text{ZK}}$. $\mathcal{A}_{\text{ZK}}$, on input 1^λ , simulates the game G_0 for $\mathcal{A}$: it runs $\mathcal{A}$ internally, samples the bit b itself and plays both oracles of Fig. 1, using its oracle Prove (according to Fig. 10) on $(\mathbf{x}_{\text{id}}, \mathbf{w}_{\text{id}})$ to obtain the proof π_{id} required in session id , while the rest of the message $(t_{\text{id}}, \pi_{\text{id}})$, the unblinding and the showing are computed by $\mathcal{A}_{\text{ZK}}$ as in the real game. Finally, when $\mathcal{A}$ outputs its bit b' , $\mathcal{A}_{\text{ZK}}$ outputs 1 to its challenger if $b' = b$, and 0 otherwise. By the definition of $\mathcal{A}_{\text{ZK}}$, when the bit of the ZK game is equal to 0, every query is answered with $\text{PS.Priv}(\mathbf{x}_{\text{id}}, \mathbf{w}_{\text{id}})$, exactly as computed by the honest client, so $\mathcal{A}_{\text{ZK}}$ perfectly simulates the real game G_0 ; when it is equal to 1, every query is answered with $\text{PS.Sim}(\mathbf{x}_{\text{id}})$, so $\mathcal{A}_{\text{ZK}}$ perfectly simulates G_1 . Thus, we have that $\Pr[\text{ZK}_{\text{PS}}^{\mathcal{A}_{\text{ZK}}}(\lambda) = 1] = \frac{1}{2}(1 - p_0) + \frac{1}{2}p_1$, and therefore:

$$|p_1 - p_0| \leq 2 \text{Adv}_{\text{PS}}^{\text{ZK}}(\mathcal{A}_{\text{ZK}}, \lambda). \quad (1)$$

Game G_2 . In this game, we make the following change: in both sessions, for every $i \in D$, instead of committing to $\mathbf{a}_{\text{id},i}$, the client computes $(\mathbf{c}'_{\text{id},i}, \mathbf{d}'_{\text{id},i}) \leftarrow \text{Com}_{\text{Vry}}.\text{Cmt}(0)$, i.e., a commitment to a fixed dummy value 0; the values $m_{\text{id}}, \mu_{\text{id}}, t_{\text{id}}$ and $\mathbf{x}_{\text{id}}$ are computed from the resulting $\mathbf{c}_{\text{Vryid}}$ as before, that is, π_{id} is simulated on the new $\mathbf{x}_{\text{id}}$, and the showing uses the resulting $(\mathbf{c}_{\text{Vryid}}, \mathbf{d}_{\text{Vryid}})$. We show that this change is not efficiently noticeable by defining an adversary $\mathcal{A}_{\text{HDN}}$ that plays in game HDN against Com_{Vry} .

We describe the strategy of the adversary $\mathcal{A}_{\text{HDN}}$. $\mathcal{A}_{\text{HDN}}$, on input 1^λ , simulates the game G_1 for $\mathcal{A}$: it runs $\mathcal{A}$ internally and samples the bit b itself; for every session id and every $i \in D$, it obtains the commitment $\mathbf{c}'_{\text{id},i}$ by querying its oracle Cmt of Fig. 7 on the pair $(m_0, m_1) := (\mathbf{a}_{\text{id},i}, 0)$, i.e., on the value committed in G_1 and on the value committed in G_2 . Everything else is computed by $\mathcal{A}_{\text{HDN}}$ as in G_1 : specifically, it computes the commitments $\mathbf{c}'_{\text{id},i}$ with $i \notin D$ itself, samples κ_{id} , computes $m_{\text{id}}, \mu_{\text{id}}$ and t_{id} from the commitments, simulates π_{id} via $\text{PS.Sim}(\mathbf{x}_{\text{id}})$, and plays the showing oracle on $(\mathbf{attrs}_b, \text{cred}_b)$. Note that Cmt returns the commitment $\mathbf{c}'_{\text{id},i}$ but not its opening $\mathbf{d}'_{\text{id},i}$, so $\mathcal{A}_{\text{HDN}}$ must run the game for $\mathcal{A}$ without the openings $\mathbf{d}'_{\text{id},i}$ with $i \in D$. This is possible because none of the values it has to compute depends on them: (i) t_{id} is computed from $m_{\text{id}} = \mathbf{c}'_{\text{id},1} \parallel \dots \parallel \mathbf{c}'_{\text{id},\ell}$, hence from the commitments only; (ii) π_{id} would require the openings as part of the witness, but in G_1 it is simulated from $\mathbf{x}_{\text{id}}$ alone; (iii) the showing sends the openings $\mathbf{d}'_{\text{id},i}$ only for $i \in \text{flds}(\phi_{\text{Vry}})$, and $\text{flds}(\phi_{\text{Vry}}) \cap D = \emptyset$, so every opening sent by the showing is one that $\mathcal{A}_{\text{HDN}}$ computed itself. Finally, when $\mathcal{A}$ outputs its bit b' , $\mathcal{A}_{\text{HDN}}$ outputs 1 to its challenger if $b' = b$, and 0 otherwise. By the definition of $\mathcal{A}_{\text{HDN}}$, when the challenge bit of HDN is 0, so that each query $(\mathbf{a}_{\text{id},i}, 0)$ with $i \in D$ is answered with a fresh commitment to $\mathbf{a}_{\text{id},i}$, exactly as

computed by the honest client, $\mathcal{A}_{\text{HDN}}$ perfectly simulates G_1 ; when it is 1, so that each such query is answered with a fresh commitment to 0, $\mathcal{A}_{\text{HDN}}$ perfectly simulates G_2 . Thus, as above, we have that $\Pr\left[\text{HDN}_{\text{Com}_{\text{vry}}}^{\mathcal{A}_{\text{HDN}}}(\lambda) = 1\right] = \frac{1}{2}(1 - p_1) + \frac{1}{2}p_2$, and therefore:

$$|p_2 - p_1| \leq 2 \text{Adv}_{\text{Com}_{\text{vry}}}^{\text{hdn}}(\mathcal{A}_{\text{HDN}}, \lambda). \quad (2)$$

After G_2 , the tuples $(\mathbf{c}_{\text{vry}_{\text{id}}}, \mathbf{d}_{\text{vry}_{\text{id}}}, \mu_{\text{id}})_{\text{id} \in \{0,1\}}$ are i.i.d., namely on $i \notin D$ they commit to the same values, on $i \in D$ to 0.

INDEPENDENCE OF G_2 FROM THE BIT b . For this last step, we follow the same argument used in [Lys23] to prove that “plain” blind RSA (i.e., RSA-BSSA, without the NIZK proof and the commitments) satisfies the blindness property, and we show that the view of $\mathcal{A}$ in G_2 is independent of b . Throughout, $\mu_{\text{id}} \in \mathbb{Z}_N^*$ in every session that does not abort, since the client aborts otherwise (as described in $\text{CRAC}_{\text{RSA}}.\text{IClient}_0$), exactly as in the blinding step of RSA-BSSA [DJW23]; a session that aborts outputs $\perp$, and then the showing does not take place, and the view of $\mathcal{A}$ is trivially independent of b .

We analyze two cases based on key generation. When the public key is honestly generated by the adversary, i.e., e is coprime to $\varphi(N)$ as for every key output by RSAGen , the use of a blinding factor $\kappa_{\text{id}} \leftarrow \mathbb{Z}_N^*$ ensures that the pair $(\mu_{\text{id}}, \sigma_{\text{id}})$ is statistically independent of the values t_{id} and u_{id} that are observed and chosen by the issuer: $t_{\text{id}} = \mu_{\text{id}} \kappa_{\text{id}}^e$ is uniform in $\mathbb{Z}_N^*$ and independent of μ_{id} , the response u_{id} is chosen by $\mathcal{A}$ as a function of its view so far, which reads neither b nor the tuples other than through t_0, t_1 , and, whenever the client accepts, σ_{id} is the unique e -th root of μ_{id} . Since we have already established that the simulated proof π_{id} provides no additional information (game G_1) and that the tuples $(\mathbf{c}_{\text{vry}_{\text{id}}}, \mathbf{d}_{\text{vry}_{\text{id}}}, \sigma_{\text{id}})_{\text{id} \in \{0,1\}}$ are i.i.d. (note after G_2), this statistical independence implies unlinkability: the showing message is a fixed function of the tuple $(\mathbf{c}_{\text{vry}_b}, \mathbf{d}_{\text{vry}_b}, \sigma_b)$ and of values that do not depend on b , so the view of $\mathcal{A}$ is identically distributed for $b = 0$ and $b = 1$; and if the showing does not take place, the view is independent of b by fact (a).

As noted in [Lys23], the adversary might also generate the public key maliciously (recall that in the ULK game in Fig. 1, $\mathbf{vk}$ is chosen by the adversary), choosing e not coprime to $\varphi(N)$ so that $\kappa \mapsto \kappa^e$ is not a permutation of $\mathbb{Z}_N^*$ and t_{id} reveals the residuosity class of μ_{id} , i.e., its class modulo the subgroup of e -th residues. Lysyanskaya shows that for any maliciously chosen RSA modulus N and public exponent e , security depends on the PSSGen function. She outlines two approaches for protection: (1) requiring proofs that the signer’s public key is honestly generated, as in [GRSB19], or (2) ensuring that the input message contains high-entropy components unknown to the signer before key generation. Our construction follows the latter approach, similar to the RSA-BSSA blindness security proof of [Lys23], by providing randomness in the commitments that form m_{id} . This ensures that even with maliciously generated keys, t_{id} reveals nothing about the tuple $(\mathbf{c}_{\text{vry}_{\text{id}}}, \mathbf{d}_{\text{vry}_{\text{id}}}, \sigma_{\text{id}})$ beyond the residuosity class of μ_{id} , which is trivial whenever the client accepts the issuer’s response: $\sigma_{\text{id}}^e \equiv \mu_{\text{id}}$ forces μ_{id} to be an e -th residue, and then t_{id} is uniform among the e -th residues and σ_{id} is uniform among the e -th roots of μ_{id} given $(\mu_{\text{id}}, t_{\text{id}}, u_{\text{id}})$ [Lys23, Lemmas 5–7] (which only use that $\kappa \mapsto \kappa^e$ is a homomorphism of $\mathbb{Z}_N^*$). Thus it is not possible to extract any information about the bit b from the input t_{id} , just as in the case of honestly generated public keys. Since the view of $\mathcal{A}$, and hence its output b' , is independent of b , we have $\Pr[b' = b] = \frac{1}{2}$, i.e., $p_2 = \frac{1}{2}$.

Combining this equality with Eqs. (1) and (2) completes the proof of the theorem.

E.3 Proof of Thm. 3

We first describe the simulator Sim , then show through a sequence of games that no PPT distinguisher D tells the real world from the ideal one.

THE SIMULATOR. Sim runs on input $(1^\lambda, \text{vk})$ with $\text{vk} = (N, e, \text{PSS})$. When the oracle hands it $(\text{id}, \text{pol}, \phi_{\text{Iss}})$, Sim opens session id and proceeds as follows.

1. Set $\text{cert} := \perp$ and $\mathbf{a}_i := \perp$ for every $i \in [\ell]$.
2. $\forall i \in [\ell]$, compute $(\mathbf{c}'_i, \mathbf{d}'_i) \leftarrow \text{Com}_{\text{vry}}.\text{Cmt}(\perp)$; set $\mathbf{c}_{\text{vry}} := (\mathbf{c}'_i)_{i \in [\ell]}$, $\mathbf{d}_{\text{vry}} := (\mathbf{d}'_i)_{i \in [\ell]}$ and $m := \mathbf{c}'_1 \parallel \dots \parallel \mathbf{c}'_\ell$.
3. Compute $\mu := \text{PSS}.\text{Enc}(m)$.
4. Sample $\kappa \leftarrow \mathbb{Z}_N^*$ and compute $t := \mu \kappa^e \pmod{N}$.
5. Set $\mathbf{x} := (\text{vk}, t, \text{pol}, \phi_{\text{Iss}})$ and $\mathbf{w} := (\perp, (\perp)_{i \in [\ell]}, \mathbf{c}_{\text{vry}}, \mathbf{d}_{\text{vry}}, r, \kappa)$, and then compute the proof $\pi \leftarrow \text{PS}.\text{Prv}(\mathbf{x}, \mathbf{w})$.¹⁸
6. Send (t, π) to $\mathcal{A}$ in session id and output the output of $\mathcal{A}$.

Steps 1–6 are exactly $\text{CRAC}_{\text{RSA}}.\text{IClient}_0(\text{vk}, \text{pol}, \phi_{\text{Iss}}, \perp, (\perp)_{i \in [\ell]})$: the simulator is the honest client on dummy inputs, and never invokes $\text{PS}.\text{Sim}$ nor uses any trapdoor. It ignores the reply of $\mathcal{A}$ because CR discards the output of the client, so neither the unblinding nor the check $\sigma^e \equiv \mu$ needs to be emulated.

SETUP FOR THE REDUCTION. Every game outputs the bit of D , and we write $p_i := \Pr[\text{G}_i = 1]$. As in Fig. 2, id is the session identifier, and the values of session id carry the subscript id ; every modification below is applied to every session opened by D , which may be interleaved. Since D is PPT , it opens polynomially many sessions, but no bound on their number enters the analysis: the oracles of ZK and HDN answer any number of queries, so one reduction per game handles all sessions at once.

Game G_0 . This is the real world $\text{CR}_{\text{CRAC}_{\text{RSA}}}^{\mathcal{A}, \text{Sim}, \text{D}, 0}(\lambda)$: every session runs the honest client on the inputs $(\text{cert}_{\text{id}}, \mathbf{attrs}_{\text{id}})$ chosen by D , and the client proves the Honest Branch of $\text{R}_{\text{CRAC}_{\text{RSA}}}$ with the witness $\mathbf{w}_{\text{id}} = (\text{cert}_{\text{id}}, \mathbf{attrs}_{\text{id}}, \mathbf{c}_{\text{vry}_{\text{id}}}, \mathbf{d}_{\text{vry}_{\text{id}}}, \kappa_{\text{id}})$. The witness is valid because the oracle has checked $\text{P}(\text{pol}_{\text{id}}, \text{cert}_{\text{id}}, \mathbf{attrs}_{\text{id}}) = 1$ and $\phi_{\text{Iss}_{\text{id}}}(\mathbf{attrs}_{\text{id}}) = 1$ before opening the session.

Game G_1 . In this game, we make the following change: in every session, instead of creating the proof via $\text{PS}.\text{Prv}(\mathbf{x}_{\text{id}}, \mathbf{w}_{\text{id}})$, the client uses the simulator $\text{PS}.\text{Sim}$ to simulate a proof for the statement $\mathbf{x}_{\text{id}}$. We show that this change is not efficiently noticeable by defining an adversary $\mathcal{A}_{\text{ZK}, 1}$ that plays in game ZK against PS .

We describe the strategy of the adversary $\mathcal{A}_{\text{ZK}, 1}$. $\mathcal{A}_{\text{ZK}, 1}$, on input 1^λ , simulates the game G_0 for $(\mathcal{A}, \text{D})$: it runs D and $\mathcal{A}$ internally and plays the issuance oracle of Fig. 2; for every session id , it obtains the proof π_{id} by querying its oracle Prove of Fig. 10 on $(\mathbf{x}_{\text{id}}, \mathbf{w}_{\text{id}})$, i.e., on the statement and on the Honest-Branch witness of the honest client, while the rest of the message $(t_{\text{id}}, \pi_{\text{id}})$ is computed by $\mathcal{A}_{\text{ZK}, 1}$ as in the real game. Finally, when D outputs its bit b' , $\mathcal{A}_{\text{ZK}, 1}$ outputs the same bit to its challenger. By the definition of $\mathcal{A}_{\text{ZK}, 1}$, when the bit of the ZK game is equal to 0, every query is answered with $\text{PS}.\text{Prv}(\mathbf{x}_{\text{id}}, \mathbf{w}_{\text{id}})$, exactly as computed by the honest client, so $\mathcal{A}_{\text{ZK}, 1}$ perfectly simulates the real game G_0 ; when it is equal to 1, every query is answered with $\text{PS}.\text{Sim}(\mathbf{x}_{\text{id}})$, so $\mathcal{A}_{\text{ZK}, 1}$ perfectly simulates G_1 . Thus, we have that $\Pr[\text{ZK}_{\text{PS}}^{\mathcal{A}_{\text{ZK}, 1}}(\lambda) = 1] = \frac{1}{2}(1 - p_0) + \frac{1}{2} p_1$, and therefore:

$$|p_1 - p_0| \leq 2 \text{Adv}_{\text{PS}}^{\text{zk}}(\mathcal{A}_{\text{ZK}, 1}, \lambda). \quad (3)$$

After G_1 , no value seen by $\mathcal{A}$ depends on the openings $\mathbf{d}_{\text{vry}_{\text{id}}}$: the commitment $\mathbf{c}_{\text{vry}_{\text{id}}}$ is never sent in the clear, and π_{id} is computed from $\mathbf{x}_{\text{id}}$ alone.

¹⁸Note that, the witness is valid: the common conditions of $\text{R}_{\text{CRAC}_{\text{RSA}}}$ hold because Sim holds all the openings and computed t honestly, and the Trapdoor Branch holds because every attribute equals $\perp$. The Honest Branch is not needed.

Game G₂. In this game, we make the following change: in every session, instead of committing to $\mathbf{a}_{\text{id},i}$ for every $i \in [\ell]$, the client computes $(\mathbf{c}'_{\text{id},i}, \mathbf{d}'_{\text{id},i}) \leftarrow \text{Com}_{\text{vry}}.\text{Cmt}(\perp)$ for every $i \in [\ell]$; the values m_{id} , μ_{id} , t_{id} and $\mathbf{x}_{\text{id}}$ are computed from the resulting $\mathbf{c}_{\text{vry},\text{id}}$ as before, and π_{id} is simulated on the new $\mathbf{x}_{\text{id}}$. We show that this change is not efficiently noticeable by defining an adversary $\mathcal{A}_{\text{HDN}}$ that plays in game HDN against Com_{vry} .

We describe the strategy of the adversary $\mathcal{A}_{\text{HDN}}$. $\mathcal{A}_{\text{HDN}}$, on input 1^λ , simulates the game G₁ for $(\mathcal{A}, \text{D})$: for every session id and every $i \in [\ell]$, it obtains the commitment $\mathbf{c}'_{\text{id},i}$ by querying its oracle Cmt of Fig. 7 on the pair $(m_0, m_1) := (\mathbf{a}_{\text{id},i}, \perp)$, i.e., on the value committed by the honest client in G₁ and on the value committed in G₂. Everything else is computed by $\mathcal{A}_{\text{HDN}}$ as in G₁: specifically, it samples κ_{id} , computes m_{id} , μ_{id} and t_{id} from the commitments received, and simulates π_{id} via $\text{PS.Sim}(\mathbf{x}_{\text{id}})$. Note that Cmt returns the commitments $\mathbf{c}'_{\text{id},i}$ but not their openings $\mathbf{d}'_{\text{id},i}$, so $\mathcal{A}_{\text{HDN}}$ must run the game for $(\mathcal{A}, \text{D})$ without them; this is possible because t_{id} is a function of the commitments only, and π_{id} is simulated. Finally, when D outputs its bit b' , $\mathcal{A}_{\text{HDN}}$ outputs the same bit to its challenger. By the definition of $\mathcal{A}_{\text{HDN}}$, when the challenge bit of HDN is 0, every query is answered with a fresh commitment to $\mathbf{a}_{\text{id},i}$, exactly as computed by the honest client, so $\mathcal{A}_{\text{HDN}}$ perfectly simulates G₁; when it is 1, every query is answered with a fresh commitment to $\perp$, so $\mathcal{A}_{\text{HDN}}$ perfectly simulates G₂. Thus, as above, $\Pr[\text{HDN}_{\text{Com}_{\text{vry}}}^{\mathcal{A}_{\text{HDN}}}(\lambda) = 1] = \frac{1}{2}(1 - p_1) + \frac{1}{2}p_2$, and therefore:

$$|p_2 - p_1| \leq 2 \text{Adv}_{\text{Com}_{\text{vry}}}^{\text{hdn}}(\mathcal{A}_{\text{HDN}}, \lambda). \quad (4)$$

This is the only place where t is handled: since t_{id} is deterministically computed from $(\mathbf{c}_{\text{vry},\text{id}}, \kappa_{\text{id}})$, indistinguishability of the commitments carries over to t_{id} with no further argument, whatever the key (N, e) chosen by D . After G₂, the message $(t_{\text{id}}, \pi_{\text{id}})$ is computed without reading $(\text{cert}_{\text{id}}, \text{attrs}_{\text{id}})$.

Game G₃. In this game, we make the following change: in every session, instead of simulating the proof via $\text{PS.Sim}(\mathbf{x}_{\text{id}})$, the client creates it via

$$\text{PS.Priv}(\mathbf{x}_{\text{id}}, \mathbf{w}_{\text{id}}^\perp)$$

with $\mathbf{w}_{\text{id}}^\perp := (\perp, (\perp)_{i \in [\ell]}, \mathbf{c}_{\text{vry},\text{id}}, \mathbf{d}_{\text{vry},\text{id}}, \kappa_{\text{id}})$, i.e., it produces π_{id} from $\text{cert} = \perp$ and the all- $\perp$ attribute vector. We show that this change is not efficiently noticeable by defining an adversary $\mathcal{A}_{\text{ZK},2}$ that plays in game ZK against PS .

We describe the strategy of the adversary $\mathcal{A}_{\text{ZK},2}$. $\mathcal{A}_{\text{ZK},2}$, on input 1^λ , simulates the game G₂ for $(\mathcal{A}, \text{D})$: for every session id , it obtains the proof π_{id} by querying its oracle Prove on $(\mathbf{x}_{\text{id}}, \mathbf{w}_{\text{id}}^\perp)$, i.e., on the statement and on the Trapdoor-Branch witness, while the rest of the message $(t_{\text{id}}, \pi_{\text{id}})$ is computed by $\mathcal{A}_{\text{ZK},2}$ as in G₂. Note that $\mathcal{A}_{\text{ZK},2}$ knows the witness $\mathbf{w}_{\text{id}}^\perp$, since in G₂ it generates every commitment itself and hence holds every opening; the witness is valid, as noted in the description of Sim . Finally, when D outputs its bit b' , $\mathcal{A}_{\text{ZK},2}$ outputs the same bit to its challenger. By the definition of $\mathcal{A}_{\text{ZK},2}$, when the bit of the ZK game is equal to 0, every query is answered with $\text{PS.Priv}(\mathbf{x}_{\text{id}}, \mathbf{w}_{\text{id}}^\perp)$, exactly as computed by Sim , so $\mathcal{A}_{\text{ZK},2}$ perfectly simulates G₃; when it is equal to 1, every query is answered with $\text{PS.Sim}(\mathbf{x}_{\text{id}})$, so $\mathcal{A}_{\text{ZK},2}$ perfectly simulates G₂. Thus, as above, $\Pr[\text{ZK}_{\text{PS}}^{\mathcal{A}_{\text{ZK},2}}(\lambda) = 1] = \frac{1}{2}(1 - p_3) + \frac{1}{2}p_2$, and therefore:

$$|p_3 - p_2| \leq 2 \text{Adv}_{\text{PS}}^{\text{zk}}(\mathcal{A}_{\text{ZK},2}, \lambda). \quad (5)$$

In G₃ the message of session id is computed by running $\text{CRAC}_{\text{RSA}}.\text{IClient}_0(\text{vk}, \text{pol}_{\text{id}}, \phi_{\text{Issid}}, \perp, (\perp)_{i \in [\ell]})$, and the reply of $\mathcal{A}$ is not used, since the output of the client is discarded; everything else, namely the messages exchanged between D and $\mathcal{A}$ and the replies of $\mathcal{A}$, is unchanged. This is precisely the ideal world with the simulator above, i.e., $\Pr[G_3 = 1] = \Pr[\text{CR}_{\text{CRAC}_{\text{RSA}}}^{\mathcal{A}, \text{Sim}, \text{D}, 1}(\lambda) = 1]$.

Combining this equality with Eqs. (3) to (5) completes the proof of the theorem.

E.4 Proof of Thm. 4

We first describe the extractor **Ext**, then show through a sequence of games that the advantage of every PPT adversary $\mathcal{A}$ is bounded by the advantages of reductions against the building blocks.

SETUP FOR THE REDUCTION. Since $\mathcal{A}$ is PPT, the number of issuance sessions it opens is bounded by a polynomial q_I , and we index these sessions by $\text{sid} \in [q_I]$ in the order in which they are opened. In session sid the adversary sends $\text{pm}_{\mathcal{C},\text{sid}}^0 = (t_{\text{sid}}, \pi_{\text{sid}})$, the issuer verifies π_{sid} on the statement $\mathbb{x}_{\text{sid}} = (\text{vk}, t_{\text{sid}}, \text{pol}_{\text{sid}}, \phi_{\text{Isssid}})$ and, if it accepts, replies with $u_{\text{sid}} := t_{\text{sid}}^d \pmod{N}$ and we write $\mathcal{J}$ for the set of sessions in which π_{sid} was accepted, and w_{sid} for the witness that the issuance oracle extracted in such a session. Similarly, the number of clients that $\mathcal{A}$ creates through the honest-issuance oracle is bounded by a polynomial q_C . For an honest client cid we write $\text{cred}_{\text{cid}} = (\sigma_{\text{cid}}, \mathbf{c}_{\text{Vrycid}}, \mathbf{d}_{\text{Vrycid}})$ with $\mathbf{c}_{\text{Vrycid}} = (c'_{\text{cid},i})_{i \in [\ell]}$ and $\mathbf{d}_{\text{Vrycid}} = (d'_{\text{cid},i})_{i \in [\ell]}$, m_{cid} for the message formed by $\mathbf{c}_{\text{Vrycid}}$, and $\text{attrs}_{\text{cid}} = (a_{\text{cid},i})_{i \in [\ell]}$ for its attributes, as in Fig. 3. In an accepted verification session the adversary sends $(\mathbf{c}_{\text{Vry}}^*, \sigma^*, \text{opn}^*, (d_i^*)_{i \in \text{flds}(\phi_{\text{Vry}}^*)}, \text{aux}^*)$, and we write $\mathbf{c}_{\text{Vry}}^* = (c_i^*)_{i \in [\ell]}$ and m^* for the message formed by $\mathbf{c}_{\text{Vry}}^*$; recall that m determines $\mathbf{c}_{\text{Vry}}$, since commitments have a fixed length.

According to the definition, in every following game, the output is the bit **win** computed as in Fig. 3, except that some games, before computing **win**, check for a specific event and output 0 when it occurs; we write $p_i := \Pr[G_i = 1]$, so that $\text{Adv}_{\text{CRAC}_{\text{RSA}}}^{\text{KNW}}(\mathcal{A}, \text{Ext}, \lambda) = p_0$. Moreover, each game outputs 0 on every run on which the previous one does. That is, we will write E_i for the (additional) event checked by G_i , we therefore have $p_{i-1} - p_i = \Pr[G_{i-1} = 1 \wedge E_i] \leq \Pr[E_i]$, and it suffices to bound $\Pr[E_i]$ for every i .

THE EXTRACTOR. The extractor **Ext** is the following algorithm. On input $(\text{vk}, \text{pol}, \phi_{\text{Iss}})$ and the transcript tr of a completed issuance session, it parses tr to recover the message (t, π) sent by $\mathcal{A}$, sets $\mathbb{x} := (\text{vk}, t, \text{pol}, \phi_{\text{Iss}})$ and runs $\text{w} \leftarrow \text{PS.Ext}(\mathbb{x}, \pi)$, where **PS.Ext** is the extractor of **PS** given by Def. 12. It then parses $\text{w} = (\text{cert}, \text{attrs}, \mathbf{c}_{\text{Vry}}, \mathbf{d}_{\text{Vry}}, \kappa)$ and outputs $(\text{cert}, \text{attrs})$; if w cannot be parsed, it outputs $(\perp, (\perp)_{i \in [\ell]})$. The extractor is PPT since **PS.Ext** is PPT.

Game G_0 . This is the real game $\text{KNW}_{\text{CRAC}_{\text{RSA}}}^{\mathcal{A}, \text{Ext}}(\lambda)$ with the extractor **Ext** above.

Game G_1 . In this game, when $\mathcal{A}$ halts and before computing **win**, the game checks the event $(E_1 : \iff \exists \text{sid} \in \mathcal{J} : (\mathbb{x}_{\text{sid}}, \text{w}_{\text{sid}}) \notin \text{R}_{\text{CRAC}_{\text{RSA}}})$, i.e., some completed issuance session whose extracted value is not a witness for its statement, and outputs 0 if it occurs. We bound $\Pr[E_1]$ through the game **KS** against **PS**.

Fix $j \in [q_I]$ and let $\mathcal{A}_{\text{KS},j}$ be the PPT adversary that runs the experiment G_0 by itself and outputs the pair $(\mathbb{x}_j, \pi_j)$ of session j when $j \in \mathcal{J}$, and $(\perp, \perp)$ otherwise. The value $\text{PS.Ext}(\mathbb{x}_j, \pi_j)$ is then the witness w_j that **Ext** uses in that session. Since $\text{PS.Vry}(\mathbb{x}_j, \pi_j) = 1$ precisely when $j \in \mathcal{J}$, the event $(E_1^j : \iff j \in \mathcal{J} \wedge (\mathbb{x}_j, \text{w}_j) \notin \text{R}_{\text{CRAC}_{\text{RSA}}})$ is exactly the event $\text{KS}_{\text{PS}}^{\mathcal{A}_{\text{KS},j}}(\lambda) = 1$. As $E_1 = \bigcup_{j \in [q_I]} E_1^j$, a union bound gives:

$$\Pr[E_1] \leq \sum_{j=1}^{q_I} \text{Adv}_{\text{PS}}^{\text{KS}}(\mathcal{A}_{\text{KS},j}, \lambda). \quad (6)$$

After G_1 , for every $\text{sid} \in \mathcal{J}$ the witness w_{sid} satisfies $\text{R}_{\text{CRAC}_{\text{RSA}}}$. In particular:

- (i) for every $i \in [\ell]$ it holds that $\text{Com}_{\text{Vry}}.\text{Opn}(a_{\text{sid},i}, c'_{\text{sid},i}, d'_{\text{sid},i}) = 1$, where $\mathbf{c}_{\text{Vrysid}} = (c'_{\text{sid},i})_{i \in [\ell]}$ and $\mathbf{d}_{\text{Vrysid}} = (d'_{\text{sid},i})_{i \in [\ell]}$;

- (ii) the blinded value t_{sid} is well formed: letting m_{sid} be the message formed by $\mathbf{cv}_{\text{ry}_{\text{sid}}}$ and $\mu_{\text{sid}} := \text{PSS.Enc}(m_{\text{sid}})$ its PSS encoding, it holds that $t_{\text{sid}} = \mu_{\text{sid}} \kappa_{\text{sid}}^e \pmod{N}$. Hence the issuer's response $u_{\text{sid}} = t_{\text{sid}}^d$ satisfies $u_{\text{sid}} = \mu_{\text{sid}}^d \kappa_{\text{sid}}^{ed} = \sigma_{\text{sid}} \kappa_{\text{sid}} \pmod{N}$, where $\sigma_{\text{sid}} := \mu_{\text{sid}}^d = \text{RSA.Sig}(\text{ik}, m_{\text{sid}})$ is the RSA-PSS signature on m_{sid} . In other words, in session sid the issuer has signed m_{sid} , and $\mathcal{A}$ receives that signature blinded by κ_{sid} . Consequently, the messages on which a signature has been issued during the game are exactly those in $M := \{m_{\text{sid}} : \text{sid} \in \mathcal{J}\} \cup \{m_{\text{cid}} : \text{cid} \in C\}$, where C is the set of clients created by the honest-issuance oracle;
- (iii) one of the two branches of $\text{RCRAC}_{\text{RSA}}$ holds: either the Honest Branch, i.e., $\text{P}(\text{pol}_{\text{sid}}, \text{cert}_{\text{sid}}, \mathbf{attrs}_{\text{sid}}) = 1$ and $\phi_{\text{Isssid}}(\mathbf{attrs}_{\text{sid}}) = 1$, or the Trapdoor Branch, i.e., $\mathbf{a}_{\text{sid},i} = \perp$ for every $i \in [\ell]$. Recall that the issuance oracle of Fig. 3 sets win only for a session whose extracted attributes are not legitimate *and* contain some $\mathbf{a}_{\text{sid},i} \neq \perp$. Such a session does not satisfy the Trapdoor Branch, hence it satisfies the Honest Branch, and its attributes are legitimate. Consequently, in G_1 the issuance oracle never sets win , every extracted vector is added to $\mathcal{Q}$, so that $\mathcal{Q} = \{\mathbf{attrs}_{\text{sid}} : \text{sid} \in \mathcal{J}\}$, and win can be set only in the final step;
- (iv) the secret key d is used by the game in two places only, and in both it can be replaced by a signing oracle for RSA, i.e., an oracle that on input a message m returns $\text{PSS.Enc}(m)^d$. The first place is the reply $u_{\text{sid}} = t_{\text{sid}}^d$ of the issuer. Since t_{sid} is chosen by $\mathcal{A}$, it is a priori an arbitrary element of $\mathbb{Z}_N^*$, and a signing oracle does not compute e -th roots of arbitrary elements; but by (ii) it holds that $t_{\text{sid}} = \text{PSS.Enc}(m_{\text{sid}}) \kappa_{\text{sid}}^e$, hence $t_{\text{sid}}^d = \text{PSS.Enc}(m_{\text{sid}})^d \cdot \kappa_{\text{sid}} = \text{RSA.Sig}(\text{ik}, m_{\text{sid}}) \cdot \kappa_{\text{sid}} \pmod{N}$, where the first factor is the signature on m_{sid} and the second is known from $\mathbf{w}_{\text{sid}}$. The second place is the signature σ_{cid} stored in cred_{cid} of an honest client cid , which equals $\text{RSA.Sig}(\text{ik}, m_{\text{cid}})$, while the rest of that execution is internal to the oracle and never seen by $\mathcal{A}$. In other words, the two operations of the game that involve d are a signature on m_{sid} , multiplied by κ_{sid} , and a signature on m_{cid} .

Game G_2 . In this game, when $\mathcal{A}$ halts and before computing win , the game checks the following event $(E_2 : \iff \exists (\phi_{\text{Vry}}^*, \mathbf{opn}^*) \in \mathcal{W} : m^* \notin M)$. Namely, the honest verifier accepted a showing built on a commitment vector that was never signed, and outputs 0 if it occurs. We bound $\Pr[E_2]$ by constructing an adversary $\mathcal{A}_{\text{EUF-CMA}}$ that plays in game EUF-CMA against RSA.

We describe the strategy of $\mathcal{A}_{\text{EUF-CMA}}$. By the definition of EUF-CMA, it receives as challenge input a verification key $\mathbf{vk} = (N, e, \text{PSS})$ and has access to the signing oracle **Sign** of Fig. 5. It embeds the challenge key into the game, running $\mathcal{A}$ on input $(1^\lambda, \mathbf{vk})$ and playing for it the four oracles of Fig. 3, as follows.

- The honest-verification and the verification oracles read only $\mathbf{vk}$, so $\mathcal{A}_{\text{EUF-CMA}}$ runs them exactly as in G_1 .
- In an issuance session sid , $\mathcal{A}_{\text{EUF-CMA}}$ acts as in G_1 except for the reply of the issuer, which it cannot compute, since it does not hold d . Upon accepting π_{sid} , it runs $\mathbf{w}_{\text{sid}} \leftarrow \text{PS.Ext}(\mathbf{x}_{\text{sid}}, \pi_{\text{sid}})$, as the issuance oracle does through **Ext**; if $(\mathbf{x}_{\text{sid}}, \mathbf{w}_{\text{sid}}) \notin \text{RCRAC}_{\text{RSA}}$ it halts with output $\perp$, which is the event E_1 on which G_1 already outputs 0. Otherwise it reads m_{sid} and κ_{sid} from $\mathbf{w}_{\text{sid}}$, queries **Sign** on m_{sid} , obtaining σ_{sid} , and replies with $u_{\text{sid}} := \sigma_{\text{sid}} \kappa_{\text{sid}} \pmod{N}$. By (iv) in G_1 this is exactly t_{sid}^d , which is what $\mathcal{A}$ expects.
- In the honest-issuance oracle for a client cid , $\mathcal{A}_{\text{EUF-CMA}}$ queries **Sign** on m_{cid} , obtaining σ_{cid} , and stores $\text{cred}_{\text{cid}} := (\sigma_{\text{cid}}, \mathbf{cv}_{\text{ry}_{\text{cid}}}, \mathbf{d}_{\text{vry}_{\text{cid}}})$. Again by (iv) in G_1 this is the credential that the honest issuance would produce.

The two oracles that use ik in G_1 are therefore answered through **Sign**, and the simulation of G_1 is perfect. Note also that $\mathcal{A}_{\text{EUF-CMA}}$ queries **Sign** once per completed issuance session and once per honest client, and never otherwise, so the set $\mathcal{Q}$ of Fig. 5 is exactly the set M of (ii) in the game G_1 . When $\mathcal{A}$ halts, $\mathcal{A}_{\text{EUF-CMA}}$ looks for an accepted verification session with $m^* \notin M$;

if it finds one it outputs (m^*, σ^*) , and otherwise it outputs $\perp$. Such a session is accepted only if $\text{RSA.Vry}(\text{vk}, m^*, \sigma^*) = 1$, and $m^* \notin \mathcal{Q}$, so the pair is a valid forgery. Hence $\mathcal{A}_{\text{EUF-CMA}}$ wins EUF-CMA whenever E_2 occurs, and:

$$\Pr[E_2] \leq \text{Adv}_{\text{RSA}}^{\text{euf-cma}}(\mathcal{A}_{\text{EUF-CMA}}, \lambda). \quad (7)$$

After G_2 , every accepted verification session has $m^* \in M$, hence either $\mathbf{c}_{\text{Vry}}^* = \mathbf{c}_{\text{Vry}_{\text{sid}}}$ for some $\text{sid} \in \mathcal{J}$, or $\mathbf{c}_{\text{Vry}}^* = \mathbf{c}_{\text{Vry}_{\text{cid}}}$ for some honest client cid .

Game G_3 . In this game, when $\mathcal{A}$ halts and before computing win , the game checks the following event ($E_3 : \iff \exists (\phi_{\text{Vry}}^*, \mathbf{opn}^*) \in \mathcal{W}, \exists \text{sid} \in \mathcal{J}, \exists i \in \text{flds}(\phi_{\text{Vry}}^*) : \mathbf{c}_{\text{Vry}}^* = \mathbf{c}_{\text{Vry}_{\text{sid}}} \wedge \mathbf{opn}_i^* \neq \mathbf{a}_{\text{sid},i}$). Namely, the adversary opened a (component of the vector of a) commitment signed in one of its own issuance sessions to a value other than the one extracted from that session. The game outputs 0 if E_3 occurs. We bound $\Pr[E_3]$ by defining an adversary $\mathcal{A}_{\text{BND},1}$ that plays in game BND against Com_{vry} .

We describe the strategy of $\mathcal{A}_{\text{BND},1}$. On input 1^λ , $\mathcal{A}_{\text{BND},1}$ runs G_2 entirely by itself. That is, it generates (vk, ik) , plays the four oracles for $\mathcal{A}$ as in the game G_2 . If E_3 occurs, it fixes one triple $((\phi_{\text{Vry}}^*, \mathbf{opn}^*), \text{sid}, i)$ and outputs $(\mathbf{c}'_{\text{sid},i}, \mathbf{a}_{\text{sid},i}, \mathbf{opn}_i^*, \mathbf{d}'_{\text{sid},i}, \mathbf{d}_i^*)$. Both openings are valid: $\text{Com}_{\text{vry}}.\text{Opn}(\mathbf{a}_{\text{sid},i}, \mathbf{c}'_{\text{sid},i}, \mathbf{d}'_{\text{sid},i}) = 1$ by fact (i) of G_1 , and $\text{Com}_{\text{vry}}.\text{Opn}(\mathbf{opn}_i^*, \mathbf{c}'_{\text{sid},i}, \mathbf{d}_i^*) = 1$ because the honest verifier accepted the session and $\mathbf{c}_i^* = \mathbf{c}'_{\text{sid},i}$, the two commitment vectors being equal. As $\mathbf{opn}_i^* \neq \mathbf{a}_{\text{sid},i}$, the two openings are to different values, so $\mathcal{A}_{\text{BND},1}$ wins BND whenever E_3 occurs, and:

$$\Pr[E_3] \leq \text{Adv}_{\text{Com}_{\text{vry}}}^{\text{bnd}}(\mathcal{A}_{\text{BND},1}, \lambda). \quad (8)$$

After G_3 , consider an accepted verification session with $\mathbf{c}_{\text{Vry}}^* = \mathbf{c}_{\text{Vry}_{\text{sid}}}$ for some $\text{sid} \in \mathcal{J}$. It has $\mathbf{opn}_i^* = \mathbf{a}_{\text{sid},i}$ for every $i \in \text{flds}(\phi_{\text{Vry}}^*)$, and $\mathbf{attrs}_{\text{sid}} \in \mathcal{Q}$, so the first condition of the final step of Fig. 3 is not met and the session does not set win . Hence, in G_3 , win can be set only by an accepted verification session with $\mathbf{c}_{\text{Vry}}^* = \mathbf{c}_{\text{Vry}_{\text{cid}}}$ for some honest client cid and $\{(i, \mathbf{opn}_i^*) : i \in \text{flds}(\phi_{\text{Vry}}^*)\} \not\subseteq \mathcal{R}_{\text{cid}}$, that is, by a showing built on the credential of an honest client that reveals, in some field i^* , a pair $(i^*, \mathbf{opn}_{i^*}^*)$ that this client never revealed to $\mathcal{A}$.

Game G_4 . In this game, when $\mathcal{A}$ halts and before computing win , the game checks the following event ($E_4 : \iff \exists (\phi_{\text{Vry}}^*, \mathbf{opn}^*) \in \mathcal{W}, \exists \text{cid}, \exists i^* \in \text{flds}(\phi_{\text{Vry}}^*) : \mathbf{c}_{\text{Vry}}^* = \mathbf{c}_{\text{Vry}_{\text{cid}}} \wedge (i^*, \mathbf{opn}_{i^*}^*) \notin \mathcal{R}_{\text{cid}}$). Namely, the honest verifier accepted a showing built on the credential of an honest client which reveals, in some field, a value that this client never revealed to $\mathcal{A}$. The game outputs 0 if E_4 occurs.

By the closing remark of G_3 , this is exactly the condition under which win can still be set, so G_4 never outputs 1, i.e., $p_4 = 0$, and it only remains to bound $\Pr[E_4]$. We split E_4 according to whether the revealed value agrees with the attribute of the honest client in the field i^* . That is:

$$(E_{4,\text{neq}} : \iff E_4 \wedge \mathbf{opn}_{i^*}^* \neq \mathbf{a}_{\text{cid},i^*}) \quad (E_{4,\text{eq}} : \iff E_4 \wedge \mathbf{opn}_{i^*}^* = \mathbf{a}_{\text{cid},i^*}).$$

The two events are disjoint, and their union is exactly the event $E_4 := E_{4,\text{neq}} \cup E_{4,\text{eq}}$, so we have the following $\Pr[E_4] = \Pr[E_{4,\text{neq}}] + \Pr[E_{4,\text{eq}}]$, and we bound the two probabilities separately. They are of a different nature. If we are in the event $E_{4,\text{neq}}$, the adversary opens a commitment of an honest client to a value other than the committed one, which is a binding break and is handled as in G_3 . Otherwise, in $E_{4,\text{eq}}$ the two openings agree on the value, so nothing is broken in the sense of binding: what $\mathcal{A}$ has done is to produce a valid opening of a commitment whose opening it never received, the field i^* having never been revealed to it. We bound this second event through the hiding of Com_{vry} .

BOUNDING $\Pr[E_{4,\text{neq}}]$. We define an adversary $\mathcal{A}_{\text{BND},2}$ that, on input 1^λ , runs G_3 entirely by itself, as $\mathcal{A}_{\text{BND},1}$ does. If $E_{4,\text{neq}}$ occurs, it fixes one triple $((\phi_{\text{Vry}}^*, \mathbf{opn}^*), \text{cid}, i^*)$ and outputs

$(c'_{\text{cid},i^*}, a_{\text{cid},i^*}, \text{opn}_{i^*}^*, d'_{\text{cid},i^*}, d_{i^*}^*)$; it holds the opening d'_{cid,i^*} because it ran the honest issuance of cid itself. Both openings are valid, the first by the correctness of Com_{vry} and the second because the honest verifier accepted the session and $c_{i^*}^* = c'_{\text{cid},i^*}$, the two commitment vectors being equal; and the two values differ. Hence:

$$\Pr[E_{4,\text{neq}}] \leq \text{Adv}_{\text{Com}_{\text{vry}}}^{\text{bnd}}(\mathcal{A}_{\text{BND},2}, \lambda). \quad (9)$$

BOUNDING $\Pr[E_{4,\text{eq}}]$. The idea is to replace one commitment of one honest client with a commitment to $\perp$: if $\mathcal{A}$ can still open it to the attribute, then either it has noticed the substitution, which breaks the hiding of Com_{vry} , or it has opened a commitment to $\perp$ to a value different from $\perp$, which breaks its binding. Since the substitution has to be decided before the run, the reduction guesses which client and which field, at a loss of a factor $q_C \cdot \ell$.

We describe the strategy of an adversary $\mathcal{A}_{\text{HDN}}$ that plays in game HDN against Com_{vry} . On input 1^λ , $\mathcal{A}_{\text{HDN}}$ samples a pair $(\text{cid}^*, i^*) \leftarrow \$ [q_C] \times [\ell]$ and runs G_3 for $\mathcal{A}$ by itself, generating (vk, ik) and playing the four oracles, with two exceptions:

- when the honest-issuance oracle creates the client cid^* , $\mathcal{A}_{\text{HDN}}$ obtains the commitment c'_{cid^*,i^*} by querying its oracle Cmt of Fig. 7 on the pair $(m_0, m_1) := (a_{\text{cid}^*,i^*}, \perp)$, and computes the rest of the credential as the oracle would, i.e., the remaining commitments with their openings, the message m_{cid^*} from all of them, and $\sigma_{\text{cid}^*} \leftarrow \text{RSA.Sig}(\text{ik}, m_{\text{cid}^*})$, which by fact (iv) is how the honest issuance computes it;
- if $\mathcal{A}$ ever asks the honest-verification oracle to make the client cid^* reveal the field i^* , $\mathcal{A}_{\text{HDN}}$ halts and outputs a uniformly random bit.

Note that Cmt returns the commitment but not its opening, so $\mathcal{A}_{\text{HDN}}$ must run the game without d'_{cid^*,i^*} . This is possible: the issuance of cid^* is played by the game on both sides, so no proof is ever computed for it, and a showing of cid^* sends the opening of a field only if it reveals that field, which the second exception rules out. When $\mathcal{A}$ halts, $\mathcal{A}_{\text{HDN}}$ outputs $b' := 0$ if $\mathcal{W}$ contains an accepted tuple with $\mathbf{c}_{\text{vry}}^* = \mathbf{c}_{\text{vry},\text{cid}^*}$, $i^* \in \text{flds}(\phi_{\text{vry}}^*)$ and $\text{opn}_{i^*}^* = a_{\text{cid}^*,i^*}$, and $b' := 1$ otherwise. Suppose first that the challenge bit b of HDN is 0. Then c'_{cid^*,i^*} is a fresh commitment to a_{cid^*,i^*} , exactly as the honest client would compute it, so $\mathcal{A}_{\text{HDN}}$ perfectly simulates G_3 as long as it does not halt. The pair (cid^*, i^*) is sampled independently of the run, and $E_{4,\text{eq}}$ occurring on that pair implies that the field i^* of cid^* was never revealed, hence that $\mathcal{A}_{\text{HDN}}$ does not halt and that $b' = 0$; therefore $\Pr[b' = 0 \mid b = 0] \geq \Pr[E_{4,\text{eq}}]/(q_C \cdot \ell)$.

Suppose now that $b = 1$. Then c'_{cid^*,i^*} is a commitment to $\perp$, and $b' = 0$ means that $\mathcal{A}$ has opened it to a_{cid^*,i^*} , which differs from $\perp$ since $\perp$ is not a valid attribute. We turn this into a binding break: let $\mathcal{A}_{\text{BND},3}$ be the adversary that runs $\mathcal{A}_{\text{HDN}}$, replacing the query to Cmt with a pair $(c, d) \leftarrow \text{Com}_{\text{vry}}.\text{Cmt}(\perp)$ that it computes itself, and that outputs $(c, \perp, \text{opn}_{i^*}^*, d, d_{i^*}^*)$ whenever $b' = 0$. The first opening is valid by construction, the second because the honest verifier accepted the session and $c_{i^*}^* = c$, and the two values differ; therefore $\Pr[b' = 0 \mid b = 1] \leq \text{Adv}_{\text{Com}_{\text{vry}}}^{\text{bnd}}(\mathcal{A}_{\text{BND},3}, \lambda)$.

Since $\Pr[\text{HDN}_{\text{Com}_{\text{vry}}}^{\mathcal{A}_{\text{HDN}}}(\lambda) = 1] = \frac{1}{2} \Pr[b' = 0 \mid b = 0] + \frac{1}{2} (1 - \Pr[b' = 0 \mid b = 1])$, the two bounds above give $\Pr[b' = 0 \mid b = 0] - \Pr[b' = 0 \mid b = 1] \leq 2 \text{Adv}_{\text{Com}_{\text{vry}}}^{\text{hdn}}(\mathcal{A}_{\text{HDN}}, \lambda)$, whence:

$$\Pr[E_{4,\text{eq}}] \leq q_C \cdot \ell (2 \text{Adv}_{\text{Com}_{\text{vry}}}^{\text{hdn}}(\mathcal{A}_{\text{HDN}}, \lambda) + \text{Adv}_{\text{Com}_{\text{vry}}}^{\text{bnd}}(\mathcal{A}_{\text{BND},3}, \lambda)). \quad (10)$$

Combining Eqs. (9) and (10) bounds $\Pr[E_4]$, and combining such bound with Eqs. (6) to (8) completes the proof of the theorem.

F On Revocation in CRACS

As is common in privacy-preserving credential systems, stronger privacy guarantees complicate revocation: an issuer that does not learn the content of the issued credential cannot, in general, later identify it for revocation. This difficulty is inherent to the goal of issue-show unlinkability and is therefore shared by all prior constructions achieving it, which resort to short-lived credentials, status lists over blinded identifiers, or cryptographic accumulators. The same mechanisms apply to our setting. The simplest one, adopted by current EUDIW deployments through short validity periods and batch issuance, is to issue short-lived credentials and revoke by non-renewal.

Revocation lists also remain available. During issuance the issuer learns only $t = f(s)$, where s is chosen uniformly at random by the user and a commitment c to s is included in the credential; to revoke, the issuer publishes t . At presentation, the user proves through a non-membership proof that the value committed in c has no image among the published identifiers, without revealing it. Note that this preserves unlinkability with the issuance session: the issuer sees t but never c , and bridging the two would require s , hence breaking the hiding property of the commitment or the one-wayness of f .

Verifying such a proof would push the verifier beyond checking standardized signatures and hashes. Here our setting is in fact better placed than prior work, which already requires non-standard operations at presentation time for the credential itself. The non-membership proof can be decoupled from presentation by resorting to designated authorities (e.g., the issuers themselves) that verify such proofs and produce signed standardized attestations stating that c does not correspond to a revoked identifier; the verifier then checks only the attestation, remaining within elementary standardized operations. Attestations can be obtained in advance and in bulk, independently of any specific presentation, so that such an authority learns nothing about credential usage.