RETRACT: Expressive Designated Verifier Anonymous Credentials

Anonymous Author(s)

ABSTRACT

Anonymous credentials (ACs) are digital cryptographically-secure versions of paper and digital credentials that let us selectively prove possession of encoded attributes (claims) to verifiers such as digital services, employers, or government departments without disclosing any other information. While attributes by governmental issuers usually reflect basic personal information about the credential holder (e.g., name, gender, age, address), attributes can also reflect more extensive claims about holders, such as the holder's platform details and configuration. Since the attributes might be sensitive, it is popular to embed additional attributes in the credential about the existing attributes, e.g., that age is above 18, thus allowing a holder to show that their age satisfies some condition without revealing the exact age. However, since each verifier might have different policies that must be satisfied, it is becoming increasingly impractical for issuers to embed all possible claims in a credential. To mitigate this problem and allow arbitrary policies to be checked against individual attributes without complicating or overwhelming the credential, we propose to let verifiers dynamically define policies as high-level programs which can be *verifiably* executed by holders on their credentials. Furthermore, to mitigate the potential risk of dishonest verifiers attempting to benefit or otherwise leak sensitive information learned through this unlimited expressiveness of policies, we propose making the proofs *designated verifier*. Thus, any proof produced for one verifier cannot be used to convince another.

KEYWORDS

Anonymous Credentials, Designated Verifier, zkSNARKs

ACM Reference Format:
Anonymous Author(s). 2023. RETRACT: Expressive Designated Verifier Anonymous Credentials. In *Proceedings of 18th International Conference on Availability, Reliability and Security (ARES '23).* ACM, New York, NY, USA, 12 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Currently, there is an increasing shift to decentralized digital identity models, where there is no single governing organization that has control over identity data origination. Instead, participants produce and manage their own identifiers and credentials without deference or permission from any other administrative organization. The World Wide Web Consortium (W3C) is currently developing two new standards to realize this emerging model, namely, Decentralized Identifiers (DIDs) [51] and Verifiable Credentials (VCs) [45].

In the ecosystem of Verifiable Credentials, the Issuer issues a credential containing a set of claims on a Subject and transfers it to a Holder, who is typically the same entity. The Holder stores the VCs in a storage called the Identity Wallet. In response to a request from the Verifer, the Holder retrieves one or more stored VCs from her Wallet and presents them to the Verifer. Alternatively, the Holder can construct Verifiable Presentations (VPs), i.e. collection of claims that a Holder can contract from different VCs issued by varying entities. Then, a Holder can prove to a Verifer that it owns a VC or VP with certain attributes. This is usually achieved through a unique identifier (e.g., public key), owned by the Holder that enables her to generate proof of possession on specific claims (e.g., a digital signature with the corresponding private key).

VCs can be combined with anonymous credentials [14, 19] to enable the Holder to manage her privacy by choosing the level of information disclosure. That is, the Holder can select only some of the attributes in the credentials she owns and prove that they are certified by a trusted Issuer, without revealing any further information; i.e., a signature from the Holder's unique identifier or other remaining attributes. This property is called *selective disclosure.*

One core challenge is the verification of the integrity and origin of the presented VCs or VPs: *How can someone be sure that they really belong to the claimed entity without disclosing unecessary information?* On a technical level, this translates into Holders having control of their own VCs and DIDs through their Wallets, which can ensure that credentials and (private) keys can only become available to this specific Holder as the actual owner of the issued wallet credentials. Since it is only the Holder (as the Identity Owner) that knows the key associated with a DID, the level of control and credential management assurance relies on possessing and controlling the private key for creating (anonymized) VPs. However, is the integration of only such a single authentication factor enough? For example, in the case of Europe, the eIDAS regulation [49] clearly defines the requirement for multiple authentication factors to achieve a Level of Assurance classified as "substantial" (e.g., fingerprints and secret key).

One of the necessary measures to solve such security gaps, is to incorporate zero-knowledge proofs [13, 18, 26, 27, 33, 42, 46, 48, 50], where credential holders can prove that they possess attributes that satisfy some condition without revealing any other information. However, several schemes still only support a limited range of predicates [13, 26, 46, 48, 50], such as basic boolean operators, or being limited to range proofs or set membership proofs. Furthermore, another challenge, as mentioned in [32], occurs when considering anonymous credentials on devices, such as smartphones, where we encounter issues such as credential sharing and the need to cope with fewer computational resources. To address this issue, authors of [32] proposed the notion of core/helper anonymous credentials, where credentials are split between a secure core (e.g., a SIM card) and a more resourceful helper (e.g., a smartphone). Here the idea is that the helper cannot use the credential without the core's help. Though, as the authors mention, the core's effort must be minimal and independent of the credential's size due to its limited resources. Finally, most prior literature on anonymous credentials does not

ARES '23, August 29–September 01, 2023, Benevento, Italy
© 2023 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

consider the designated verifier property [35]. Specifically, in most schemes, there is no attempt to prevent the verifier from leaking whatever information they learned from the verification process to other third parties. The designated verifier property is needed to protect against such malicious activities by dishonest verifiers. Regardless of whether the holder discloses attributes or proves to possess attributes that satisfy a policy or a combination of the two, it should not be possible for verifying parties to misuse or leak information obtained from the verification process.

Contribution: We present a fully *expRessive dEsignaTed veRifier Anonymous CredenTials* (RETRACT) scheme in the core/helper model of [32] that uses BBS+ signatures for building credentials and incorporates state-of-the-art verifiable computation techniques to allow holders to prove arbitrary predicates on their credentials in zero knowledge. In a nutshell, to combine BBS+ signatures with verifiable computation, we consider commit-carrying zero-knowledge Succinct Non-interactive ARguments of Knowledge (cc-zkSNARK) [17, 30] proof constructions that accept predicates expressed as arithmetic circuits. We then show how to use well-established proof statement composition methods [15] for extending a *proof of knowledge* of a valid BBS+ signature with proof that the commitment contains attributes from the BBS+ signature while also ensuring that the overall scheme remains *designated verifier*. While the generation of zkSNARK proofs is generally considered slow, the proofs are short and verified remarkably fast, making zkSNARKs attractive, especially in Distributed Ledger Technology (DLT), e.g., the anonymous cryptocurrency Zcash and Ethereum, and for verifiable credentials. Our prototype implementation is accessible online [3].

2 RELATED WORKS

Flexible credentials. Following the initial work of Chaum [20], there has been a long line of work (e.g., [6, 12, 13, 32]) with successively more efficient and expressive anonymous credentials that have been widely deployed in several real-world applications, such as U-Prove [40], and Idemix [16]. Recently, we have also witnessed a synergy between anonymous credentials and predicate proofs to improve expressiveness. For example, Trinsic [48] uses BBS+ signatures [4] and allows for range-based predicate proofs using basic arithmetic operators. Similarly, the Decentralized Identity Foundation (DIF) allows some algebraic rules and set membership checks as part of their presentation exchange specification [26]. Another recent solution is Dock [27], which has recently upgraded from supporting basic predicate proofs, e.g., membership and range proofs, to supporting more arbitrary predicates expressed in Circom [34] and proven correct using LegoGro16 [17]. In [18], researchers proposed *zero-knowledge credentials* in the decentralized identity (DID) ecosystem, where holders can employ a general-purpose zkSNARK proof system (described in Section 3.5) to produce proofs of arbitrary computations over their credentials. Similarly, authors of [42] also consider zkSNARKs and propose a toolkit for creating complex statements and the composition of credentials.

Designated verifier. While not an anonymous credential scheme, authors of [31] propose a single-sign-on (SSO) protocol that uses BBS+ signatures [4] and adopts a form of *designated verifier*. Specifically, in their SSO scheme, authentication tags can only be validated by the verifier of the service for which they were designated. However, as the authors note, this version of "designated verifier" is slightly different from that defined initially by Jacobsson [35] (described in Section 3.6). Specifically, in [35], the idea of a designated verifier is to prevent the verifier from convincing others about a transcript since the verifier could just as well have generated it, whereas in [31] anyone can verify the signer of authentication tags. In [25], authors propose functional credentials based on homomorphic attribute-hiding predicate encryption schemes. The idea for holders to prove statements is: given a ciphertext encoding a given policy, a holder decrypts the ciphertext to convince a verifier that they know a key for a set of attributes that matches the policy. However, in their scheme [25], the idea is to keep policies secret from designated verifiers. While useful in certain applications, this definition of the designated verifier also differs from that of [35]. Finally, [24] propose using smooth projective hash functions (SPHF) [23] to allow holders to make designated-verifier proofs.

Core/helper setting. As mentioned in [32], a prominent example of the core/helper setting is the Direct Anonymous Attestation (DAA) protocol [9], which was designed for privacy-preserving remote attestation of platforms. Here the core device is the Trusted Platform Module (TPM) [47], and the helper is the hosting platform to which the TPM is connected. While DAA is technically a group signature protocol for creating anonymous signatures (with optional linkability) on messages to convince a verifier that an authorized TPM signed a message, some recent works extend the DAA protocol with attributes (DAA-A), and selective disclosure [10, 11, 21]. While [21] considered CL [14] and SDH [8] signatures for credentials, the recent DAA-A schemes [10, 11] shifted to q-SDH BBS+ signatures [4]. However, noting how DAA-A schemes are tailored towards a specific core (i.e., the TPM), authors of [32] proposed core/helper anonymous credentials (CHAC), which uses a combination of signatures with flexible public keys (SFPK) [5] and a novel notion of aggregatable attribute-based equivalence class signatures (AAEQ). In the generalized core/helper model defined by [32], the core can be any secure element, such as a SIM card, an NFC-based smart card, a "software-based" Trusted Execution Environment (TEE) like TrustZone or SGX, or even a TPM.

3 BACKGROUND AND PRELIMINARIES

This section presents the considered primitives and terminologies used in describing our scheme. Note that we denote sequences and vectors in **bold**. Furthermore, by $[n]$ we denote the set of integers $\{1, \dots, n\}$ and by $(a_i)_{i \in [L]}$, we denote the tuple $(a_1, \dots, a_L)$.

3.1 Bilinear Groups and Pairings

Let $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ be three finite cyclic groups with prime order p. We work with bilinear groups $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$, where e defines the mapping $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, which is bilinear, i.e., $e(g_1^x, g_2^y) = e(g_1, g_2)^{xy}$, non-degenerate, i.e., for all generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, $e(g_1, g_2)$ generates $\mathbb{G}_T$, and efficient, i.e., there exists an efficient algorithm $\mathcal{G}(1^\lambda)$ that outputs the bilinear group and an efficient algorithm to compute $e(a, b)$ for any $a \in \mathbb{G}_1, b \in \mathbb{G}_2$. In our scheme, bilinear groups are used to support the prominent BBS+ signature scheme described in Section 3.4, which we utilize to express verifiable credentials as issued by some trusted issuer.

Furthermore, note that pairings are often classified into one of three types (see [11] for details). However, for the purposes of this paper, it suffices to say that we consider the third type (Type-III), which allows for efficient operations in $\mathbb{G}_1$ and is used in the considered BBS+ signature scheme, which has been proven secure under the JOC version (supporting Type-III pairings) of the q-Strong Diffie-Hellman (qSDH) assumption [7]. Finally, we use $1_{\mathbb{G}}$ to denote the identity element in the group $\mathbb{G}$.

3.2 Pedersen Commitment

The Pedersen commitment scheme [41] is an unconditionally hiding and computationally binding commitment scheme based on the discrete logarithm problem. It consists of three algorithms Ped = (Setup, Commit, VerCommit) that work as follows and satisfy the notions of *correctness, binding*, and *hiding* as defined below.

- $\mathsf{Ped.Setup}(1^\lambda, n) \to \mathbf{ck}$: given a security parameter and a desired number of values n, take $(h_0, \ldots, h_n) \leftarrow\$ \mathbb{G}_1^{n+1}$, and output $\mathbf{ck} \leftarrow (h_0, \ldots, h_n)$ as the commitment key.
- $\mathsf{Ped.Commit}(\mathbf{ck}, (u_1, \ldots, u_n)) \to (t, o)$: given a commitment key and a sequence of values, parse $\mathbf{ck} = (h_0, \ldots, h_n)$, take $o \leftarrow\$ \mathbb{Z}_q$, compute $t \leftarrow h_0^o \prod_{i=1}^n h_i^{u_i}$, and output t as the commitment, and o as the opening value (blinding factor).
- $\mathsf{Ped.VerCommit}(\mathbf{ck}, t, (u_1, \ldots, u_n), o) \to b \in \{0, 1\}$: given a commitment key, a commitment, a sequence of values, and an opening, parse $\mathbf{ck} = (h_0, \ldots, h_n)$, and only accept ($b = 1$) the commitment if $t = h_0^o \prod_{i=1}^n h_i^{u_i}$.

Correctness. For all $\lambda \in \mathbb{N}$ and any vector $\boldsymbol{u}$ of n values we have:

$$\Pr\left[\begin{array}{c}\mathbf{ck} \leftarrow \mathsf{Setup}(\lambda, n)\\ (t, o) \leftarrow \mathsf{Commit}(\mathbf{ck}, \boldsymbol{u})\end{array} : \mathsf{VerCommit}(\mathbf{ck}, t, \boldsymbol{u}, o) = 1\right] = 1$$

Binding. For every polynomial-time adversary $\mathcal{A}$ we have:

$$\Pr\left[\begin{array}{c}\mathbf{ck} \leftarrow \mathsf{Setup}(\lambda, n)\\ (t, \boldsymbol{u}, o, \boldsymbol{u}', o') \leftarrow \mathcal{A}(\mathbf{ck})\end{array} : \begin{array}{c}\mathsf{VerCommit}(\mathbf{ck}, t, \boldsymbol{u}', o')\\ \wedge \mathsf{VerCommit}(\mathbf{ck}, t, \boldsymbol{u}, o)\\ \wedge \boldsymbol{u} \neq \boldsymbol{u}'\end{array}\right] = \mathsf{negl}$$

Hiding. For $\mathbf{ck} \leftarrow \mathsf{Setup}(\lambda, n)$ and every $\boldsymbol{u}, \boldsymbol{u}'$, we require $\mathsf{Commit}(\mathbf{ck}, \boldsymbol{u}) \approx \mathsf{Commit}(\mathbf{ck}, \boldsymbol{u}')$.

3.3 Proof of Knowledge of Algebraic Statements

To prove knowledge of the secret ingredients (u, o) of a Pedersen commitment c without disclosing either value, i.e., neither the committed value u nor its opening o, we can run a zero-knowledge proof of knowledge protocol. There are essentially two common ways to design non-interactive zero-knowledge (NIZK) proofs: Sigma protocols and zkSNARK constructions [1]. The former is highly efficient for proving algebraic statements, while the latter is superior for more expressive arithmetic representations. In our scheme, we utilize both: the latter to prove arbitrary predicates expressed as arithmetic circuits on credential attributes and the former to prove that those inputs originated from a valid credential.

When referring to zero-knowledge proofs of knowledge of discrete logarithms and statements about them, we adopt the notation of [11]. For example, $PoK\{(a, b, c) : y = g^a h^b \wedge \tilde{y} = \tilde{g}^a \tilde{h}^c\}$ denotes a "*zero-knowledge proof of knowledge of integers (scalars) a, b, and c such that $y = g^a h^b$ and $\tilde{y} = \tilde{g}^a \tilde{h}^c$ holds*," where $y, g, h, \tilde{y}, \tilde{g}$, and $\tilde{h}$ are elements of some groups $\mathbb{G} = \langle g \rangle = \langle h \rangle$ and $\tilde{\mathbb{G}} = \langle \tilde{g} \rangle = \langle \tilde{h} \rangle$, respectively. The convention is that the values in the parenthesis (a, b, c) represent the secret knowledge (witnesses) that is being proven by using the other values to which the verifier has access. In our construction, we consider the generalized Schnorr protocol [44] to create proofs of such composite statements, which, due to space limitations, is described in Appendix A.

3.4 BBS+ Signatures

Inspired by the group signature scheme in [8], BBS+ signatures [11] are a multi-message digital signature scheme that allows for signing an ordered list of messages where the specially produced signature has a constant size, regardless of the number of messages. In the context of verifiable credentials, note that we consider "attributes" instead of "messages", i.e., a verifiable credential is an ordered set of attributes (representing different claims) with a corresponding BBS+ signature over those attributes from some trusted issuer. Given such a BBS+ signature, the credential holder can create zero-knowledge *proofs of knowledge* of the signature and the corresponding signed attributes and optionally disclose select attributes.

In total, a BBS+ signature scheme consist of three algorithms BBS+ = (KeyGen, Sign, Verify) that work as follows.

- $\mathsf{BBS+.KeyGen}(L) \to (\mathbf{ick}, \mathsf{ipk}, \mathsf{isk})$: given a desired number of attributes L, take $(h_0, \ldots, h_L) \leftarrow\$ \mathbb{G}_1^{L+1}$, $\mathsf{isk} \leftarrow\$ \mathbb{Z}_p^*$, $\mathsf{ipk} \leftarrow g_2^{\mathsf{isk}}$, and output $\mathbf{ick} \leftarrow (h_0, \ldots, h_L)$ as the commitment key and ipk and isk as the public and secret keys, respectively.
- $\mathsf{BBS+.Sign}(\mathsf{isk}, \mathbf{ick}, (a_1, \ldots, a_L)) \to \sigma$: given a secret key, a commitment key, a sequence of attributes to sign, parse $\mathbf{ick} = (h_0, \ldots, h_L)$, choose a random $e, s \leftarrow\$ \mathbb{Z}_p$, compute $A \leftarrow (g_1 h_0^s \prod_{i=1}^L h_i^{a_i})^{1/(e+\mathsf{isk})}$, and output $\sigma \leftarrow (A, e, s)$ as the multi-attribute BBS+ signature.
- $\mathsf{BBS+.Verify}(\mathsf{ipk}, \mathbf{ick}, (a_1, \ldots, a_L), \sigma) \to b \in \{0, 1\}$: given a public key, a commitment key, a sequence of attributes, and a purported signature, parse $\mathbf{ick} = (h_0, \ldots, h_L)$, $\sigma = (A, e, s)$, and accept ($b = 1$) the signature only if $e(A, \mathsf{ipk} \cdot g_2^e) = e(g_1 h_0^s \prod_{i=1}^L h_i^{a_i}, g_2)$.

Note that in our construction, we employ an extension of the above notation, which we describe in Section 5.1.1 to require the assistance of the credential holder's core element when producing presentations (i.e., proofs of knowledge of a BBS+ signature).

3.5 (Commit-Carrying) zkSNARKs

While Sigma protocols are efficient for algebraic statements, they are significantly slower when it comes to non-algebraic ones [1], e.g., cryptographic hash functions represented as arithmetic circuits. Fortunately, constructions called zero-knowledge Succinct Non-Interactive ARguments of Knowledge (zkSNARKs) [30] present an effective alternative approach to proving statements about functions represented as Boolean or arithmetic circuits C, which, in turn, are expressed in NP-complete languages such as Rank-1-Constraint-System (R1CS) or Quadratic Arithmetic Programs (QAPs).

In a nutshell, a zkSNARK allows the credential holder to prove that they have correctly executed an arithmetic circuit C on public input x and secret input w (called the witness), as follows. After taking C as input, a one-time setup is performed to give two public keys: an evaluation key ek and a verification key vk. The evaluation key ek enables credential holders to produce a proof π attesting to the fact that x and w satisfied C. The non-interactive proof π is zero knowledge and a *proof of knowledge*. The proof reveals nothing about u, but anyone can verify its correctness using only vk.

Furthermore, note that a credential holder is expected to supply attributes from its issued credential as secret witnesses to the arithmetic circuits to prove that its attributes satisfy some arbitrarily complex predicate. However, since a holder might cheat, we require that the holder additionally proves that the secret witness used in the proof generation matches the attribute in its issued credential. While the standard zkSNARK construction has no such capability built-in (and it is *costly* to express directly in a circuit), there, fortunately, exists an alternative construction called *commit-carrying zk-SNARKs* (cc-zkSNARKs) [17], where the proof additionally contains a commitment (in our case we consider the Pedersen commitment described in Section 3.2) to some portion u of the witness, i.e., we assume that the witness can be split into two subdomains $w = (u, \omega)$, where ω refers to the non-committed part of the witness.

In total, cc-zkSNARK schemes consist of three algorithms $\mathsf{cc}\Pi = (\mathsf{KeyGen}, \mathsf{Prove}, \mathsf{VerProof})$ that work as follows and satisfy the notions of *zero-knowledge*, *completeness*, *succinctness*, *knowledge soundness*, and *binding* as defined below.

- $\mathsf{cc}\Pi.\mathsf{KeyGen}(\mathsf{C}, W, 1^\lambda) \rightarrow (\mathbf{ck}, \mathsf{ek}, \mathsf{vk})$: given an arithmetic circuit C, a desired number of witnesses to commit to W, and a security parameter λ, output a common reference string that includes a commitment key $\mathbf{ck}$ with $W + 1$ generators, an evaluation key ek, and a verification key vk.
- $\mathsf{cc}\Pi.\mathsf{Prove}(\mathsf{C}, \mathsf{ek}, x, w) \rightarrow (t, \pi, o)$: given an evaluation key ek for a circuit C, public input x and secret witness $w = (u, \omega)$ such that $\mathsf{C}(x, w)$ holds, output a proof π, commitment t, and opening o such that $\mathsf{Ped.VerCommit}(\mathbf{ck}, t, u, o) = 1$.
- $\mathsf{cc}\Pi.\mathsf{VerProof}(\mathsf{C}, \mathsf{vk}, x, t, \pi) \rightarrow b \in \{0, 1\}$: given a verification key vk for a circuit C, public input x, a commitment t, either accepts ($b = 1$) or rejects ($b = 0$) the proof π.

Completeness. For any $\lambda \in \mathbb{N}$ and C where $\mathsf{C}(x, w) = 1$, it holds:

$$\Pr\left[\begin{array}{c}(\mathbf{ck}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{C}, W, \lambda) \\ (t, \pi, o) \leftarrow \mathsf{Prove}(\mathsf{C}, \mathsf{ek}, x, w)\end{array} : \mathsf{VerProof}(\mathsf{C}, \mathsf{vk}, x, t, \pi)\right] = 1$$

Binding. For every polynomial-time adversary $\mathcal{A}$ the following probability is $\mathsf{negl}(\lambda)$:

$$\Pr\left[\begin{array}{c}(\mathbf{ck}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{C}, W, \lambda) \\ (t, \boldsymbol{u}, o, \boldsymbol{u}', o') \leftarrow \mathcal{A}(\mathsf{C}, \mathbf{ck}, \mathsf{ek}, \mathsf{vk})\end{array} : \begin{array}{c}\mathsf{VerCommit}(\mathbf{ck}, t, \boldsymbol{u}', o') \\ \wedge \mathsf{VerCommit}(\mathbf{ck}, t, \boldsymbol{u}, o) \\ \wedge \boldsymbol{u} \neq \boldsymbol{u}'\end{array}\right]$$

Besides these notions, knowledge-soundness informally states that we can efficiently "extract" a valid witness from proofs that pass verification. Succinctness means that proofs are of size $\mathsf{poly}(\lambda) \cdot (\lambda + \log |w|)$ and can be verified in time $\mathsf{poly}(\lambda)(\lambda + |x| + \log |w|)$. Finally, zero knowledge essentially means that proofs leak nothing about the witness. See [17] for further details.

Note that we only assume commit-carrying zkSNARKs in the formalization of our scheme. Here the commitment key depends on the relation taken by KeyGen, and a commitment is freshly created by the Prove algorithm and is tied to a single proof. However, note that the cc-SNARK lifting compiler in [17] can turn any cc-SNARK into the more versatile commit-and-prove zkSNARK version where the commitment key is relation-independent and allows for finer composition of different CP-zkSNARKs. Thus, it follows that our approach can be extended to work with modular CP-zkSNARKs.

3.6 Designated Verifier Proofs

Let Φ be our proof statement (e.g., proof of knowledge of a BBS+ signature). Jakobsson in [35] introduced the concept of a designated verifier, which essentially means that in a proof of Φ, we ensure that the proof can convince only a particular verifier. The idea is simple, instead of directly proving Φ, we create a transcript π of the disjunctive proof statement $\Phi \vee \phi_{Bob}$, where ϕ_{Bob} is a proof of knowledge of the designated verifier's secret key, in this case, Bob's. It essentially becomes a designated verifier proof, as Bob, the designated verifier, can always use his trapdoor to simulate a transcript without satisfying Φ. However, we can convince Bob about Φ since we can only produce a correct transcript if we satisfy Φ as we do not know Bob's secret key. Furthermore, since a third party, Cindy, cannot distinguish between a transcript where Φ holds or ϕ_{Bob} holds, she reasonably rejects the proof and thus effectively stops a transfer of the conviction.

4 SYSTEM AND THREAT MODEL

Before delving into the protocol details, we present the considered setting and assumptions concerning the protocol participants.

4.1 System Model

We consider a network setting with four types of entities:

(1) **Holder (Helper)** is an untrusted, computationally capable device with a credential containing some attributes/claims and initially interacts with the issuer to obtain signatures over its credential. Then, on request, the helper collaborates with its core element to generate *designated verifier proofs of knowledge* of the issuer's signature over select attributes while optionally disclosing a subset of attributes and proving arbitrary relations on the remaining undisclosed attributes.
(2) **Holder (Core)** is a trusted and resource-constrained element belonging to a holder and is involved in that holder's initial credential issuance phase. Before a credential is issued to the holder's primary device (helper), the core element generates a fresh asymmetric keypair, whose public key is used in the issuer's signature over the credential to require the assistance of the core element in each presentation of the credential.
(3) **Designated Verifier** is a resource-constrained and potentially dishonest entity with a certified keypair who wishes to check whether a holder's credential satisfies some predicate.
(4) **Issuer** is a trusted entity with a certified keypair and is responsible for securely issuing credentials to holders, which includes: (i) verifying the correctness of the attributes claimed by a holder and (ii) guaranteeing that the involvement of the trusted core element of the specific holder is needed in the generation of credential presentations.

4.2 Threat Model

We assume that holders and designated verifiers are mutually distrusting. The holder assumes that the designated verifier might later attempt to profit from its credential presentations by leaking them to third parties to sell whatever information can be inferred from the underlying proof statement or disclosed attributes as being valid. Conversely, the designated verifier assumes that the holder will attempt to cheat in the proof generation by posing an invalid credential as valid or claiming that its credential satisfies the verifier's predicate when it does not.

4.3 Trust Model

As in [35], we assume that Cindy, a third party, will not trust Bob, a designated verifier, to have produced a proof $\pi = \Phi \vee \phi_{Bob}$, where ϕ_{Bob} is a proof of knowledge of Bob's secret key and Φ is some arbitrary proof statement (i.e., policy predicate) which the holder (Alice) wishes to prove the truth of. We also assume that Cindy is not an observer of the communication between a credential holder and the designated verifier. Due to space limitations, we elaborate on this decision in Appendix B. This assumption naturally holds in many use cases, especially those based on Distributed Ledger Technologies, where proofs might be stored on a blockchain.

4.4 Objectives

Let Φ be a predicate defined dynamically by a certified, designated verifier. Our protocol's overarching objectives are two-fold: (i) a designed verifier, Bob, always rejects transcripts for the proof of $\Phi \vee \phi_{Bob}$ unless Φ correctly holds on credentials issued by the trusted issuer and the core element of the corresponding credential holder was involved in producing the transcript, and (ii) a proof produced to convince a designated verifier, Bob, cannot be used later to convince another verifier, Cindy. Note that while we consider the helper potentially dishonest, a well-known problem in such a setting is that a corrupted helper can always break the privacy of an anonymous credential system, e.g., by adding identifying metadata. The core cannot check such de-anonymization attacks. Nevertheless, as in [32], we do not tolerate a malicious helper producing valid credential presentations without interacting with the core.

5 THE PROTOCOL

Overview of our solution. Fig. 1 shows a high-level work-flow of our scheme considering the different entities described in Section 4.1, which follows the core/helper model of [32]. Before obtaining verifiable credentials from the issuer, the core generates a cryptographic keypair (step 1), whose secret part never leaves the shielding of the core. The holder can now request credentials from the issuer (step 2), where the core is required to supply its public key together with a *proof of knowledge* of the secret key (step 3) to ensure that the core must be involved in all presentations of the issued credentials. After the issuing (step 4), the verifiable credentials are stored on the helper (step 5). Then, to check whether a holder in the system has credentials that satisfy some arbitrarily complex predicate, designated verifiers can craft a presentation request (step 6), which includes the predicate, describes the predicate's accepted attributes, and states which attributes should be disclosed. Supposing that a holder decides to answer a presentation request, it asks for its core's contribution (step 7) before completing the proof over the predicate (step 8) and finally sending the designated-verifier proof (and any disclosed attributes) to the verifier (step 9) who either

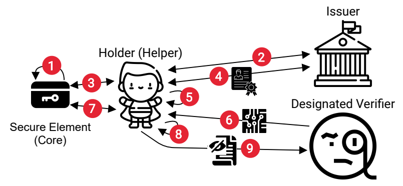

Figure 1: System model and conceptual work-flow.

accepts or rejects the proof. Note that steps 2 to 4 must occur over authentic channels, which can be realized in multiple ways. However, in this paper, we are not interested in how the issuer verifies the holder's claims before issuing authentic credentials nor in how we can establish an authentic channel during the issuance phase, i.e., we assume ideal functionality for the channel.

5.1 Building Blocks

We employ the cryptographic primitives described in Section 3 to create our protocol, which has several resemblances to the DAA with attributes (DAA-A) protocol proposed in [11]. Specifically, authors of [11] extend the BBS+ signature scheme described in Section 3.4 to require the public part of a TPM-generated secret in the issuer's BBS+ signature, thus requiring the TPM's contribution whenever the platform wants to produce *proofs of knowledge* of the signature. Similarly, we consider the same extension of the BBS+ signature, which we describe in Section 5.1.1, to require contributions from the trusted core element of credential holders in credential presentations. However, note that for clarity in presenting the primary objectives of this paper, we exclude the signature-based revocation and the use of pseudonyms from the protocol in [11], which we instead defer as extensions to decorate our protocol in Section 7.

Recall that the objective of this paper is: (i) extending the BBS+ scheme to support arbitrary predicates over attributes expressed as arithmetic circuits using cc-zkSNARKs and (ii) making credential presentations (i.e., proofs) *designated verifier*. However, note that we are working with two separate proofs. The first proof a holder must produce is a proof that its attributes satisfy the predicate using some sound cc-zkSNARK proof system, which returns, besides a proof of correctness of the predicate, a Pedersen commitment over the attributes that were passed as secret witnesses. The second proof is a *composite proof of knowledge* of a valid BBS+ signature where certain undisclosed attributes (as specified by the verifier) match the secret witnesses in the Pedersen commitment associated with the zkSNARK proof, thus proving that the attributes that satisfied the predicate originated from the issued credential. Note that both proofs must be made *designated verifier*. We describe how to make the former proof designated verifier in Section 5.1.2 and the latter in Section 5.1.3. We then put everything together in Section 5.3.

5.1.1 *Split BBS+ Signatures.*

We denote the extended BBS+ signature scheme, wherein the issuer commits to the trusted core's public key in its BBS+ signature, as a *split* (core/helper) BBS+ scheme. The scheme consist of three algorithms sBBS+ = (KeyGen, Sign, Verify) that work as follows. Note that the scheme's security follows [11].

- sBBS+.KeyGen$(L) \rightarrow (\mathbf{ick}, \mathsf{ipk}, \mathsf{isk})$: given a desired number of attributes L, take $\mathbf{ick} \leftarrow_{\$} \mathbb{G}_1^{L+2}$, $\mathsf{isk} \leftarrow_{\$} \mathbb{Z}_p^*$, $\mathsf{ipk} \leftarrow g_2^{\mathsf{isk}}$, and output $\mathbf{ick}$, ipk, and isk as the issuer's commitment key, public key, and secret key, respectively.

- sBBS+.Sign(isk, $\mathbf{ick}$, cpk, $(a_1, \ldots, a_L)) \to \sigma$: given a secret key, commitment key, a core's public key, and a sequence of attributes to sign, choose a random $e, s \leftarrow\$ \mathbb{Z}_p$, compute $A \leftarrow (g_1 \mathbf{ick}_0^s \mathsf{cpk} \prod_{i=1}^L \mathbf{ick}_{i+1}^{a_i})^{1/(e+\mathsf{isk})}$, and output $\sigma \leftarrow (A, e, s)$ as the multi-attribute BBS+ signature.
- sBBS+.Verify(ipk, $\mathbf{ick}$, cpk, $(a_1, \ldots, a_L), \sigma) \to b \in \{0,1\}$: in a public key, a commitment key, a core's public key, a sequence of attributes, and a purported signature, parse $\sigma = (A, e, s)$, and accept $(b = 1)$ the signature only if $A \neq 1_{\mathbb{G}_1}$ and $e(A, \mathsf{ipk} \cdot g_2^e) = e(g_1 \mathbf{ick}_0^s \mathsf{cpk} \prod_{i=1}^L \mathbf{ick}_{i+1}^{a_i}, g_2)$.

Before issuing verifiable BBS+ signatures to credential holders for credentials with a certain number of attributes L, an issuer must first create its keypair: $(\mathbf{ick}, \mathsf{ipk}, \mathsf{isk}) \leftarrow$ sBBS+.KeyGen(L) and register its public key and commitment key (i.e., generators in $\mathbb{G}_1$) at some trusted certificate authority as described in [11].

Assumptions. For brevity, we consider an already registered issuer. Furthermore, we assume that protocol participants have been equipped with the system parameters consisting of a security parameter λ, a bilinear group $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of prime order p with generators $g_1, \mathbf{ick}_0, \ldots, \mathbf{ick}_{L+1}$ of $\mathbb{G}_1$ and g_2 of $\mathbb{G}_2$ and a bilinear map e, generated via $\mathcal{G}(1^\lambda)$. We further assume a random oracle $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$, which is used for the Fiat-Shamir heuristic [28] to make non-interactive zero-knowledge proofs in the random oracle model.

PoK of BBS+ Signature. Let $D \subset \{1, \ldots, L\}$ denote the selection of attribute indices that a holder wants to disclose as part of its proof of knowledge and $U = \{1, \ldots, L\} \setminus D$ denote the set of undisclosed attributes. To prove knowledge of a BBS+ signature while selectively disclosing attributes a_i with $i \in D$, the holder first computes $b \leftarrow g_1 \mathbf{ick}_0^s \mathsf{cpk} \prod_{i=1}^L \mathbf{ick}_{i+1}^{a_i}$ and then proceeds as follows. Randomize the credential by taking $r_1 \leftarrow\$ \mathbb{Z}_p^*$, set $A' \leftarrow A^{r_1}$, and set $r_3 \leftarrow 1/r_1$. Set $\overline{A} \leftarrow A'^{-e} \cdot b^{r_1} (= A'^{\mathsf{isk}})$. Choose $r_2 \leftarrow\$ \mathbb{Z}_p$, set $d \leftarrow b^{r_1} \cdot \mathbf{ick}_0^{-r_2}$, and set $s' \leftarrow s - r_2 \cdot r_3$. The holder now proves knowledge of a BBS+ signature following (1).

$$\pi \in PoK\{(\mathsf{csk}, \{a_i\}_{i \in U}, e, r_2, r_3, s') :$$
$$\overline{A}/d = A'^{-e} \mathbf{ick}_0^{r_2} \wedge g_1 \prod_{i \in D} \mathbf{ick}_{i+1}^{a_i} = d^{r_3} \mathbf{ick}_0^{-s'} \mathbf{ick}_1^{-\mathsf{csk}} \prod_{i \in U} \mathbf{ick}_{i+1}^{-a_i}\} \tag{1}$$

The resulting proof is $(A', \overline{A}, d, \pi)$. To verify a proof, the verifier checks $A' \neq 1_{\mathbb{G}_1}$, $e(A', \mathsf{ipk}) = e(\overline{A}, g_2)$, and verifies the proof π.

5.1.2 Designated Verifier Circuits. Recall that a trapdoor allows us to simulate a valid proof without knowing the satisfying witness, and the simulated proof is indistinguishable from a "real" proof. While typically, we would not want a prover to have access to a trapdoor, our scheme depends on trapdoors to make proofs *designated verifier.* Without a trapdoor, a proof that a predicate (i.e., circuit) was satisfied can be very revealing to third parties for which the proof was not originally intended. This is particularly true for circuits representing more complex functions, such as membership checks, knowledge of preimages (secrets), and range checks.

While it might be ridiculous to include a trapdoor inside simple circuits where anyone knows a satisfying witness, e.g., a circuit that only includes a range check, more complex circuits must include a trapdoor since satisfying witnesses are not publicly known. We continue by first defining designated verifier circuits in Definition 5.1, and then proceed by giving three examples of trapdoor functions with Bob as the contextual designated verifier. (We compare the performance of each trapdoor in our evaluation in Section 6.)

Definition 5.1 (Designated Verifier Circuit). Let C be a circuit for the function f on public input x and secret witness w, i.e., $\mathsf{C}(x, w)$ holds only if $f(x, w)$ holds. We define a *designated verifier circuit*, DVC, as a circuit that accepts $x = (x_1, x_2)$ and $w = (u, \omega)$, and includes a second function h, which we call the designated trapdoor, such that $\mathsf{DVC}(x, w)$ holds either if $f(x_1, u)$ holds *or* $h(x_2, \omega)$ holds.

Example 5.2 (Trapdoor #1: PoK of RSA secret key). Bob has a certified RSA keypair with some public modulus n. We can define our designated trapdoor h in our circuit $\mathsf{DVC}((x_1, x_2), (u, \omega))$ as: $h(n, p, q) : p \times q = n$, where p and q are passed as secret witnesses to the circuit in ω and n is passed in x_2. Thus, anyone besides Bob who sees a proof with n as a public input will reject it since Bob might have cheated. Conversely, anyone except Bob can only create a valid proof over the circuit with n in x_2 if they know a satisfying witness to $f(x_1, u)$ since they do not know Bob's secret key.

Example 5.3 (Trapdoor #2: PoK of EC secret key). Bob has a certified EC keypair with public key $y = g^x$. We can define our designated trapdoor h in our circuit $\mathsf{DVC}((x_1, x_2), (u, \omega))$ as: $h(y, x) : y = g^x$, where x is passed as a secret witness to the circuit in ω and y is passed in x_2. The conviction follows that of example 5.2.

Example 5.4 (Trapdoor #3: PoK of preimage). Bob has a secret value x with a certified image $y = \mathsf{H}(x \,\|\, r)$, where H is a sound hashing function, and r is some secret blinding factor initially supplied by Bob. We can define our designated trapdoor h in our circuit $\mathsf{DVC}((x_1, x_2), (u, \omega))$ as: $h(y, x, r) : y = \mathsf{H}(x \,\|\, r)$, where x and r are passed as secret witnesses to the circuit in ω and y is passed in x_2. The conviction follows that of example 5.2.

In practice, we need assurance that a predicate includes a trapdoor to which the designated verifier has access. For example, consider a circuit C with a supposed trapdoor from Example 5.3 and includes ek and vk as its evaluation and verification keys, respectively. We can determine the existence of this trapdoor with (2).

$$\begin{aligned} &(\mathsf{pk}', \mathsf{sk}') \leftarrow\$ \mathsf{KGen}(1^\lambda) \\ &(t', \pi', o') \leftarrow \mathsf{cc\Pi.Prove}(\mathsf{C}, \mathsf{ek}, (x, \mathsf{pk}'), (\{0,1\}^W, \mathsf{sk}')) \\ &\qquad \mathsf{cc\Pi.VerProof}(\mathsf{C}, \mathsf{vk}, (x, \mathsf{pk}'), t', \pi') \overset{?}{=} 1 \end{aligned} \tag{2}$$

If (2) holds, the circuit is designated verifier according to Definition 5.1. However, note that such checks require an extra execution of the proof generation for a circuit, which can be expensive for large circuits. Thus, in practice, we might outsource such trapdoor verification either to a certification authority or distributed worker farms [52] that return proof about the presence of the trapdoor. For brevity, in the remaining paper, we denote by DVC a circuit that has been verified to be a designated verifier circuit.

5.1.3 Designated Verifier Sigma Protocols. Let us assume that we have a credential comprising L attributes: $(a_1, \ldots, a_L)$. Let D denote the attribute indices we are supposed to disclose. The remaining undisclosed attribute indices are $U = \{1, \ldots, L\} \setminus D$. Furthermore, let

DVC be a circuit for which we want to prove to the designated verifier that we have satisfying witnesses. Finally, let $\boldsymbol{u} = (u_1, \ldots, u_W)$ specify the sequence of attribute indices that we must supply as witnesses to the circuit, where $\boldsymbol{u}_i \in U$ and W denotes the slice of committed witnesses as determined during the circuit's key generation, i.e., the circuit's commitment key $\mathbf{ck}$ contains $W+1$ generators.

To produce a designated-verifier proof over the circuit using the specified attributes from our credential, where dvpk is the public key of the designated verifier, and x is some public input, we run:

$$(t_u, \pi, o) \leftarrow \mathsf{cc}\Pi.\mathsf{Prove}(\mathsf{DVC}, \mathsf{ek}, (x, \mathsf{dvpk}), ((a_{\boldsymbol{u}_i})_{i\in[W]}, \omega))$$

To prove knowledge of the secret ingredients of the Pedersen commitment while also allowing the designated verifier to produce valid proofs, we form the following disjunctive proof statement:

$$\pi \in PoK\{(\{a_i\}_{i\in\boldsymbol{u}}, o) \vee \mathsf{dvsk} : t_u = \mathbf{ck}_0^o \prod_{i=1}^{W} \mathbf{ck}_i^{a_{u_i}} \vee \mathsf{dvpk} = g^{\mathsf{dvsk}}\} \tag{3}$$

Then, to produce non-interactive proofs that we know of a solution to one of the problems without anyone learning which solution we know, we use the idea described in [15], which is based on the initial result proposed by Cramer et al. [22]. The idea is based on the generalization of the Schnorr protocol [44] (described in Appendix A), where we execute the two proof systems in parallel, but we additionally allow the prover to "cheat" in one of them in an indistinguishable manner, as follows (computations are done modulo the curve order p). Pick $r_{a_i} \leftarrow\$\, \mathbb{Z}_p$ for $i \in \boldsymbol{u}$, $r_o, r_{\mathsf{dvsk}} \leftarrow\$\, \mathbb{Z}_p$, and a random (cheating) challenge $c_2 \leftarrow\$\, \mathbb{Z}_p$ for the second term of (3) since we are not the designated verifier. We then compute the two commitments: $t_1 \leftarrow \mathbf{ck}_0^{r_o} \prod_{i=1}^{W} \mathbf{ck}_i^{r_{a_{u_i}}}$ and $t_2 \leftarrow g^{r_{\mathsf{dvsk}}} \cdot \mathsf{dvpk}^{c_2}$. To get the challenge for the first term, c_1, we first compute $c \leftarrow \mathsf{H}(t_1 \,\|\, t_2 \,\|\, \mathbf{ck} \,\|\, g \,\|\, t_u \,\|\, \mathsf{dvpk})$ and then $c_1 \leftarrow c - c_2$. We can then compute the challenge responses as: $s_{a_i} \leftarrow r_{a_i} + c_1 a_i$ for $i \in \boldsymbol{u}$, $s_o \leftarrow r_o + c_1 o$, and $s_{r_{\mathsf{dvsk}}} \leftarrow r_{\mathsf{dvsk}}$. Our final proof transcript for (3) is $\pi' \leftarrow (c_1, c_2, t_u, \{s_{a_i}\}_{i\in\boldsymbol{u}}, s_o, s_{r_{\mathsf{dvsk}}})$, which we give to the designated verifier together with the zkSNARK proof π.

To verify that π' is a valid transcript for (3), where $t_u \in \pi'$ is a satisfying commitment to π, i.e., $\mathsf{cc}\Pi.\mathsf{VerProof}(\mathsf{DVC}, \mathsf{vk}, (x, \mathsf{dvpk}), t_u, \pi) = 1$ holds, the verifier first reconstructs the commitments: $t_1' \leftarrow \mathbf{ck}_0^{s_o} \cdot (\prod_{i=1}^{W} \mathbf{ck}_i^{s_{a_{u_i}}}) \cdot t_u^{-c_1}$ and $t_2' \leftarrow g^{s_{r_{\mathsf{dvsk}}}} \cdot \mathsf{dvpk}^{c_2}$, and then checks that $c_1 + c_2 = \mathsf{H}(t_1' \,\|\, t_2' \,\|\, \mathbf{ck} \,\|\, g \,\|\, t_u \,\|\, \mathsf{dvpk})$ holds. Note that the only missing ingredient of (3) is proving that the attributes originated from a valid BBS+ signature, i.e., that the discrete logarithms of the Pedersen commitment t_u match the attribute indices specified by $\boldsymbol{u}$ in a valid credential, which we explain in Section 5.3.

5.2 Core/Helper Credential Issuance

As described in Section 5.1.1, we assume that protocol participants have been equipped with the system parameters and the issuer's trusted cryptographic materials. Fig. 2 shows the issuance protocol initiated by a holder that wishes to get the issuer's BBS+ signature over its claims. After the issuer has performed all the necessary steps to verify the authenticity of the holder's claims, it opens a communication channel with the holder's trusted core element. Here the issuer ensures the core's active participation whenever the helper wishes to produce *proofs of knowledge* of the issuer's BBS+ signature over the holder's claims. To do so, the issuer sends a fresh challenge to the core, requesting the core to generate a fresh key pair whose public key should be included in the BBS+ signature. Note that by including the core's public key in the signature, we prevent the helper from independently producing *proofs of knowledge* of the signature since only the core can produce the necessary contributions in the Schnorr protocol to prove knowledge of the public key's discrete logarithm, i.e., the core secret key csk in (1). Like [11], we assume that this communication between the issuer and the core occurs over an authenticated channel and, similarly, that there is a secure channel between the holder and its core.

When the core has created its key pair, it produces a *signed proof of knowledge* that it knows the secret key behind its public key: $\pi \leftarrow\$\, SPoK\{(\mathsf{csk}) : \mathsf{cpk} = \mathbf{ick}_1^{\mathsf{csk}}\}(n)$, where n is the verifier's fresh challenge. The proof and public key are then sent to the issuer, who, after verifying the proof, proceeds to use its secret key to generate a BBS+ signature over the holder's claims that also incorporates the core's public key. Finally, given the issuer's signature, the holder checks the signature's validity before storing it in persistent storage. The holder also stores the product the issuer signed to ease the computational effort needed to produce credential presentations.

5.3 Designated Verifier Credential Presentations

To demonstrate the protocol, let us consider a simple setting where a credential holder knows of a designated verifier, and the designated verifier wants to determine whether the holder has valid credentials whose attributes satisfy some designated verifier circuit DVC. Like Section 5.1.3, we consider credentials with L attributes, where the order and meaning of the attributes are known[1]. Let D denote the attribute indices the designated verifier wants to be disclosed such that $U = \{1, \ldots, L\} \setminus D$ contains the remaining undisclosed attribute indices. Finally, let $\boldsymbol{u} = (u_1, \ldots, u_W)$ specify the sequence of attribute indices that should be passed as circuit witnesses, with $\boldsymbol{u}_i \in U$. Fig. 3 shows the participants engage in the protocol.

Like in Section 5.1.3, to prove that the specified attributes satisfy the predicate, the holder generates a commit-carrying zkSNARK proof over the circuit with the specified selection of attributes as the *committed* part of the witness:

$$(t_u, \pi', o) \leftarrow \mathsf{cc}\Pi.\mathsf{Prove}(\mathsf{DVC}, \mathsf{ek}, (x, \mathsf{dvpk}), ((a_{\boldsymbol{u}_i})_{i\in[W]}, \omega))$$

Then, to generate a designated-verifier zero-knowledge proof of a valid BBS+ signature while proving that the committed witnesses t_u from the zkSNARK proof equal the attributes specified by $\boldsymbol{u}$ in the signed credential, we merge the proof statements of (1) and (3), resulting in (4). The holder first randomizes its BBS+ signature as described in Section 5.1.1 and then produces a proof of (4):

$$\begin{aligned}\pi \in SPoK\Bigg\{&(\mathsf{csk}, \{a_i\}_{i\in U}, e, r_2, r_3, s', o) \vee \mathsf{dvsk} : \Big(\\ &\overline{A}/d = A'^{-e}\mathbf{ick}_0^{r_2} \wedge g_1 \prod_{i\in D} \mathbf{ick}_{i+1}^{a_i} = d^{r_3}\mathbf{ick}_0^{-s'}\mathbf{ick}_1^{-\mathsf{csk}} \prod_{i\in U} \mathbf{ick}_{i+1}^{-a_i}\\ &\wedge t_u = \mathbf{ck}_0^o \prod_{i=1}^{W} \mathbf{ck}_i^{a_{u_i}}\Big) \vee \Big(\mathsf{dvpk} = g^{\mathsf{dvsk}}\Big)\Bigg\}(n)\end{aligned} \tag{4}$$

[1]In practice, the credentials follow richer and standardized formats (e.g., the schema defined by W3C [50]), where we also encounter serialization challenges.

Holder (Core)		**Holder (Helper)**		**Issuer**
$\mathbf{ick}$		$(\mathbf{ick}, \mathsf{ipk}, g_1, g_2), (a_i)_{i\in[L]}$		$\mathbf{ick}, \mathsf{ipk}, \mathsf{isk}, g_1 \in \mathbb{G}_1, g2 \in \mathbb{G}_2$
		Ask Issuer to sign credentials	$\xrightarrow{(a_i)_{i\in[L]}}$	$n \leftarrow\$ \{0,1\}^\lambda$
$\mathsf{csk} \leftarrow\$ \mathbb{Z}_p, \mathsf{cpk} \leftarrow \mathbf{ick}_1^{\mathsf{csk}}$	$\xleftarrow{n}$	- -	$\xleftarrow{n}$	
$r \leftarrow\$ \mathbb{Z}_p$				
$t \leftarrow \mathbf{ick}_1^r$				
$c \leftarrow \mathsf{H}(\mathbf{ick}_1 \parallel \mathsf{cpk} \parallel t \parallel n)$				
$s \leftarrow r - \mathsf{csk} \cdot c$	$\xrightarrow{\pi = (c,s), \mathsf{cpk}}$	- -	$\xrightarrow{\pi = (c,s), \mathsf{cpk}}$	$t' \leftarrow \mathbf{ick}_1^s \mathsf{cpk}^c$
				$c' \leftarrow \mathsf{H}(\mathbf{ick}_1 \parallel \mathsf{cpk} \parallel t' \parallel n)$
				$c' \overset{?}{=} c$
		$\mathsf{sBBS+.Verify}(\mathsf{ipk}, \mathbf{ick}, \mathsf{cpk}, (a_i)_{i\in[L]}, \sigma) \overset{?}{=} 1$	$\xleftarrow{\sigma = (A,e,s)}$	$\sigma \leftarrow \mathsf{sBBS+.Sign}(\mathsf{isk}, \mathbf{ick}, \mathsf{cpk}, (a_i)_{i\in[L]})$
		store $b \leftarrow g_1 \mathbf{ick}_0^s \mathsf{cpk} \prod_{i=1}^{L} \mathbf{ick}_{i+1}^{a_i}$		

Figure 2: Credential issuance using the split BBS+ signature scheme.

This statement is a disjunction of two outer relations. The first outer relation is a conjunction of three inner relations, which the holder attempts to prove together with its core, and the second is for the designated verifier. Note that we create the outer disjunction following the idea in Section 5.1.3. Specifically, we execute the Schnorr proof of knowledge protocol for each of the four relations to prove knowledge of the different discrete logarithms. However, we use a different half of the challenge in each of the outer relations.

Finally, to verify that π is a valid transcript for (4), where t_u is a satisfying commitment to π', i.e., $\mathsf{cc\Pi.VerProof}(\mathsf{DVC}, \mathsf{vk}, (x, \mathsf{dvpk}), t_u, \pi') = 1$ holds, the verifier reconstructs and verifies the four commitments as shown in Fig. 3. Then, to verify the accompanying, randomized BBS+ signature $(A', \overline{A}, d)$ against the issuer's public key, the verifier checks that $A' \neq 1_{\mathbb{G}_1}$ and $e(A', \mathsf{ipk}) = e(\overline{A}, g_2)$.

Note that if the holder attempts to use an invalid sequence of attributes to satisfy the predicate (i.e., one not specified by $\boldsymbol{u}$), then the proof will be rejected. Specifically, since the verifier's reconstruction of the t_3' commitment considers the s-values corresponding to the correct sequence of attributes as specified by $\boldsymbol{u}$, the reconstructed commitment would inevitably differ from t_u, producing a different challenge c' and thus causing the proof to be rejected.

5.3.1 Simulating Transcripts. Demonstrating how the protocol is *designated verifier*, Fig. 4 shows the designated verifier producing valid transcripts for arbitrary attributes using the demonstrative trapdoor in the circuit and satisfying the latter outer relation in (4).

6 PERFORMANCE EVALUATION

Our evaluation aims to answer the questions of (i) how efficient our protocol is for creating credential presentations and (ii) how costly the considered method is for making it *designated verifier*.

Experimental setup. To program demonstrative circuits, we used xJsnark [37], a high-level code-to-circuit compilation framework that employs a mix of optimizations to minimize circuit complexity. With xJsnark, our high-level code is compiled into low-level circuits, which, using the jsnark interface [36], are translated into the R1CS constraint system and fed into the libsnark [38] backend for instantiating a particular zkSNARK proof system over the circuit. In our case, we considered an implementation [39] of the LegoGro16 [17] proof system (over the BN254 curve) for producing commit-carrying zkSNARK proofs. The specific LegoGro16 implementation is essentially a commit-and-prove variant of libsnark's implementation of Groth16 [30]. However, since we only require the zkSNARK to be commit-carrying, we assume the complexity of the commit-carrying variant here. As our testbed (holder), we considered a machine with an AMD Ryzen 7 3700X processor and 16 GB of memory (experiments were conducted in a WSL2 environment).

Notation. By $k\mathbb{G}_i$, we denote k exponentiations (scalar multiplications) in the group $\mathbb{G}_i$, by $k\mathbb{G}_i^j$ we denote k j-multi exponentiations, and by kP we denote k pairing operations. We let W denote the number of witnesses committed to in the considered commit-carrying zkSNARK proof system, L denote the number of attributes, and D and U denote the number of disclosed and undisclosed attributes, respectively. Finally, to reason about the computational complexity incurred by varying arithmetic circuit sizes, we consider circuits as R1CS instances where we use M to denote the number of constraints and N to denote the number of variables in the instance.

Asymptotic performance. Table 1 shows the computational complexity of our construction described in Section 5.3 and includes a comparison with other related core/helper credential protocols as described in Section 2. Note, however, that *none* of the other schemes consider *designated verifiers* nor zkSNARKs. Therefore, we split our protocol's effort over two rows for comparison purposes.

The difference between ours and the two DAA-A schemes is that we are not using pseudonyms and we have two challenges (for the disjunction). Like CHAC [32], our core only needs to compute a single EC scalar multiplication, regardless of the number of attributes. Furthermore, note that for the considered curve, Groth16's proof is 127 bytes and contains 3 group elements (2 $\mathbb{G}_1$ elements and 1 $\mathbb{G}_2$ element), and 3 pairings dominate verification. However, with the considered commit-carrying variant, we additionally have a Pedersen commitment to the W witnesses in $\mathbb{G}_1$.

Holder (Core)		**Holder (Helper)**		**Designated Verifier**
$\mathbf{ick}$, (csk, cpk)		$(\mathbf{ick}, \mathsf{ipk}, g_1, g_2)$, $(a_i)_{i \in [L]}$, $\sigma = (A, e, s)$, b, dvpk		$(\mathsf{dvsk}, \mathsf{dvpk} = g^{\mathsf{dvsk}})$, $(\mathbf{ick}, \mathsf{ipk}, g_1, g_2)$, $(\mathsf{DVC}, \mathbf{ck}, \mathsf{ek}, \mathsf{vk})$, x
				$D \subset \{1, \ldots, L\}$ **attribute indices to disclose**
		Generate cc-zkSNARK proof		$\boldsymbol{u} = (u_1, \ldots, u_W)$ **indices to pass as witness to circuit**
		$w \leftarrow ((a_{\boldsymbol{u}_i})_{i \in [W]}, \omega)$	$\xleftarrow{(\mathsf{DVC}, \mathbf{ck}, \mathsf{ek}), D, \boldsymbol{u}, x, n}$	$n \leftarrow\$ \{0,1\}^\lambda$
		$(t_u, \pi', o) \leftarrow \mathsf{cc}\Pi.\mathsf{Prove}(\mathsf{DVC}, \mathsf{ek}, (x, \mathsf{dvpk}), w)$		
		Randomize BBS+ credential		
		$r_1 \leftarrow\$ \mathbb{Z}_p^*, r_2 \leftarrow\$ \mathbb{Z}_p, r_3 \leftarrow \frac{1}{r_1}$		
		$A' \leftarrow A^{r_1}$		
		$\overline{A} \leftarrow A'^{-e} \cdot b^{r_1}$		
		$d \leftarrow b^{r_1} \cdot \mathbf{ick}_0^{-r_2}$		
		$s' \leftarrow s - r_2 r_3$		
		Schnorr PoK of discrete logarithms		
		$r_{a_i} \leftarrow\$ \mathbb{Z}_p$ for $i \in U = \{1, \ldots, L\} \setminus D$		
		$r_e, r_{r_2}, r_{r_3}, r_{s'}, r_o, r_{\mathsf{dvsk}}, c_2 \leftarrow\$ \mathbb{Z}_p$		
$r_{\mathsf{csk}} \leftarrow\$ \mathbb{Z}_p$	$\longleftarrow$	**get Schnorr commitment for secret key**		
$t_{\mathsf{csk}} \leftarrow \mathbf{ick}_1^{r_{\mathsf{csk}}}$	$\xrightarrow{t_{\mathsf{csk}}}$	$t_1 \leftarrow A'^{r_e} \cdot \mathbf{ick}_0^{r_{r_2}}$		
		$t_2 \leftarrow d^{r_{r_3}} \cdot \mathbf{ick}_0^{r_{s'}} \cdot t_{\mathsf{csk}} \prod_{i \in U} \mathbf{ick}_{i+1}^{r_{a_i}}$		
		$t_3 \leftarrow \mathbf{ck}_0^{r_o} \cdot \prod_{i=1}^{W} \mathbf{ck}_i^{r_{a_{\boldsymbol{u}_i}}}$		
		$t_4 \leftarrow g^{r_{\mathsf{dvsk}}} \cdot \mathsf{dvpk}^{c_2}$		
		$c \leftarrow \mathsf{H}(n \parallel A' \parallel \overline{A} \parallel d \parallel t_1 \parallel t_2 \parallel t_3 \parallel t_4 \parallel t_u \parallel$ $g \parallel g_1 \parallel D \parallel \boldsymbol{u} \parallel \mathbf{ck} \parallel \mathbf{ick} \parallel \mathsf{ipk} \parallel \mathsf{dvpk})$		
		$c_1 \leftarrow c - c_2$		
$s_{\mathsf{csk}} \leftarrow r_{\mathsf{csk}} + c_1 \cdot \mathsf{csk}$	$\xleftarrow{c_1}$	**get Schnorr response for secret key**		
	$\xrightarrow{s_{\mathsf{csk}}}$	$s_{a_i} \leftarrow r_{a_i} + c_1 a_i$ for $i \in U$		
		$s_e \leftarrow r_e + c_1 e$		
		$s_{r_2} \leftarrow r_{r_2} - c_1 r_2$		
		$s_{r_3} \leftarrow r_{r_3} - c_1 r_3$		
		$s_{s'} \leftarrow r_{s'} + c_1 s'$		
		$s_o \leftarrow r_o + c_1 o$		
		$s_{r_{\mathsf{dvsk}}} \leftarrow r_{\mathsf{dvsk}}$		
		$\pi \leftarrow (c_1, c_2, \pi', t_u, s_{\mathsf{csk}}, \{s_{a_i}\}_{i \in U}, s_e, s_{r_2}, s_{r_3},$ $s_{s'}, s_o, s_{r_{\mathsf{dvsk}}})$	$\xrightarrow{\{A', \overline{A}, d, \pi, \{a_i\}_{i \in D}\}_{\mathsf{dvpk}}}$	**Verify that** $\mathsf{cc}\Pi.\mathsf{VerProof}(\mathsf{DVC}, \mathsf{vk}, (x, \mathsf{dvpk}), t_u, \pi') \overset{?}{=} 1$
				$t'_1 \leftarrow A'^{s_e} \cdot \mathbf{ick}_0^{s_{r_2}} \cdot (\overline{A}/d)^{c_1}$
				$t'_2 \leftarrow d^{s_{r_3}} \cdot \mathbf{ick}_0^{s_{s'}} \cdot \mathbf{ick}_1^{s_{\mathsf{csk}}} \cdot (\prod_{i \in U} \mathbf{ick}_{i+1}^{s_{a_i}}) \cdot (g_1 \prod_{i \in D} \mathbf{ick}_{i+1}^{a_i})^{c_1}$
				$t'_3 \leftarrow \mathbf{ck}_0^{s_o} \cdot (\prod_{i=1}^{W} \mathbf{ck}_i^{s_{a_{\boldsymbol{u}_i}}}) \cdot t_u^{-c_1}$
				$t'_4 \leftarrow g^{s_{r_{\mathsf{dvsk}}}} \cdot \mathsf{dvpk}^{c_2}$
				$c' \leftarrow \mathsf{H}(n \parallel A' \parallel \overline{A} \parallel d \parallel t'_1 \parallel t'_2 \parallel t'_3 \parallel t'_4 \parallel t_u \parallel$ $g \parallel g_1 \parallel D \parallel \boldsymbol{u} \parallel \mathbf{ck} \parallel \mathbf{ick} \parallel \mathsf{ipk} \parallel \mathsf{dvpk})$
				Verify that $c' \overset{?}{=} c_1 + c_2$, $A' \overset{?}{\neq} 1_{\mathbb{G}_1}$ **and** $e(A', \mathsf{ipk}) \overset{?}{=} e(\overline{A}, g_2)$

Figure 3: Holder proves knowledge of a valid BBS+ signature whose undisclosed attributes satisfy some arbitrary predicate only to a designated verifier. Since the designated verifier can also create the proof using the trapdoors, it is worthless to anyone else.

Empirical performance. To determine the cost of the different trapdoors mentioned in Section 5.1.2, we used xJsnark to generate the corresponding circuits. As also reported at [2], it costs only 2578 constraints to express a function for verifying knowledge of the two secret prime factors of a 2048-bit RSA key's modulus as an arithmetic circuit. However, proving knowledge of an ECDSA secret key costs almost 700K constraints if we consider the NIST P-256 curve. For the third choice of a trapdoor, we mentioned that we could express a hashing function in the circuit and then prove knowledge of the secret preimage. In our case, we considered the recent permutation function called Poseidon [29] that was made to be expressed inexpensively in an arithmetic circuit. With our implementation of Poseidon, it cost only 241 constraints when considering an arity of 2 and 262 constraints for an arity of 3, which allows for preimages fitting three field elements, i.e., a preimage bitwidth of 762 bits considering the BN254 curve. Note that from our

Table 1: Comparison of our scheme's complexity to similar schemes when creating credential presentations by the holder (helper) and its secure element (core) and verification by the verifier.

Scheme	Core	Holder (helper)	Verifier	Credential size	Presentation size
DAA-A [11]	$3\mathbb{G}_1$	$\mathcal{O}(U\mathbb{G}_1)$	$\mathcal{O}(L\mathbb{G}_1) + 2P$	$2\mathbb{Z}_p + 1\mathbb{G}_1$	$\mathcal{O}(U\mathbb{Z}_p) + 4\mathbb{G}_1 + 1\lambda$
DAA-A [10]	$3\mathbb{G}_1$	$\mathcal{O}(U\mathbb{G}_1)$	$\mathcal{O}(L\mathbb{G}_1) + 2P$	$2\mathbb{Z}_p + 1\mathbb{G}_1$	$\mathcal{O}(U\mathbb{Z}_p) + 4\mathbb{G}_1 + 1\lambda$
CHAC [32]	$1\mathbb{G}_1$	$\mathcal{O}(D(\mathbb{G}_1 + \mathbb{G}_2))$	$\mathcal{O}(DP)$	$\mathcal{O}(L(\mathbb{G}_1 + \mathbb{G}_2))$	$6\mathbb{G}_1 + 3\mathbb{G}_2$
This work (sBBS+)	$1\mathbb{G}_1$	$\mathcal{O}(U\mathbb{G}_1)$	$\mathcal{O}(L\mathbb{G}_1) + 2P$	$2\mathbb{Z}_p + 1\mathbb{G}_1$	$\mathcal{O}(U\mathbb{Z}_p) + 3\mathbb{G}_1 + 2\lambda$
This work (ccGroth16)		$\mathcal{O}(2W\mathbb{G}_1) + \mathcal{O}((3N + M)\mathbb{G}_1 + N\mathbb{G}_2)$	$\mathcal{O}(W\mathbb{G}_1) + 3P$		$3\mathbb{G}_1 + 1\mathbb{G}_2$

```
Simulate Transcript$((A', \overline{A}, d), (\mathbf{ick}, \mathsf{ipk}, g_1), (\mathsf{DVC}, \mathbf{ck}), D, \boldsymbol{u}, x, n)$
$(a_1, \ldots, a_L) \leftarrow\$ \mathbb{Z}_p^L$
$w \leftarrow ((a_{\boldsymbol{u}_i})_{i \in [W]}, \mathsf{dvsk})$
$(t_{\boldsymbol{u}}, \pi', o) \leftarrow \mathsf{cc\Pi.Prove}(\mathsf{DVC}, \mathsf{ek}, (x, \mathsf{dvpk}), w)$
$r_{a_i} \leftarrow\$ \mathbb{Z}_p$ for $i \in U = \{1, \ldots, L\} \setminus D$
$r_e, r_{r_2}, r_{r_3}, r_{s'}, r_{\mathsf{csk}}, r_o, r_{\mathsf{dvsk}}, c_1 \leftarrow\$ \mathbb{Z}_p$
$t_1 \leftarrow A'^{r_e} \cdot \mathbf{ick}_0^{r_{r_2}} \cdot (\overline{A}/d)^{c_1}$
$t_2 \leftarrow d^{r_{r_3}} \cdot \mathbf{ick}_0^{r_{s'}} \cdot \mathbf{ick}_1^{r_{\mathsf{csk}}} \cdot (\prod_{i \in U} \mathbf{ick}_{i+1}^{r_{a_i}}) \cdot (g_1 \prod_{i \in D} \mathbf{ick}_{i+1}^{a_i})^{c_1}$
$t_3 \leftarrow \mathbf{ck}_0^{r_o} \cdot (\prod_{i=1}^{W} \mathbf{ck}_i^{r_{a_{\boldsymbol{u}_i}}}) \cdot t_{\boldsymbol{u}}^{-c_1}$
$t_4 \leftarrow g^{r_{\mathsf{dvsk}}}$
$c \leftarrow \mathsf{H}(n \parallel A' \parallel \overline{A} \parallel d \parallel t_1 \parallel t_2 \parallel t_3 \parallel t_4 \parallel t_{\boldsymbol{u}} \parallel$
        $g \parallel g_1 \parallel D \parallel \boldsymbol{u} \parallel \mathbf{ck} \parallel \mathbf{ick} \parallel \mathsf{ipk} \parallel \mathsf{dvpk})$
$c_2 \leftarrow c - c_1$
$s_{a_i} \leftarrow r_{a_i}$ for $i \in U$
$(s_e, s_{r_2}, s_{r_3}, s_{s'}, s_{\mathsf{csk}}, s_o) \leftarrow (r_e, r_{r_2}, r_{r_3}, r_{s'}, r_{\mathsf{csk}}, r_o)$
$s_{\mathsf{dvsk}} \leftarrow r_{\mathsf{dvsk}} - c_2 \cdot \mathsf{dvsk}$
$\pi \leftarrow (c_1, c_2, \pi', t_{\boldsymbol{u}}, s_{\mathsf{csk}}, \{s_{a_i}\}_{i \in U}, s_e, s_{r_2}, s_{r_3}, s_{s'}, s_o, s_{r_{\mathsf{dvsk}}})$
return $(A', \overline{A}, d, \pi, \{a_i\}_{i \in D})$
```

Figure 4: Designated verifier simulating correct transcripts.

timings of generating proofs with libsnark, it was evident that we prove the satisfaction of circuits at a rate of $\approx$ 77.8 constraints/ms on our considered setup. Finally, computing the three pairings during proof verification took approximately 2 ms using libsnark.

7 SECURITY PROPERTIES AND EXTENSIONS

Besides its secure implementation, the proposed RETRACT scheme's foundational security is guaranteed by the security of the underlying BBS+ signature scheme [11] and zkSNARK proof system [17, 30]. In the following, we give an intuitive description of the different security properties our scheme is designed to provide and how it achieves them. Note that we only cover essential properties here.

Unforgeability. As described in Section 5.3, the verifier will reject a proof if the holder uses attributes whose indices were not specified by $\boldsymbol{u}$. Thus, the holder cannot cheat by choosing a different sequence of attributes to satisfy the predicate. Furthermore, since a verifier only accepts a proof of (4) if the corresponding randomized signature is valid under the issuer's public key, only the actual holder of a credential issued by the trusted issuer can produce correct transcripts that a verifier accepts. Therefore, an entity cannot use forged or otherwise invalid credentials to convince a verifier about satisfying a predicate. Nor is it possible for an entity to present attributes that it does not possess believably.

Designated verifier. Only the designated verifier can be convinced by a proof produced by a holder since the verifier can produce indistinguishable transcripts, as demonstrated in Fig. 4. While malicious verifiers might attempt to infringe on the privacy of credential holders by crafting predicates to lure out excessive information, we note that proofs are always limited in convincing a particular verifier. Furthermore, similar to the revocation of credentials, it is possible to castrate such misbehaving verifiers in an established credential system, or we could slightly modify the system setup to require all predicates to be certified by trusted parties.

Unlinkability & Selective Disclosure. Inherent to anonymous credentials, different credential presentations from the same holder should not be linkable, and it should be possible to disclose attributes in a verifiable manner selectively. Both of these properties are guaranteed by the underlying BBS+ signature scheme.

Dependability. Like [32], regardless of whether credentials are leaked or a helper device is completely compromised, it should only be possible to produce valid credential presentations with assistance from the trusted core associated with the holder for which the credentials were initially issued. This property is guaranteed by the split BBS+ signature scheme described in Section 5.1.1.

Optional revocation. An important feature to effectively handle the dynamic nature of the set of entities of a credential system is the possibility of revoking the credentials of misbehaving parties. While not explicitly considered in our construction, there are several ways to enforce the revocation of specific credentials. For example, leveraging the expressiveness of circuits, we can include a non-membership check (e.g., using Merkle trees or RSA accumulators) and add a conjunctive clause to prevent all revoked credentials from satisfying the predicate. Another method to enforce revocation is to require holders to produce separate non-revocation proofs when presenting their credentials, e.g., using the signature-based revocation mechanism of [11].

8 CONCLUSIONS

We presented RETRACT, a novel and fully expressive anonymous credential scheme allowing credential holders to prove knowledge of satisfying credentials to arbitrary predicates while ensuring that only the designated verifier believes them. Our construction demonstrated how to combine state-of-the-art commit-carrying zkSNARK constructions with the widely used BBS+ signature scheme.

REFERENCES

[1] Shashank Agrawal, Chaya Ganesh, and Payman Mohassel. 2018. Non-interactive zero-knowledge proofs for composite statements. In *Annual International Cryptology Conference*. Springer, 643–673.

[2] akosba. 2022. akosba/xjsnark: A high-level framework for developing efficient zk-SNARK circuits. https://github.com/akosba/xjsnark

[3] Anon. 2022. Prototype code. https://github.com/anonavailability/RETRACT

[4] Man Ho Au, Willy Susilo, and Yi Mu. 2006. Constant-size dynamic k-TAA. In *International conference on security and cryptography for networks*. Springer, 111–125.

[5] Michael Backes, Lucjan Hanzlik, and Jonas Schneider-Bensch. 2019. Membership privacy for fully dynamic group signatures. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2181–2198.

[6] Foteini Baldimtsi and Anna Lysyanskaya. 2013. Anonymous credentials light. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 1087–1098.

[7] Dan Boneh and Xavier Boyen. 2008. Short signatures without random oracles and the SDH assumption in bilinear groups. *Journal of cryptology* 21, 2 (2008), 149–177.

[8] Dan Boneh, Xavier Boyen, and Hovav Shacham. 2004. Short group signatures. In *Annual international cryptology conference*. Springer, 41–55.

[9] Ernie Brickell, Jan Camenisch, and Liqun Chen. 2004. Direct anonymous attestation. In *Proceedings of the 11th ACM conference on Computer and communications security*. 132–145.

[10] Jan Camenisch, Liqun Chen, Manu Drijvers, Anja Lehmann, David Novick, and Rainer Urian. 2017. One TPM to bind them all: Fixing TPM 2.0 for provably secure anonymous attestation. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 901–920.

[11] Jan Camenisch, Manu Drijvers, and Anja Lehmann. 2016. Anonymous attestation using the strong diffie hellman assumption revisited. In *International Conference on Trust and Trustworthy Computing*. Springer, 1–20.

[12] Jan Camenisch, Maria Dubovitskaya, Kristiyan Haralambiev, and Markulf Kohlweiss. 2015. Composable and modular anonymous credentials: definitions and practical constructions. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 262–288.

[13] Jan Camenisch and Thomas Groß. 2008. Efficient attributes for anonymous credentials. In *Proceedings of the 15th ACM conference on Computer and communications security*. 345–356.

[14] Jan Camenisch and Anna Lysyanskaya. 2004. Signature schemes and anonymous credentials from bilinear maps. In *Annual international cryptology conference*. Springer, 56–72.

[15] Jan Camenisch and Markus Stadler. 1997. Proof systems for general statements about discrete logarithms. *Technical Report/ETH Zurich, Department of Computer Science* 260 (1997).

[16] Jan Camenisch and Els Van Herreweghen. 2002. Design and implementation of the idemix anonymous credential system. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*. 21–30.

[17] Matteo Campanelli, Dario Fiore, and Anaïs Querol. 2019. LegoSNARK: modular design and composition of succinct zero-knowledge proofs. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2075–2092.

[18] Melissa Chase, Esha Ghosh, Srinath Setty, and Daniel Buchner. 2020. Zero-knowledge credentials with deferred revocation checks. Retrieved Sep 27, 2022 from https://github.com/decentralized-identity/snark-credentials/blob/master/whitepaper.pdf/

[19] David Chaum. 1985. Security without Identification: Transaction Systems to Make Big Brother Obsolete. *Commun. ACM* 28, 10 (1985).

[20] David Chaum. 1985. Security without identification: Transaction systems to make big brother obsolete. *Commun. ACM* 28, 10 (1985), 1030–1044.

[21] Liqun Chen and Rainer Urian. 2015. DAA-A: Direct anonymous attestation with attributes. In *International Conference on Trust and Trustworthy Computing*. Springer, 228–245.

[22] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. 1994. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Annual International Cryptology Conference*. Springer, 174–187.

[23] Ronald Cramer and Victor Shoup. 2002. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 45–64.

[24] Xudong Deng, Chengliang Tian, Fei Chen, and Hequn Xian. 2021. Designated-Verifier Anonymous Credential for Identity Management in Decentralized Systems. *Mobile Information Systems* 2021 (2021).

[25] Dominic Deuber, Matteo Maffei, Giulio Malavolta, Max Rabkin, Dominique Schröder, and Mark Simkin. 2018. Functional Credentials. *Proc. Priv. Enhancing Technol.* 2018, 2 (2018), 64–84.

[26] DIF. 2022. DIF Presentation Exchange. Retrieved Sep 27, 2022 from https://identity.foundation/presentation-exchange/

[27] dock. 2022. Dock: Verifiable Credentials Company. Retrieved November 13, 2022 from https://www.dock.io/

[28] Amos Fiat and Adi Shamir. 1986. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*. Springer, 186–194.

[29] Lorenzo Grassi et al. 2021. Poseidon: A new hash function for zero-knowledge proof systems. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*.

[30] Jens Groth. 2016. On the size of pairing-based non-interactive arguments. In *EUROCRYPT*. Springer, 305–326.

[31] Jinguang Han, Liqun Chen, Steve Schneider, Helen Treharne, and Stephan Wesemeyer. 2018. Anonymous single-sign-on for n designated services with traceability. In *European Symposium on Research in Computer Security*. Springer, 470–490.

[32] Lucjan Hanzlik and Daniel Slamanig. 2021. With a little help from my friends: constructing practical anonymous credentials. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2004–2023.

[33] iden3. 2022. iden3. Retrieved November 13, 2022 from https://github.com/iden3

[34] iden3. 2022. iden3/circom: zkSnark circuit compiler. Retrieved November 13, 2022 from https://github.com/iden3/circom

[35] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. 1996. Designated verifier proofs and their applications. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 143–154.

[36] Ahmed Kosba. 2021. Java zkSNARK library. https://github.com/akosba/jsnark

[37] Ahmed Kosba et al. 2018. xJsnark: A framework for efficient verifiable computation. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 944–961.

[38] SCIPR Lab. 2020. C++ zkSNARK library. https://github.com/scipr-lab/libsnark

[39] matteocam. 2022. libsnark: a C++ library for zkSNARK proofs. Retrieved November 13, 2022 from https://github.com/matteocam/libsnark-lego

[40] Christian Paquin and Greg Zaverucha. 2011. U-prove cryptographic specification v1. 1. *Technical Report, Microsoft Corporation* (2011).

[41] Torben Pryds Pedersen. 1991. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*. Springer, 129–140.

[42] Michael Rosenberg, Jacob White, Christina Garman, and Ian Miers. 2022. zk-creds: Flexible Anonymous Credentials from zkSNARKs and Existing Identity Infrastructure. *Cryptology ePrint Archive* (2022).

[43] Shahrokh Saeednia, Steve Kremer, and Olivier Markowitch. 2003. An efficient strong designated verifier signature scheme. In *International conference on information security and cryptology*. Springer, 40–54.

[44] Claus-Peter Schnorr. 1991. Efficient signature generation by smart cards. *Journal of cryptology* 4, 3 (1991), 161–174.

[45] Manu Sporny, Dave Longley, and David Chadwick. 2022. Verifiable Credentials Data Model 1.1. https://www.w3.org/TR/vc-data-model/.

[46] Syh-Yuan Tan and Thomas Groß. 2020. Monipoly—an expressive q-SDH-based anonymous attribute-based credential system. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 498–526.

[47] TCG. 2022. TPM 2.0 Library - Trusted Computing Group. Retrieved February 24, 2022 from trustedcomputinggroup.org/resource/tpm-library-specification/

[48] trinsic. 2022. Trinsic - A full-stack self-sovereign identity (SSI) platform. Retrieved November 13, 2022 from https://trinsic.id/

[49] European Union. 2015. *Council regulation (EU) no 1502/2015*. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=OJ:JOL_2015_235_R_0002

[50] W3C. 2022. Verifiable Credentials Data Model v1.1. Retrieved Sep 27, 2022 from https://www.w3.org/TR/vc-data-model/

[51] W3C Recommendation. 2021. Decentralized Identifiers (DIDs) v1.0 Core architecture, data model, and representations. https://www.w3.org/TR/did-core/.

[52] Howard Wu et al. 2018. {DIZK}: A distributed zero knowledge proof system. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 675–692.

A SCHNORR PROOF OF DISCRETE LOG

Given a protocol description in the notation in Section 3.3, a common method of compiling the actual protocol is following the idea behind the Schnorr *proof of knowledge of a discrete logarithm* protocol [44], which is a traditional three-move zero-knowledge Sigma protocol, i.e., a commit-challenge-response protocol. The idea is relatively simple. For proving knowledge of the value a in $y = g^a$, the prover generates randomness r and sends $t \leftarrow g^r$ to the verifier. Then, the verifier generates a random challenge c and sends it to the prover. The prover now computes the challenge response $s \leftarrow r+ca$, and sends s to the verifier. The verifier is convinced that the prover knows the discrete log of y only if $g^s = ty^c$. Furthermore, to make it a NIZK, i.e., collapse the three moves into one single move, we can use the Fiat-Shamir heuristic [28] in the random oracle model by replacing the verifier's random challenge with that of a value from a hash function H (modeled as a random oracle) on the prover's first message t and the input. Thus, in one round, the prover computes the challenge $c \leftarrow \mathsf{H}(g\|y\|t)$ and response $s \leftarrow r - cm$, and then sends $\pi = (c, s)$ to the verifier, who computes $t' \leftarrow g^s y^c$ and $c' \leftarrow \mathsf{H}(g\|y\|t')$, and accepts the proof if the challenges match: $c = c'$.

Note that we can generalize the Schnorr method to prove knowledge of the solutions (discrete logarithms) to several terms, each containing several exponents, where, for each term, $y = g^a h^a$, the prover transmits one group element and one response value for each exponent. The general idea for proving the AND (i.e., conjunction) of multiple statements is to execute them in parallel and use the same challenge. It gets more complicated for OR proofs, i.e., the disjunction of statements. In our construction, we use the idea described in [15] for composing a disjunction of statements to create designated verifier proofs.

B ON THE BREAK OF THE "STRONG" DESIGNATED VERIFIER

Note that [35] also proposed the *strong designated verifier* to essentially prevent Dave, an observer of the protocol interaction between a holder (Alice) and the designated verifier (Bob), from being convinced about the statement being proven by Alice (without Bob disclosing its secret key). They argue that we can promote a protocol to become a strong designated verifier by having Alice *probabilistically* encrypt the transcript using Bob's public key since then Bob cannot convince Dave about the decrypted message (due to the probabilistic encryption) since Bob can produce indistinguishable transcripts. Later, [43] proposed a more efficient method of achieving the same strongness property without requiring the signature to be encrypted by instead requiring Bob's secret key in the verification. However, we note here that Bob might succeed in convincing Dave if we assume that Dave initially observed the specific transcript being transmitted from Alice to Bob with the help of verifiable computation (e.g., use of zkSNARKs as described in Section 3.5). For example, in the first case, let c be the probabilistically encrypted transcript transferred from Alice to Bob (and observed by Dave). To convince Dave that c decrypts to π using Bob's secret key x_{Bob} (without having to disclose the key), Dave can send a decryption cipher as an arithmetic circuit $\mathsf{C}_{\mathsf{Dec}}$ to Bob, which takes an encrypted message as a public input and a decryption key as the secret witness. Then Bob can generate a zkSNARK proof that $\mathsf{C}_{\mathsf{Dec}}(c, x_{Bob}) = \pi$ and send it to Dave, who gets convinced that Alice produced the specific transcript. However, this "attack" is only possible if Dave initially observed the transmission to obtain the trusted reference value c. Therefore, in our current system, we assume that any third party, such as Dave, who is interested in receiving a conviction from dishonest designated verifiers was *not* actively observing the protocol interaction between the holder and the designated verifier.