IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Integrating Threshold Opening with Threshold Issuance of Anonymous Credentials over Blockchains for a Multi-certifier Communication Model

ADEEBA NAAZ [1], T. V. PAVAN KUMAR B [2], MARIA FRANCIS [3], AND KOTARO KATAOKA[4]
[1]Department of Computer Science & Engineering, Indian Institute of Technology Hyderabad, India (E-mail: cs19mtech11034@iith.ac.in)
[2]Department of Computer Science & Engineering, Indian Institute of Technology Hyderabad, India (E-mail: cs19mtech11020@iith.ac.in)
[3]Department of Computer Science & Engineering, Indian Institute of Technology Hyderabad, India (E-mail: kotaro@cse.iith.ac.in)
[4]Department of Computer Science & Engineering, Indian Institute of Technology Hyderabad, India (E-mail: mariaf@cse.iith.ac.in)

ABSTRACT Authentication while maintaining anonymity when availing a service over the internet is a significant privacy challenge. Anonymous credentials (AC) address this by providing the user with a credential issued by a trusted entity that convinces the service provider ($\mathcal{SP}$) that the user is authenticated but reveals no other information. The existing AC schemes assume a single trusted authority (certifier) that validates all the user attributes. In practice, however, a user may require different attributes to be attested by different certifiers. This means that the user has to get multiple credentials, increasing the burden on the $\mathcal{SP}$ who has to verify each one of them. Moreover, complete anonymity can be misused. We propose a ***d**ecentralized **t**hreshold **r**evocable **a**nonymous **c**redential (DTRAC)* scheme over blockchains that supports – a) attestation of attributes by multiple certifiers, and b) anonymity revocation through a set of distributed openers, by integrating threshold opening to the state-of-the-art threshold anonymous credential issuance scheme, Coconut [34]. DTRAC generates a single credential on attributes that are attested by multiple certifiers, freeing the SP from the hassle of verifying multiple credentials. We analyze the security of DTRAC formally in the universal composability (UC) framework. We also implement a prototype on Ethereum using smart contracts and give a detailed analysis of its performance. We compare the verification time for credentials with attributes attested by multiple certifiers in both DTRAC and Coconut and see that in terms of execution time and gas consumption, DTRAC performs significantly better than Coconut. It also scales better, with the performance gain of DTRAC over Coconut increasing linearly with the number of certifiers.

INDEX TERMS Zero-knowledge proofs, threshold issuance of credentials, threshold opening of credentials, multi-certifier model.

I. INTRODUCTION

With companies collecting massive amounts of personal data and internet applications becoming increasingly pervasive, privacy has become a major concern. The public, wary of sharing data, wants applications and services to strictly adhere to the data minimization principle, i.e., collect only relevant and limited personal information. The General Data Protection Regulation (GDPR) [37] establishes a legal framework for protecting personal data across the European Union and its compliance has increased the potential liability of managing users' personal data. At the same time, allowing for anonymity with no authentication is infeasible for many applications, especially those that provide services based on different categories of user attributes. Also, complete anonymity may encourage misuse of services, illegal activities and abusive behavior.

Anonymous credentials (AC) are privacy-preserving cryptographic authentication mechanisms that aim to give us the best of both worlds – authentication as well as anonymity. These mechanisms authenticate a user while allowing her to keep her attributes secret and issue her with a credential that can be presented several times while remaining *unlinkable* (or untraceable). Many AC schemes ([7], [10], [12], [15], [25]) have a centralized credential issuer who verifies the attributes of the user and issues the user with a credential. But the issuer can be compromised or corrupt and is a single point of failure. Garman *et al.* [20] circumvents this limitation by introducing decentralized anonymous credentials (DAC) where the issuance of credentials happens over a distributed ledger. But presenting a credential in their scheme is expensive due to a double discrete-logarithm proof. CanDID [27], another credential issuance scheme, allows for a decentralized committee to issue credentials and is legacy compatible. But it uses secure multiparty computation (MPC) techniques that are computationally intensive. Coconut [34], an efficient, short and constant size credential scheme, over blockchains, allows for threshold issuance in a decentralized manner. But the security of the scheme was analyzed informally. Alfredo *et al.* [31] proved the security of Coconut formally but makes a few modifications to the scheme for that. We use this modified version for credential issuance to ensure unforgeability. Versatile ABS [3] is an attribute-based signature scheme that supports threshold opening to revoke the anonymity of the user. But in this scheme every service request needs to contain additional information to trace the user, making it inefficient. However, none of the above support validation of attributes by multiple certifiers in a single credential. Our proposed scheme aims to resolve the following two gaps in existing AC schemes.

1) The existing AC schemes assume that a single trusted authority (certifier) – which itself can be a single entity or a distributed set of entities – certifies/attests all the user attributes. If a service requires different sets of user attributes to be verified by different authorities then the user has to get multiple ACs, one from each authority. The service provider ($\mathcal{SP}$) then must verify all these credentials before providing the service, which increases the burden on the $\mathcal{SP}$. This is the case, for instance, with versatile ABS and Coconut where multiple credentials have to be verified if multiple certifiers attest to different attributes.
2) To bring accountability to anonymity, AC schemes either look at credential revocation [8], [11], [36], [38] or anonymity revocation/opening of a credential [3], [10], [11]. However, all the schemes proposed for the latter either rely on a centralized trusted entity and do not consider the possibility of it being corrupt, i.e. they do not support threshold opening, or they need more information in the credential to trace the user.

We use the following use case to explain the reasoning behind having multiple certifiers, each validating a different subset of user attributes. Suppose Alice wants to apply for a personal loan from a finance company, Bob. Bob can provide a loan if the person meets certain eligibility criteria, say, criteria related to age and income. Alice would not want to share her personal information, her actual income details, or where she works, with Bob. Any identity provider (IdP) can act as a certifier and certify the identity attributes. However, this agency cannot be expected to certify the income attributes for which we need an income certifier. For example, after verifying the supporting documents, an employer can provide Alice with an income certificate that certifies her income details. Alice now has to obtain a credential based on a set of attributes such as DoB from the identity attributes and income from the income certificate. Thus, there are two certificates issued by two different certifiers that Alice needs to present to Bob, and he has to verify both. Also, Bob's task increases with the number of certificates.

In order to address this problem, our scheme proposes a specially constructed digital certificate, referred to as a *verifiable certificate* (Vcert), to be issued by each certifier that attests to specific user attributes. These Vcerts hide the attributes using cryptographic primitives, but they are still verifiable. A secret master key is associated with all the Vcerts of one particular user, which links all the Vcerts of that user. In the loan application use case, the IdP provides an ID Vcert on the identity attributes and an income certifier provides an income Vcert on the income-related attributes. In our scheme, a set of validators distributed over a blockchain verifies the attributes of these multiple Vcerts during a credential request. The zero-knowledge proofs of knowledge (ZKPoKs) of attributes are provided to the validators to convince them that the attributes were certified by the appropriate certifiers, but they reveal no other information. Each validator issues a partial credential on those attributes. A threshold number (not necessarily all) of these partial credentials are sufficient to form an anonymous credential and it can be verified using the aggregated public key of the validators. Our scheme, Decentralized Threshold Revocable Anonymous Credentials (DTRAC), enables Alice to obtain a single credential that she may use to prove to Bob that she meets all the eligibility criteria without revealing any private information. She is also free from the hassle of presenting numerous credentials. Note that we can have more than two certifiers, each attesting to their respective user attributes. We refer to this as a multi-certifier communication model. DTRAC extends the Coconut threshold issuance scheme [31] to support this multi-certifier communication model and provides a single anonymous credential for attributes attested by multiple certifiers. We analyzed the efficiency of DTRAC and our evaluations show that DTRAC performs significantly better than running multiple instances of Coconut. Also, this performance gain that DTRAC has over the Coconut scheme increases linearly as the number of certifiers increases making DTRAC more scalable than Coconut.

The revocation of anonymity in DTRAC is broadly based on the threshold opening scheme implemented for dynamic group signatures [9]. However, the threshold opening scheme in [9] works only for a single attribute, whereas credentials typically contain more. Hence, we extend it to support multiple attributes and integrate it into our threshold issuance scheme, allowing a designated set of *distributed openers* to open an anonymous credential if at least a threshold number of openers agree. We also provide a formal security analysis of DTRAC (Sec. VI).

Our Contributions.

1) DTRAC supports a multi-certifier model and it issues a single anonymous credential to a user on a set of attributes where multiple certifiers certify/attest specific subsets of user attributes.

2) DTRAC enables optional revocation of anonymity that does not rely on a centralized opener by integrating a threshold opening scheme, previously designed for group signatures, to the threshold credential issuance scheme. The existing threshold opening scheme was modified in this work to allow for multiple attributes.
3) We provide a formal security analysis of DTRAC in the universal composability (UC) framework.
4) We provide a working PoC on the Ethereum blockchain for the loan application use case and give a detailed evaluation. We demonstrate that our scheme performs significantly better than the state-of-the-art threshold issuance scheme, Coconut in terms of execution time and gas consumption, for credentials with attributes attested by multiple certifiers.

Paper Organization. Sec. II introduces preliminaries of techniques used in our scheme. We give an overview of our approach and the building blocks that we construct as part of our scheme in Sec. III. We describe the system architecture in Sec. IV and the details of the actual construction in Sec. V. The formal security analysis is done in Sec. VI. The implementation details and the evaluation of the prototype is provided in Sec. VII and VIII, respectively. Sec. IX describes the related work and Sec. X contains concluding remarks.

II. PRELIMINARIES

We use $\mathbb{Z}$ to denote the set of integers, p for a prime number, $\mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z}$ for the ring of integers modulo p and 1^k for the security parameter. For an integer $n \geq 1$, $[n]$ denotes the set $\{1, \dots, n\}$.

We extensively use bilinear pairings in our work and they rely on elliptic curve groups and the hardness of the *discrete logarithm problem (DLP)* over elliptic curve groups. We give a brief introduction to bilinear groups in Appendix A-A. We use type III bilinear pairing groups [18] to build our scheme. We represent them as $(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T, e)$, where $\mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T$ are cyclic groups of prime order p and $e : \mathbb{G} \times \tilde{\mathbb{G}} \rightarrow \mathbb{G}_T$ is an efficiently computable, non-degenerate, bilinear mapping. Let $\mathbb{G}$ be an elliptic curve group of prime order p, DLP over elliptic curves is defined as follows: Let $P \in \mathbb{G}$ be a point on the curve and Q be a point in the subgroup generated by P. DLP is the problem of finding $x \in \mathbb{Z}$ such that $Q = xP$. It is assumed to be computationally hard. Note that we use the additive notation for elliptic curve group operations.

A. ZERO-KNOWLEDGE PROOFS (ZKPS)

Zero-knowledge proof is a two-party protocol that enables a prover to convince a verifier about the validity of a statement without revealing any further information. For a comprehensive introduction to ZKPs and ZKPoKs, one can refer to [21]. Here we outline the basic idea of Schnorr's interactive ZKP for discrete logarithms [32] used in our scheme. Let G be a generator of the elliptic curve group $\mathbb{G}$ of prime order p where DLP is assumed to be hard. Let Q and G be the two publicly available elliptic curve points such that $Q = xG$ and x be the user's secret. The protocol starts with the prover (user) sending an elliptic curve point $A = rG$, where r is a randomly chosen element in $\mathbb{Z}_p$, to the verifier. The prover receives a random challenge $c \in \mathbb{Z}_p$ from the verifier. The prover sends $z = (r + cx) \bmod p$ to the verifier and the verifier is convinced of the user's knowledge of the secret x if $zG = A + cQ$. Schnorr's protocol can be transformed to the non-interactive ZKP setting using the Fiat-Shamir heuristic [16].

For ZKPs we use notations from [13]. For example,

$$\text{ZKPoK}\{(x_1, \dots, x_n) : \mathcal{R}(x_1, \dots, x_n, y_1, \dots y_m) = true\},$$

represents the prover's knowledge of the secret values $(x_1, \dots, x_n)$ in zero-knowledge that satisfy a boolean relation $\mathcal{R}$ over the public inputs $\{y_1, \dots y_m\}$. The Schnorr's protocol described above can be represented as $\pi = \text{ZKPoK}\{(x) : Q = xG\}$.

B. GENERALIZED PEDERSEN COMMITMENTS

A commitment scheme is a cryptographic algorithm that enables a prover to commit to a message without revealing it and without being able to modify it later. We use the Pedersen commitment scheme [28] in this work and in its generalized version, a prover can commit to a set of messages $m_1, \dots, m_n \in \mathbb{Z}_p$ using a random value r, $C = Commit(m_1, \dots, m_n; r)$ which it sends to the verifier. C reveals no information about the messages and it is computationally hard to find another set of messages $(m'_1, \dots, m'_n) \neq (m_1, \dots, m_n)$ such that $Commit(m_1, \dots, m_n; r) = Commit(m'_1, \dots, m'_n; r)$. The construction relies on the hardness of the DLP problem.

- *Setup.* Pick an elliptic curve group $\mathbb{G}$ of prime order p with generators $G, H_1, \dots, H_n$ where DLP is assumed to be hard.
- *Commit*$(m_1, \dots, m_n; r) = rG + m_1H_1 + \dots + m_nH_n$.

There are efficient ZKPoK protocols that prove the knowledge of the opening of a commitment that satisfy numerous boolean relations over the commitment's opening [2].

C. PUBLICLY VERIFIABLE SECRET SHARING

A publicly verifiable secret sharing (PVSS) scheme [35] consists of a dealer and n participants. Each participant is associated with a public-private key pair. The dealer splits the secret into n shares and encrypts them with the public keys of the participants and distributes it to the participants. Anyone with access to the public keys of the participants can verify the correctness of the distributed shares, but they cannot view them. PVSS prevents malicious dealers from sending incorrect shares to the participants.

D. THRESHOLD PS SIGNATURES

PS signatures [29] are short signatures based on type III bilinear pairings. They are *randomizable* in the following way: given a signature σ for a message m, $a\sigma$ for a scalar $a \neq 0$, gives us a fresh signature for m. They are constant in size, unlike say, CL signatures [12] where the signature size is linear in the number of messages to be signed. A $(t, n)-$ threshold credential issuance scheme requires at least t signers out of n to issue an anonymous credential. PS signatures do not support threshold issuance. The Coconut implementation [34] addresses this limitation and introduces threshold PS signatures. But the authors analyze the security informally. Alfredo *et al.* [31] provides a formal security proof for a modified Coconut scheme and DTRAC uses this version, that provides unforgeability, for credential issuance. For credential verification, however, we use the older version of Coconut but this does not affect the correctness or security of our scheme.

E. THRESHOLD OPENING SCHEME

A $(t, n)-$ threshold opening scheme reveals/opens the user's identity associated with an anonymous credential if at least t openers out of the n work together. In [9], a PVSS scheme (see Sec. II-C) is used to integrate threshold opening with threshold issuance for group signatures. The users send the opening information required for anonymity revocation during credential issuance, encrypted using the public keys of the openers. A zero-knowledge proof is also sent by the user to convince the issuers of the correctness of the encrypted opening information. The issuance happens through a public ledger so that the openers can collect the opening information. Later, while revoking the anonymity, a threshold number of openers work collaboratively to identify the user. Note that the threshold opening scheme for group signatures [9] supports only one attribute and in DTRAC we extend it to multiple attributes since credentials typically contain more than one attribute. Typically, the openers are different entities from the issuers.

F. TAMPER-RESISTANT PUBLIC LEDGER

Blockchains are based on a synchronized distributed ledger technology (DLT), which acts as a decentralized database, maintaining information replication and sharing among several nodes distributed over remote locations, with a consistent view across all the nodes. A blockchain thus offers transparent, verifiable and tamper-proof data storage. Our system uses the blockchain as a publicly distributed append-only ledger. It ensures all the honest validators and openers receive the same data and this enables threshold issuance and threshold opening of a credential. Our PoC is implemented over the Ethereum blockchain as its smart contracts allow for the necessary mathematical operations and enable interaction among distributed stakeholders in a tamper-resistant and verifiable manner.

G. UNIVERSAL COMPOSABILITY FRAMEWORK

The UC framework [14], an ideal-world/real-world paradigm, guarantees the security of protocols that are arbitrarily composed of other instances of the same or different protocols. This is useful in showing the security of our scheme. To show the security of a protocol Π in the real world in the UC framework, we have to first propose an ideal functionality $\mathcal{F}$ which captures the desired/ideal properties of the system. All the parties involved send their inputs to $\mathcal{F}$, which locally computes the outputs and hands them back.

We say that the protocol Π in the real world UC-securely realizes an ideal functionality $\mathcal{F}$ if the environment $\mathcal{Z}$ cannot distinguish between an adversary $\mathcal{A}$ interacting with the protocol Π from a simulator $\mathcal{S}$, that we construct to simulate the ideal adversary, interacting with the ideal functionality $\mathcal{F}$. We first design a hybrid model where the real-world protocol Π has access to certain functionalities in the ideal world and show that it UC-securely realizes $\mathcal{F}$. This is enough to show the security of the protocol since the UC paradigm states that if we replace each invocation of the ideal functionalities that Π has access to with its corresponding real-world protocols then Π securely realizes $\mathcal{F}$ in the real world.

• • •

III. DTRAC MODEL AND BUILDING BLOCKS

A. STAKEHOLDERS

Fig. 1 describes the stakeholders and their interactions with each other. The *user* is an entity who wishes to remain anonymous while availing of service from a service provider. The service provider needs to verify if the user attributes have been attested by the certifiers and if the attributes satisfy certain conditions before providing a service. The user relies on multiple certifiers to certify several subsets of her attributes. She consents to have her anonymity revoked in certain circumstances by a designated set of openers.

A *certifier* $(\mathcal{CP})$ is an organization that certifies specific user attributes. The $\mathcal{CP}$ is trusted to certify the attributes correctly. In our scheme, we can have multiple $\mathcal{CP}$s, each attesting to specific user attributes. After certifying the user attributes, each $\mathcal{CP}$ issues a specially constructed and privacy-preserving digital certificate called a verifiable certificate (Vcert) (see Sec. III-C1) on those attributes. The actual certification procedure depends on the use case and is outside the scope of this paper.

The *distributed validators* are an independent set of authorities where each validator issues the user with a partial credential (Pcred) after verifying the Vcerts. At least a threshold number of validators have to issue Pcreds to form an anonymous credential of the user. They do not view the actual attributes but only the ZKPoK that the attributes were verified by the respective certifier.

The *distributed openers* can open an anonymous credential if deemed necessary. They are typically an independent set of authorities where each opener maintains a private registry to store their opening share corresponding to each user. At least a threshold number of openers have to work collaboratively to open an anonymous credential, i.e., perform an anonymity revocation.

A *service provider* $(\mathcal{SP})$ provides a service to an authenticated user anonymously. The $\mathcal{SP}$ verifies that the attributes are validated by the (distributed) validators but in the process learns nothing about the user attributes unless the user decides to disclose them.

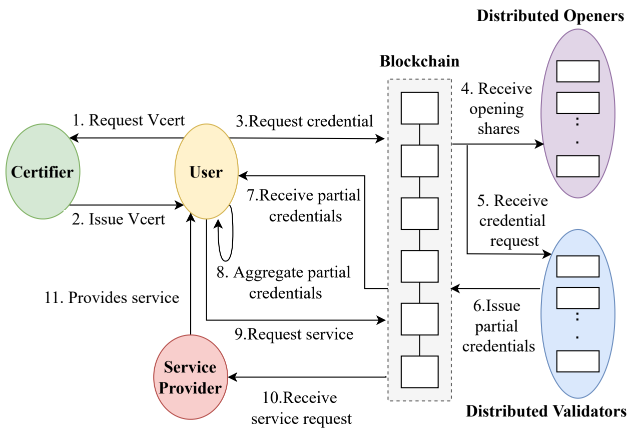
Distributed Openers
Blockchain
1. Request Vcert
3.Request credential
4. Receive opening shares
Certifier
User
7.Receive partial credentials
5. Receive credential request
2. Issue Vcert
8. Aggregate partial credentials
11. Provides service
6.Issue partial credentials
9.Request service
Service Provider
10.Receive service request
Distributed Validators

FIGURE 1: Stakeholders of DTRAC

B. THREAT MODEL

Our threat model assumes that the certifiers will do their job of certifying the attributes correctly. We also assume that they will not collude with each other. If they do collude, since the Vcerts issued by multiple certifiers are linkable, the information of each Vcert, and thus all the attributes of the user, will be revealed. But in our scheme even if the certifiers collude with the $\mathcal{SP}$, the credential will remain anonymous. In fact, the $\mathcal{SP}$ is not trusted and can collude with all the other entities and the credential will still remain unlinkable. The threat model assumes users to be potentially malicious. Let n_V and t_V be the total number of validators and the corresponding threshold value, respectively, and n_O and t_O be the total number of openers and the corresponding threshold value, respectively. Integrity and availability are guaranteed under the corruption of a subset of validators that are less than the $min(t_V-1, n_V-t_V)$ and a subset of openers that are less than the $min(t_O-1, n_O-t_O)$. That is, forging of a credential is prevented if there are at most t_V-1 corrupt validators. DTRAC also precludes the misuse of anonymity revocation if there are at most t_O-1 corrupt openers. We assume that the number of dishonest validators cannot exceed the value $min(t_V-1, n_V-t_V)$. If $(n_V-t_V) \geq (t_V-1)$ then the maximum number of dishonest validators is $\tilde{t}_V = t_V-1$. Otherwise, $\tilde{t}_V = (n_V-t_V) < (t_V-1)$. A similar assumption holds for openers while revoking the anonymity of a user. An opener who participates in the opening protocol is assumed to share the correct opening information with the other openers.

We assume that the blockchain infrastructure is secure and that adversaries do not control enough resources to disrupt the decentralized consensus of the network. The adversary can eavesdrop on all the blockchain transactions, and we assume there exists a secure channel for off-chain communication.

C. BUILDING BLOCKS

DTRAC comprises the following building blocks – verifiable certificates (Vcerts), a multi-certifier model, a threshold issuance scheme and a threshold opening scheme. They are built using pairings-based cryptographic primitives, a publicly verifiable secret sharing scheme (PVSS), threshold PS signatures, Pedersen commitments, and zero-knowledge proofs of knowledge (ZKPoKs), all of which we briefly describe in Sec. II. A PVSS scheme helps integrate threshold opening with the threshold issuance scheme. The opening information is encrypted by the user using the public keys of the openers and is shared with all the openers. The distributed validators verify the correctness of the opening information as part of threshold issuance using the PVSS scheme, which does not happen in the Coconut scheme where opening is not a feature.

As stated before, one of the novel features of DTRAC is that it allows multiple certifiers to attest to specific attributes, and we explain in detail the building blocks specially designed for it below.

1) Verifiable Certificates

A verifiable certificate (Vcert) is an enhanced version of an attestation certificate. The Vcert hides the user's attributes but still can be used to prove their authenticity. Using Vcerts, the validators can verify that a certifier has attested the user attributes, without actually viewing the attributes.

The user requests for a Vcert on a set of attributes using the Pedersen commitment to her attributes and a ZKPoK, to prove the correctness of the commitment. The binding property of Pedersen commitments ensures that the user cannot modify the attributes that she has committed to. A certifier verifies the user's attributes corresponding to the commitment and signs on it. The Vcert comprises a) the commitment to the user attributes and b) the digital signature of the certifier who verified these attributes. The construction is explained in Section V-A.

A user can have multiple Vcerts, each for a different set of attributes. An identity provider, for example, can be a certifier who verifies the user's identity attributes and provides the user with an identity Vcert. It is trusted to do that job properly. Except for the identity provider (IdP), every other certifier verifies the user's identity based on the identity Vcert provided by the user in zero-knowledge. All the Vcerts of a particular user are linked by a secret master key, generated once by the user. Note that the user has to use the same secret master key for all the certifiers if the Vcerts have to be consolidated into a single credential.

The Vcert issuance design supports all-or-nothing non-transferability – if the user shares any one of her Vcerts with another user, i.e. the user shares her Vcert with the master secret key and the attributes, then it effectively means she is sharing all her Vcerts. This should deter any user from sharing them. To enable this, a public repository is maintained by each certifier where it stores the encrypted user attributes, which can be decrypted using the user's secret master key.

2) Multi-certifier communication model

One of the novel features of DTRAC is that it allows multiple certifiers to verify disjoint subsets of user attributes. For instance, we can have an identity provider confirming the user's identity attributes and an employer attesting to, say, the user's income attributes. Each certifier provides a verifiable certificate (Vcert) to the user after verifying the corresponding attributes. The actual process by which the certifier verifies will depend on the application and is beyond the scope of this work.

The multi-certifier model enables a user to obtain Vcerts on different sets of attributes, each verified by the respective certifier in a privacy-preserving manner. The issuance of a Vcert uses a non-interactive ZKP protocol that hides the user's secret master key. The Vcerts are then presented to a set of distributed validators to obtain an anonymous credential.

The multi-certifier model is more efficient than running multiple instances of a single certifier threshold issuance scheme, such as the Coconut scheme, in parallel. This is because our model requires verification of only a single credential by the $\mathcal{SP}$ as opposed to multiple credentials in Coconut. For example, consider the loan application use case discussed in Sec. I. On running two instances of the Coconut scheme, the user gets two credentials – one based on her identity attributes and another based on her income attributes. The $\mathcal{SP}$ has to verify both credentials. In our proposed scheme, the $\mathcal{SP}$ needs to verify only a single credential and it contains all the required attributes. The two certifiers provide attestation on their respective sets of attributes in the form of an ID Vcert and an income Vcert and the user requests an anonymous credential from the distributed validators in a single interaction. Since DTRAC provides a single credential, the size of the credential is constant irrespective of the number of attributes. This is not the case if we have different credentials for each set of attributes. The performance benefit of having only one credential to verify is very significant as we will show in Sec. VIII-C, where we will compare the PoC we develop with the Coconut scheme. In fact, the performance gain increases linearly as the number of certifiers increases.

IV. SYSTEM ARCHITECTURE

Here, we give the system overview of our scheme. The working of DTRAC is divided into four phases as depicted in Fig. 2: *Registration, Issuance, Verification* and *Opening*. We describe each phase and its corresponding cryptographic methods below and in Sec. V we provide more details of all these methods.

A. REGISTRATION PHASE

The steps 1.1 and 1.2 in Fig. 2 correspond to the registration phase where a user obtains an attestation on her attribute from a certifier in the form of a Vcert. As depicted in Fig. 3, the user first computes the commitment C to her attribute m and her secret master key sk_u using the *GenCommitment* method. Also, to prove the correctness of the commitment, she generates a ZKPoK using the *GenZKPoK* method. The user sends the generated information to a certifier and requests for a Vcert which is an off-chain communication. The certifier verifies the ZKPoK using the *VerifyZKPoK* method. Once the verification is successful, the certifier signs, $sign$, on the commitment using the *SignCommitment* method, and sends the Vcert, $(C, sign)$, to the user.

The user can request multiple Vcerts on different sets of attributes from the respective certifiers. If there is no dependency between the Vcerts, the registration phase can be parallelized.

B. ISSUANCE PHASE

In the issuance phase (Steps 2.1 - 2.4 in Fig. 2), the user sends a credential request containing a commitment to a set of attributes, Vcerts and the encrypted opening shares to the validators and the openers through the blockchain. Each validator verifies the request and issues a partial credential to the user. The user needs to collect a certain number of partial credentials, the value of which is set by the application, so that they can be aggregated to form a single anonymous credential.

The issuance protocol follows a modified PS threshold issuance scheme, where t_V out of n_V partial credentials will be aggregated to form an anonymous credential. The credential request is generated using the *PrepareCredRequest* method and sent to the distributed validators. As depicted in Fig. 4, the credential request contains the following information: 1) Vcerts, 2) commitment to all the attributes (Eq. 1), 3) commitments to individual user attributes (Eq. 2), 4) encrypted opening shares of the attributes, and 5) the ZKPoKs proving that the attributes in the credential request are attested by the certifiers (Eq. 8). The encryption of the opening shares is performed using the public keys of the corresponding openers (Eq. 5) and the proofs of correctness of these ciphertexts in zero-knowledge are also provided (Eq. 7).

On receiving the request, each validator verifies the correctness of the attributes using the signature on the Vcert and the ZKPoK. Once the verification is successful, the validator issues a partial credential by blind-signing the attributes using the *BlindSign* method. The user uses the *Unblind* method to obtain the corresponding partial credential. Once the user has the threshold number of partial credentials, she aggregates them to form an anonymous credential using the *AggCred* method, and stores it in her local storage.

The distributed openers retrieve their respective opening shares from the credential request and store them in their private registry (Step 2.2 in Fig 2).

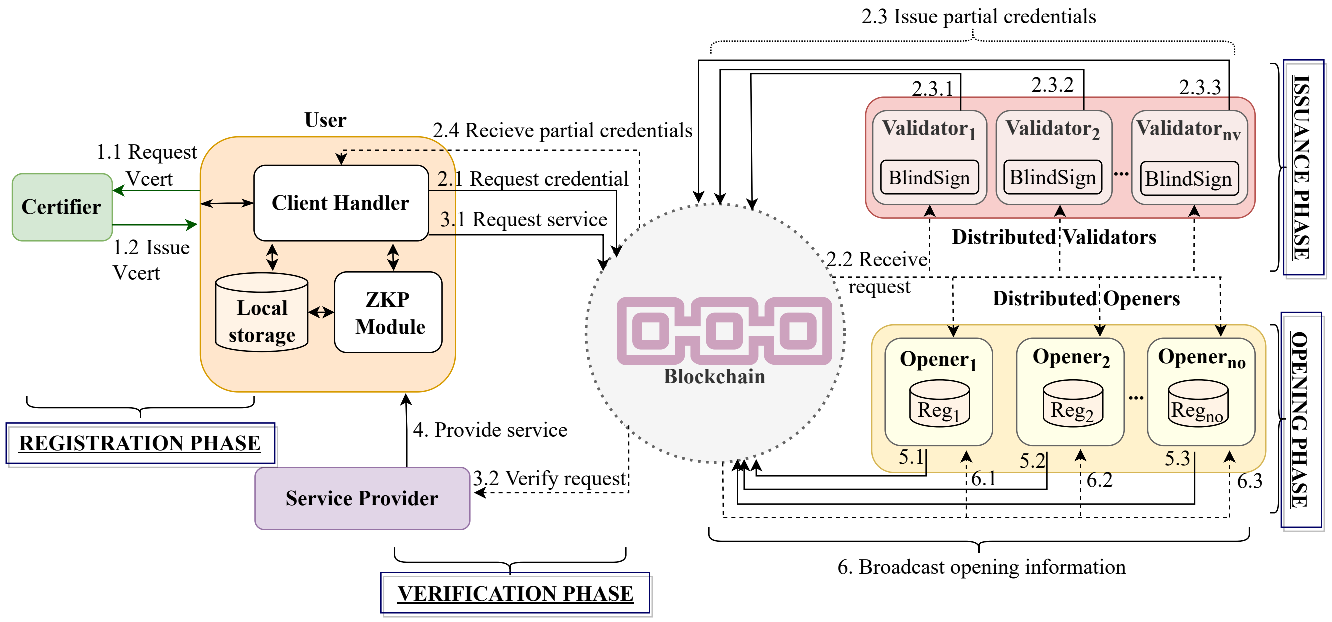

FIGURE 2: Overview of System Architecture

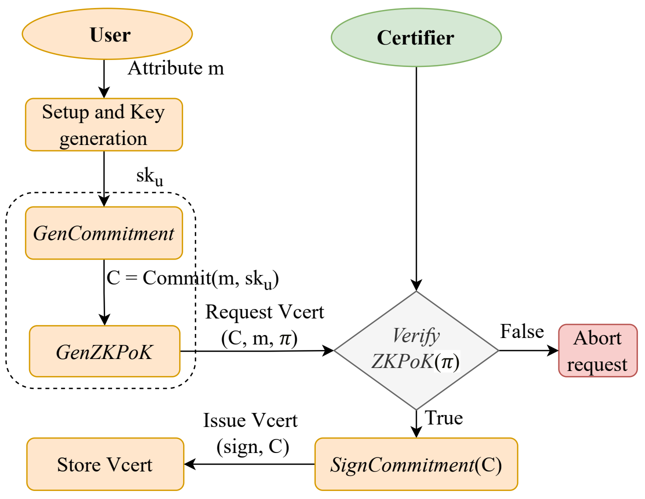

FIGURE 3: Registration phase

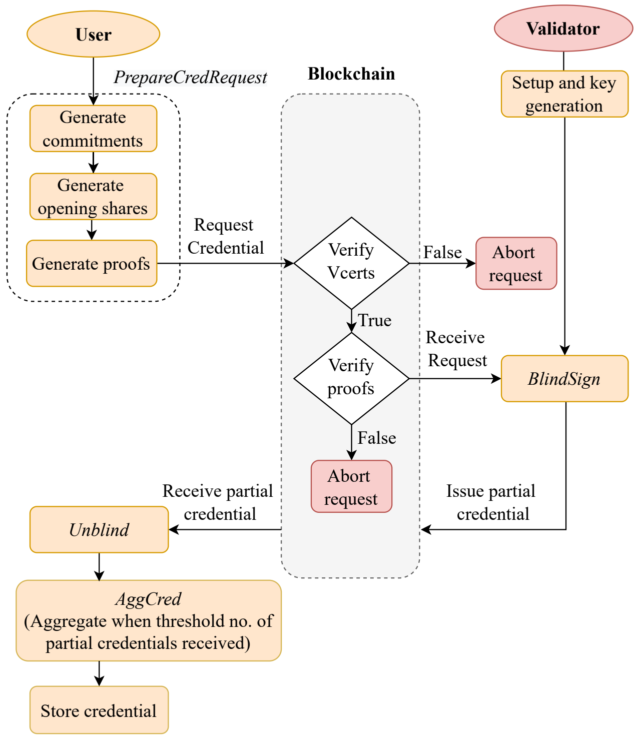

FIGURE 4: Issuance phase

C. VERIFICATION PHASE

The user requests a service from the service provider ($\mathcal{SP}$) through the blockchain by providing the aggregated credential (Steps 3.1, 3.2, 4 in Fig. 2). In the verification phase, the user has to prove to the $\mathcal{SP}$ that certain constraints over her attributes are satisfied. For example, in the loan application use case, she has to prove that her age is in a certain range and her income is above a certain value.

As depicted in Fig. 5, prior to requesting a service, the user randomizes her anonymous credential σ to obtain a fresh credential σ' and decides which attributes to disclose. She generates the ZKPoK of the undisclosed attributes (Eq. 13) using the *ProveCred* method. She then requests the service through the blockchain using σ', the ZKPoK and the disclosed attributes. The $\mathcal{SP}$ verifies the ZKPoK and the credential using the aggregated public key of the validators (Eq. 14). Upon successful verification, the $\mathcal{SP}$ grants the service.

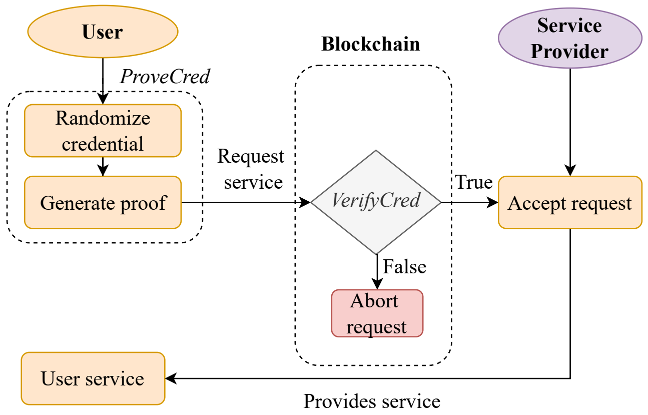

FIGURE 5: Verification phase

D. OPENING PHASE

The opening phase is executed by a designated set of distributed openers when it is deemed necessary to open the credential (Steps 5.1-6.3 in Fig. 2). Each opener broadcasts its opening information through the blockchain and upon receiving this information from at least a threshold number of openers, the openers execute the opening protocol to revoke the anonymity of the credential. It does this by linking the randomized credential used for the service request with the session in which the credential was originally issued.

As depicted in Fig. 6, each participating opener computes the opening information (Eq. 15) using the *PreOpening* method and sends it over the blockchain. They also retrieve the opening information from the other openers through the blockchain. After retrieving a threshold number, an opener calls the *OpenCred* method that outputs the issuance session identifier $reqid$ associated with the user credential (Eq. 16). Since the Vcerts are linkable, the identity Vcert corresponding to the $reqid$ can be used to identify the user.

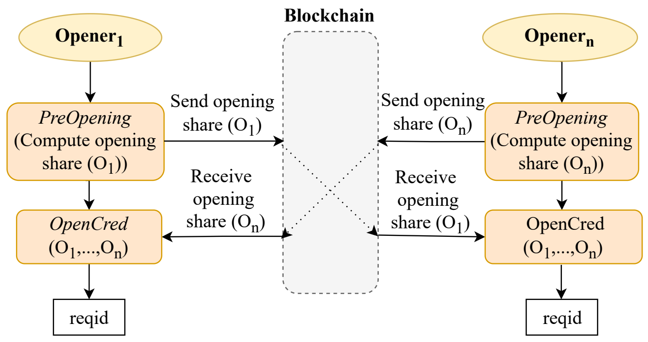
Opener$_1$
Blockchain
Opener$_n$
PreOpening (Compute opening share (O$_1$))
Send opening share (O$_1$)
Send opening share (O$_n$)
PreOpening (Compute opening share (O$_n$))
Receive opening share (O$_n$)
Receive opening share (O$_1$)
OpenCred (O$_1$,...,O$_n$)
OpenCred (O$_1$,...,O$_n$)
reqid
reqid

FIGURE 6: Opening phase

Note that the openers cannot block the partial credential being issued by the validators in the issuance phase, as the openers only collect the opening information in this phase. Due to the consensus property of blockchains, all the honest parties – the validators and the openers – receive the same information. Hence the same information cannot produce different verification results if the parties are honest.

Role of Blockchain. Blockchain, being a distributed and tamper-resistant public ledger, enables the openers to retrieve the appropriate opening shares in the issuance phase, as the roles of issuers and openers are separate. It also enables the openers to communicate during the opening protocol. An opener sends the opening information through the blockchain while opening a credential and this ensures that all the other openers receive the same information. The blockchain's consensus property makes it a stand-in for the public ledger.

E. PRIVACY PRESERVING LOANS – A USE CASE FOR DTRAC

Let Alice be a salaried employee who wants to avail a personal loan from a finance company, Bob. To be eligible for the loan, she has to satisfy certain criteria say, her age should be between 22 to 58 years and her monthly income should be greater than a certain value, 'X'. DTRAC enables her to obtain a single credential that proves she satisfies all the eligibility criteria without revealing any specific details such as her actual age or income and also frees her from the hassle of showing multiple documents. Moreover, Bob has to verify only a single credential to accept or reject the loan. Using DTRAC, Bob processes the loan without knowing the actual age and income of Alice.

Alice obtains an ID Vcert on her identity attributes such as name, age and address from one of the authorized certifiers. She also obtains an income Vcert which is an attestation of her income details from a certifier such as her employer on attributes such as income, designation and company. She also generates her secret master key, sk_u. To get a credential for the loan application, Alice has to generate a commitment on the subset of her attributes – name, age and income – and sk_u and also generate the corresponding ZKPoKs that indicate that she has indeed committed to the correct attributes. Hence, the actual attributes are hidden from the validators using commitments. She also shares the encrypted opening shares so that the credential can be opened later, if necessary. Alice sends the commitments of her attributes, the Vcerts, and the ZKPoKs as part of a credential request on the blockchain. The Vcerts and the ZKPoKs are verified on the blockchain. On successful verification, each validator blind signs the commitments to form a partial credential, *Pcred*. Upon receiving a threshold number of partial credentials, the user aggregates them to obtain a credential σ. To apply for the loan, Alice randomizes the credential as σ' and also generates the ZKPoKs that prove that her age is in the required range and her income is higher than the required value. Alice presents σ' and the ZKPoKs to Bob. Bob verifies σ' using the public key of the validators, and on successful verification, starts processing the loan.

V. DTRAC CONSTRUCTION

Here, we describe the phases and the corresponding methods of our scheme, DTRAC, in detail.

A. REGISTRATION PHASE

To request for a Vcert on q attributes $a_1, \ldots, a_q$ from a certifier, $\mathcal{CP}$, the following steps are executed.

Setup and Key Generation. The certifier, $\mathcal{CP}$, generates all the public parameters for the Pedersen commitment scheme and a public-private key pair $(pk_{\mathcal{CP}}, sk_{\mathcal{CP}}) \in \mathbb{G} \times \mathbb{Z}_p$, where $\mathbb{G}$ is an elliptic curve group of prime order p with G as its generator. The user generates a secret master key, chosen uniformly at random from $\mathbb{Z}_p$, and generated once per user. Let $pk_u = sk_u G$ be the public master key of the user.

Request Vcert. In order to request a Vcert, the user generates the commitment to its attributes and a ZKPoK of its correctness as follows.

1) *GenCommitment(*$a_1, \ldots, a_q, sk_u$*)* $\rightarrow$ *(C, r)*. The user selects a random $r \in \mathbb{Z}_p$ and computes the Pedersen commitment to her attributes and secret key as $C = Commit(sk_u, a_1, \ldots, a_q; r)$.
2) *GenZKPoK(*$C, sk_u, pk_u, a_1, \ldots, a_q, r$*)* $\rightarrow$ *(*π*)*. The user constructs a ZKPoK π that convinces the certifier that the corresponding user has knowledge of the opening of the commitment and the public key corresponding to the secret key, sk_u.

$$\pi = \text{ZKPoK}\{(sk_u, r) : C = \\ Commit(sk_u, a_1, \ldots, a_q; r) \ \wedge pk_u = sk_u G \ \}.$$

She requests for a Vcert from the $\mathcal{CP}$ using π, C, pk_u and $a_1, \ldots, a_q$.

Issue Vcert. On receiving the request, the $\mathcal{CP}$ issues the Vcert as follows.

1) *VerifyZKPoK(*C*,* $a_1, \ldots, a_q$*,* pk_u*,* π*,* pk_{id}*)* $\rightarrow$ *(*$true/false$*)*. The $\mathcal{CP}$ verifies the attributes provided to it and the ZKPoK π sent by the user. On successful verification, the $\mathcal{CP}$ stores the attributes encrypted using pk_u in a public repository and returns $true$, else $false$.
2) *SignCommitment(*C*,* $sk_{\mathcal{CP}}$*)* $\rightarrow$ *(*$sign$*)*. The $\mathcal{CP}$ signs C using $sk_{\mathcal{CP}}$ to obtain $sign$. The $\mathcal{CP}$ sends the $Vcert = (C, sign)$ to the user.
3) *VerifyVcerts(*$Vcert$*,* $pk_{\mathcal{CP}}$*)* $\rightarrow$ *(*$true/false$*)*. The user executes this method to verify the signature on the Vcerts. On successful verification it returns $true$, else $false$. Note that anyone with access to the certifier's public key, $pk_{\mathcal{CP}}$, can use this method to verify the signature on the Vcerts.

Note that the key generation step is done only once and can be used for all future communication. As stated before, our system supports a multi-certifier communication model where multiple certifiers issue separate Vcerts for multiple disjoint sets of user attributes (Sec. III-C2).

B. ISSUANCE PHASE

To construct the issuance phase, we modify Coconut's threshold PS signature scheme so as to incorporate two additional information: a) Vcerts to allow for multi-certifier communication, and b) encrypted publicly verifiable opening shares of user attributes and proofs of correctness of the shares to enable optional revocation of anonymity. To accommodate this we modify the *PrepareBlindSign* method of Coconut [34, Sec. III.D] which is itself modified from [31, Sec. 7]. We refer to the modified *PrepareBlindSign* method as *PrepareCredRequest* method.

Setup and Key Generation. The key generation of the validators can be done using a distributed key generation algorithm [23] but here, we rely on a single trusted party.

1) *Setup(*1^k*)* $\rightarrow$ (pp). Let $(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T, e)$ be a type III bilinear group. Let G, $\tilde{G}$ be the generators of $\mathbb{G}$ and $\tilde{\mathbb{G}}$, respectively. Assuming there are q user attributes, pick q group generators $H_1, \ldots, H_q \in \mathbb{G}$, randomly. The public parameter is given by $pp = (p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T, e, G, \tilde{G}, \{H_j\}_{j=1}^q)$. Let $\mathcal{RO} : \ \mathbb{G} \rightarrow \mathbb{G}$ be a random oracle that generates an element from the user's commitment to act as the common base for all the validators who validate the same credential request.
2) *ValidatorKeyGen(pp,* t_V*,* n_V*)* $\rightarrow (sk_v, pk_v)$. It picks $sk_v = (x, \{y_j\}_{j=1}^q) \in \mathbb{Z}_p^{q+1}$ as the aggregated secret key of the validators and sends the corresponding Shamir secret shares (see Sec. A-B) of sk_v, $sk_i = (x_i, \{y_{i,j}\}_{j=1}^q)$, $i \in [n_V]$ to the i-th validator. The aggregated public key is $pk_v = (\tilde{G}, \tilde{\alpha}, \{\beta_j, \tilde{\beta}_j\}_{j=1}^q) = (\tilde{G}, \ x\tilde{G}, \ \{y_j G, \ y_j \tilde{G}\}_{j=1}^q)$ and the corresponding share, $pk_i = (\tilde{G}, \tilde{\alpha}_i, \{\beta_{i,j}, \tilde{\beta}_{i,j}\}_{j=1}^q)$ is the public key of the i-th validator. .
3) *OpenerKeyGen(pp)* $\rightarrow$ (F_k, z_k). Each opener $k \in [n_O]$ generates a public-private ElGamal key pair $(F_k, z_k) \in \tilde{\mathbb{G}} \times \mathbb{Z}_p$ and initializes an empty registry Reg_k.

Credential Request. The user requests for a credential on attributes $(a_1, \ldots, a_q)$ specific to the service. For a simpler explanation, we assume that the credential request is made using a single Vcert.

- *PrepareCredRequest(*$a_1, \ldots, a_q$*,* sk_u*,* $Vcert$*)* $\rightarrow$ *(*$\{X_j\}_{j=1}^q$*,* C_a*,* $\{(U_k, \pi_k)\}_{i=1}^{n_O}$*, o,* $\{o_j\}_{j=1}^q$*,* π_s*)*. The $Vcert$ has the form $(C, sign)$, where C is the commitment on both the user attributes $a_1, \ldots, a_q$ and the master key sk_u and $sign$ is the signature of the certifier. The user computes a commitment C_a on the attributes, $a = (a_1, \ldots, a_q)$ and hashes C_a to a point H on $\mathbb{G}$,

$$C_a = oG + \sum_{j=1}^{q} a_j H_j \ \text{ and } \ H = \mathcal{RO}(C_a). \quad (1)$$

The user generates random values $o_1, \ldots, o_q \in \mathbb{Z}_p$ and computes the commitment to each individual attribute $a_j, j \in \{1, \ldots, q\}$ as

$$X_j = o_j G + a_j H. \quad (2)$$

As stated in Sec. I, we modify the threshold opening for group signatures in [9] to allow for multiple, say

q, attributes. To achieve this, the user first picks q polynomials of degree $t_O - 1$, where t_O is the threshold number of openers needed to open the credential, to generate the Shamir secret shares of the attributes (Eq. 3). These Shamir shares are then used to generate the opening share (Eq. 4) which are encrypted using the public keys of the openers similar to what is done in [9]. The correctness is validated using Eq. 7 as we use a commitment (Eq. 2) to hide the user attributes from the validators.

Let $\mathcal{P}_j \in \mathbb{Z}_p[x]$ be a polynomial of degree $t_O - 1$ with coefficients $p_{j,1}, p_{j,2}, \ldots, p_{j,t_O-1}$ chosen arbitrarily from $\mathbb{Z}_p$,

$$\mathcal{P}_j = a_j + \sum_{l=1}^{t_O-1} p_{j,l} x^l. \tag{3}$$

For every opener $\mathcal{O}_k$, $k \in [n_O]$, the user generates a Shamir secret share $s_k = (s_{k,1}, \ldots, s_{k,q})$ of $(a_1, \ldots, a_q)$ and then calculates the k-th opening share as

$$\mu_k = \sum_{j=1}^{q} s_{k,j} \tilde{\beta}_j, \tag{4}$$

where $\tilde{\beta}_j$ is a part of the public key of the validators. The user encrypts μ_k with F_k, the public key of the opener $\mathcal{O}_k$, and a random $r_k \in \mathbb{Z}_p$ to obtain U_k, the encrypted opening share of the opener $\mathcal{O}_k$ as

$$U_k = (U_{k,1}, U_{k,2}) = (r_k \tilde{G},\ r_k F_k + \mu_k). \tag{5}$$

These shares are sent to the public ledger along with their ZKPoKs for which the user has to hide the coefficients of the polynomial by constructing

$$H_{j,l} = p_{j,l} H,\ j \in [q] \text{ and } l \in [t_O - 1]. \tag{6}$$

The user generates the ZKPoK π_k for each encrypted opening share, U_k to prove the correctness of the share as follows,

$$\pi_k = \text{ZKPoK}\{(r_k, \{o_j\}_{j=1}^{q}) : U_{k,1} = r_k \tilde{G} \wedge$$
$$e(H, U_{k,2} - r_k F_k) = \prod_{j=1}^{q} e(X_j - o_j G + \sum_{l=1}^{t_O-1} k^l H_{j,l}, \tilde{\beta}_j)\}. \tag{7}$$

The user sends the Vcerts and the request parameters $\Lambda = (C_a, \{X_j\}_{j=1}^{q}, \{(U_k, \pi_k)\}_{k=1}^{n_O}, \pi_s)$ over the ledger.

Correctness of credential issuance for the multiple attribute extension. As stated before, the threshold opening protocol in [9] works only for a single attribute, whereas credentials typically contain more. We have extended it to allow for multiple attributes and we show below the correctness of this extension w.r.t the issuance phase. That is, we show that the construction of opening shares during the issuance phase is correct and that it can be verified by the pairing check we provide.

Consider the pairing equation in Eq 7. Expanding the LHS, we get

$$\begin{aligned} e(H, U_{k,2} - r_k F_k) &= e(H, \mu_k) && \text{(from Eq 5)}, \\ &= e(H, \sum_{j=1}^{q} s_{k,j} \tilde{\beta}_j) && \text{(from Eq 4)}. \end{aligned}$$

Expanding the RHS, we see below that we get the same expression.

$$\begin{aligned} &\prod_{j=1}^{q} e(X_j - o_j G + \sum_{l=1}^{t_O-1} k^l H_{j,l}, \tilde{\beta}_j) \\ &= \prod_{j=1}^{q} e(a_j H + \sum_{l=1}^{t_O-1} k^l H_{j,l}, \tilde{\beta}_j) \quad \text{(from Eq 2)}, \\ &= \prod_{j=1}^{q} e(s_{k,j} H, \tilde{\beta}_j) \quad \text{(from Eqs 3 and 6)}, \\ &= \prod_{j=1}^{q} e(H, s_{k,j} \tilde{\beta}_j) \\ &= e(H, \sum_{j=1}^{q} s_{k,j} \tilde{\beta}_j). \end{aligned}$$

Therefore, this equality check can be used to prove that the k-th opening share is constructed correctly.

The user then generates the ZKPoK π_s to prove the authenticity of the attributes in zero-knowledge as follows,

$$\begin{aligned} \pi_s = \text{ZKPoK}\{(\{a_j\}_{j=1}^{q}, o, \{o_j\}_{j=1}^{q}, r) : \\ \mathit{VerifyVcerts}(\mathit{Vcert}, pk_{\mathcal{CP}}) \wedge \\ C = \mathit{Commit}(a_1, \ldots, a_q, sk_u; r) \wedge \\ C_a = oG + \sum_{j=1}^{q} a_j H_j \wedge X_j = o_j G + a_j H\}. \end{aligned} \tag{8}$$

Note that anyone with access to the public keys of the openers can verify the correctness of the opening shares but only the openers can decrypt it. On successful verification of the proofs, each opener decrypts his respective opening share and updates his registry for the user with the session identifier $reqid$ as

$$\mathit{Reg}_k[reqid] = \mu_k. \tag{9}$$

Partial Credentials Issuance. Each validator $\mathcal{V}_i$, $i \in [n_V]$ issues the blind signature $\tilde{\sigma}_i$ using the *BlindSign* method to the user after verifying the Vcerts and ZKPoKs and sends it through the ledger. The user retrieves this $\tilde{\sigma}_i$ and unblinds it to compute Pcred$_i(\sigma_i)$ as follows.

1) *BlindSign(*sk_i*,* $\{X_j\}_{j=1}^{q}$*,* C_a*,* $\{(U_k, \pi_k)\}_{k=1}^{n_O}$*,* π_s*)* $\to$ *(*$\tilde{\sigma}_i$*).* Each validator computes $H = \mathcal{RO}(C_a)$ and verifies the ZKPoKs, π_s and π_k, $k \in [n_O]$. On successful verification, each validator computes $\tilde{S}_i = (x_i H + \sum_{j=1}^{q} y_{i,j} X_j)$, using its secret key

$(x_i, \{y_{i,j}\}_{j=1}^q)$, and sends the blind signature, $\tilde{\sigma_i} = (H, \tilde{S_i})$ to the user.

2) *Unblind($\tilde{\sigma_i}$, $\{o_j\}_{j=1}^q$) → (σ_i).* Let o_j, $j \in [q]$ be the blinding factors used to build the commitment, X_j. It parses $\tilde{\sigma_i}$ as $(H, \tilde{S_i})$ and unblinds the signature to obtain the partial credential,

$$\sigma_i = (H, \tilde{S_i} - \sum_{j=1}^{q} o_j \tilde{\beta}_j). \tag{10}$$

Credential Aggregation. Upon receiving a threshold number of Pcreds σ_i, $i \in T$, where T is a set of validator indices from which the user receives the Pcred, the user aggregates them locally using the *AggCred* method (Eq. 11) to form an anonymous credential σ.

- *AggCred($\{\sigma_i\}_{i \in T}$) → (σ).* Each σ_i is parsed as (H, S_i). The credential σ after aggregation is given by

$$\sigma = (H, S) = (H, \sum_{i \in T} w_i S_i), \tag{11}$$

where w_i is the Lagrange coefficient, defined as

$$w_i = \prod_{j \in T \setminus \{i\}} \frac{j}{(j-i)} \bmod p. \tag{12}$$

C. VERIFICATION PHASE

Before verification, the service provider ($\mathcal{SP}$) collects and aggregates the verification keys of the validators to obtain pk_v as $(\tilde{G}, \tilde{\alpha}, \{\beta_j, \tilde{\beta}_j\}_{j=1}^q)$. This process happens only once before any request.

1) ***ProveCred***$(pk_v, a_1, \ldots, a_q, \sigma) \to (\kappa, \nu, \sigma', \pi_v)$.
Here, σ is parsed as (H, S) and pk_v as $(\tilde{G}, \tilde{\alpha}, \{\beta_j, \tilde{\beta}_j\}_{j=1}^q)$. The user chooses $r, r' \in \mathbb{Z}_p^2$ and does the following : 1. randomizes the credential σ to obtain $\sigma' = (H', S') = (r'H, r'S)$, 2. computes $\kappa = \tilde{\alpha} + \sum_{j=1}^q a_j \tilde{\beta}_j + r\tilde{G}$ and $\nu = rH'$ which are required later to verify σ' without disclosing the hidden attributes, and 3. generates the ZKPoK

$$\pi_v = \text{ZKPoK}\{(a_1, \ldots, a_q, r) : \kappa = \tilde{\alpha} + \sum_{j=1}^{q} a_j \tilde{\beta}_j + r\tilde{G} \wedge \ \nu = rH'\}. \tag{13}$$

2) ***VerifyCred***$(pk_v, \sigma', \kappa, \nu, \pi_v\ ) \to (true/false)$. Here, σ' is parsed as (H', S') and the following checks are done: 1. $H' \neq 1_{\mathbb{G}}$, and 2. verifies π_v and the following pairing equation

$$e(H',\ \kappa) = e(S' + \nu,\ \tilde{G}). \tag{14}$$

On successful verification it returns $true$, else $false$. The verification process is the same as that of the Coconut scheme and therefore we omit the proof of correctness of the pairing check.

D. OPENING PHASE

In the opening phase, the openers are asked to revoke the anonymity of a randomized credential σ'. The protocol is run between all the openers, and the output is the issuance session-id, $reqid$, corresponding to the session when the user was originally issued the credential. Let $\mathcal{O}$ be the set of participating openers with their respective private registries Reg_k, $k \in \mathcal{O}$. The following two methods are executed by all the participating openers.

1) ***PreOpening***$(Reg_k,\ \sigma')$. σ' is parsed as (H', S'). To reveal the user's identity, each opener $\mathcal{O}_k$ calculates

$$T_{reqid,k} = e(H', Reg_k[reqid]), \tag{15}$$

for every $reqid$ in his registry Reg_k. Then he broadcasts $\{(reqid, T_{reqid,k})\}_{reqid \in Reg_k}$ to all the other participating openers.

2) ***OpenCred***$(\{\{(reqid, T_{reqid,k})\}_{reqid \in Reg_k}\}_{k \in \mathcal{O}}, \sigma', \tilde{\alpha})$.
The opening of a credential is essentially the verification of a signature. Camenisch *et al.* [9] uses the latest version of PS signatures [30] in his opening scheme while the Coconut scheme uses an earlier version [29]. So we need to modify the opening equation in [9, Pt. 5, *GOpen* Algorithm, Sec. 4.2] to support PS signatures of the earlier version. (Note that the scheme in [9] uses an additional scalar that we do not need here.) Here, σ' is parsed as (H', S'). For each $reqid$, the openers verify

$$e(H', \tilde{\alpha}) \prod_{j \in \mathcal{O}} T_{reqid,j}^{w_j} = e(S', \tilde{G}), \tag{16}$$

where $w_j = \prod_{l \in \mathcal{O} \setminus \{j\}} \frac{l}{(l-j)}$ is the j-th Lagrange coefficient. The $reqid$ for which the above equation is successful is the user corresponding to σ'.

Correctness of credential opening for the multiple attribute extension. We now show the correctness of DTRAC's multiple attribute extension of threshold opening w.r.t. the opening phase. That is, we show that the pairing check that the openers do is valid for the $reqid$ that needs to be revoked.

Expanding the LHS of Eq.16, we get

$$\begin{aligned}
&e(H', \tilde{\alpha}) \prod_{j\in\mathcal{O}} T_{reqid,j}^{w_j} \\
&= e(H', \tilde{\alpha}) \prod_{j\in\mathcal{O}} e(H', Reg_j[reqid])^{\,w_j} \quad \text{(from Eq 15)}, \\
&= e(H', \tilde{\alpha}) \prod_{j\in\mathcal{O}} e(H', \mu_j)^{\,w_j} \quad \text{(from Eq 9)}, \\
&= e(H', \tilde{\alpha}) \prod_{j\in\mathcal{O}} e(H', \sum_{i=1}^{q} s_{j,i}\tilde{\beta}_i)^{\,w_j} \quad \text{(from Eq 4)}, \\
&= e(H', \tilde{\alpha}) \prod_{j\in\mathcal{O}} e(H',\ w_j \sum_{i=1}^{q} s_{j,i}\tilde{\beta}_i), \\
&= e(H', \tilde{\alpha}) \prod_{j\in\mathcal{O}} e(H', w_j s_{j,1}\tilde{\beta}_1 + \cdots + w_j s_{j,q}\tilde{\beta}_q), \\
&= e(H', \tilde{\alpha}) e(H', \sum_{j\in\mathcal{O}} w_j s_{j,1}\tilde{\beta}_1 + \cdots + \sum_{j\in\mathcal{O}} w_j s_{j,q}\tilde{\beta}_q), \\
&= e(H', \tilde{\alpha} + \sum_{i=1}^{q} a_i\tilde{\beta}_i) \quad \text{(since } a_i = \sum_{j\in\mathcal{O}} w_j s_{j,i}), \\
&= e(S', \tilde{G}).
\end{aligned}$$

Given (H', S') is a PS signature on $(a_1, \ldots, a_q)$, the last equation holds since it is the verification equation of a PS signature.

We also provide a detailed security analysis in the UC framework in Sec. VI to ensure that the integration of threshold opening with threshold issuance does not affect the security and privacy of the anonymous credential scheme.

VI. SECURITY ANALYSIS IN THE UC FRAMEWORK

A. IDEAL FUNCTIONALITY $\mathcal{F}_{\mathcal{AC}}$

We first describe the ideal functionality $\mathcal{F}_{\mathcal{AC}}$ of our scheme that supports threshold issuance and threshold opening. For the former, $\mathcal{F}_{\mathcal{AC}}$ interacts with the validators $\{\mathcal{V}_1, \ldots, \mathcal{V}_{n_V}\}$ and for the latter, $\mathcal{F}_{\mathcal{AC}}$ interacts with the openers $\{\mathcal{O}_1, \ldots, \mathcal{O}_{n_O}\}$. The threshold values of the validators and openers are t_V and t_O, respectively, i.e., $\mathcal{F}_{\mathcal{AC}}$ assumes that at most $t_V - 1$ validators and at most $t_O - 1$ openers are corrupt. The other entities that $\mathcal{F}_{\mathcal{AC}}$ interacts with include a user $\mathcal{U}_j$, from the set of all users $\mathcal{U}$, and a service provider $\mathcal{SP}$. The user takes a pseudonym from the universe of pseudonyms, $\mathbb{U}_p$ to present a credential to $\mathcal{SP}$, presents authenticated Vcerts from the universe of Vcert validations, $\mathbb{U}_\phi$ during credential issuance, and shows a truth statement relating to attributes during credential show from the universe of statements relating to attributes, $\mathbb{U}_\varphi$. $\mathcal{F}_{\mathcal{AC}}$ stores and initializes three sets: $\mathcal{L}_{REQ}$ for credential requests, $\mathcal{L}_{ISS}$ for credential issuances and $\mathcal{L}_{VER}$ for credential presentations. Each instance of $\mathcal{F}_{\mathcal{AC}}$ is identified by a session identifier $sid = (\mathcal{P}, sid')$ where $\mathcal{P} = (\mathcal{V}_1, \ldots, \mathcal{V}_{n_V}, \mathcal{O}_1, \ldots, \mathcal{O}_{n_O})$ and sid' is a random value that changes with each instantiation of $\mathcal{F}_{\mathcal{AC}}$. The value qid is used to identify messages that are being communicated between entities. These messages are stored temporarily and deleted soon after the simulator responds so as to avoid replay attacks by the simulator. Since they are only temporarily stored we do not specify exactly where these messages will be stored. Note that when we say a fresh id is generated we mean that the value is unique across sid, $reqid$, qid and vid, where $reqid$ and vid are session ids that uniquely identify credential request and credential show, respectively.

Setup:
1) The validators and openers inform $\mathcal{F}_{\mathcal{AC}}$ that their initialization is done. Each entity executes its initialization as part of setup and a flag is set to indicate its successful completion.
2) $\mathcal{F}_{\mathcal{AC}}$ verifies $n_V \geq t_V$ and $n_O \geq t_O$.

Credential request:
1) When $\mathcal{F}_{\mathcal{AC}}$ receives the request $(sid, a = (a_1, \ldots, a_q), \phi)$ from a user $\mathcal{U}_j$, it creates a fresh qid and saves $(qid, \mathcal{U}_j, a, \phi)$.
2) $\mathcal{F}_{\mathcal{AC}}$ sends (sid, qid) to the simulator $\mathcal{S}$ and $\mathcal{S}$ sends it back to $\mathcal{F}_{\mathcal{AC}}$ to simulate the real-world communication between the user and the validators and the openers.
3) If $\phi \in \mathbb{U}_\phi$ and $\phi(a) = 1$, i.e. if the attributes $(a_1, \ldots, a_q)$ have been authenticated by the respective certifiers, $\mathcal{F}_{\mathcal{AC}}$ creates and stores a new record $(reqid, \mathcal{U}_j, a, \phi)$ in $\mathcal{L}_{REQ}$ with a fresh $reqid$. This tuple ensures that the validators and the openers can later request the $\mathcal{F}_{\mathcal{AC}}$ to issue and open the credential, respectively, if the attributes of $\mathcal{U}_j$ were authenticated by the certifiers.
4) $\mathcal{F}_{\mathcal{AC}}$ outputs $(sid, reqid, \phi, \mathcal{U}_j)$ to all the parties in $\mathcal{P}$. Note that a is always kept secret from the validators and the openers.
5) $\mathcal{F}_{\mathcal{AC}}$ deletes the record $(qid, \mathcal{U}_j, a, \phi)$ after all the validators and the openers have received the credential request.

Credential issuance:
1) Each validator now has $(sid, reqid, \phi, \mathcal{U}_j)$ of the user, $\mathcal{U}_j$. It sends $(sid, reqid)$ to $\mathcal{F}_{\mathcal{AC}}$. If a request with $reqid$ is not present in $\mathcal{L}_{REQ}$, $\mathcal{F}_{\mathcal{AC}}$ aborts.
2) $\mathcal{F}_{\mathcal{AC}}$ creates a fresh qid and stores $(qid, \mathcal{U}_j, a, \phi, \mathcal{V}_i, reqid)$ to indicate that $\mathcal{V}_i$ wishes to issue a partial credential to $\mathcal{U}_j$ on attribute a.
3) $\mathcal{F}_{\mathcal{AC}}$ sends (sid, qid) to the simulator $\mathcal{S}$ and $\mathcal{S}$ sends it back to $\mathcal{F}_{\mathcal{AC}}$ to simulate the real-world communication between the validator $\mathcal{V}_i$ and the user $\mathcal{U}_j$.
4) $\mathcal{F}_{\mathcal{AC}}$ stores $(reqid, \mathcal{U}_j, a, \mathcal{V} \cup \mathcal{V}_i, \mathcal{O})$ in $\mathcal{L}_{ISS}$ which indicates that the user now has partial credentials from the set of validators, $\mathcal{V} \cup \mathcal{V}_i$.
5) $\mathcal{F}_{\mathcal{AC}}$ outputs $(sid, a, \phi, \mathcal{V}_i)$ to $\mathcal{U}_j$.
6) $\mathcal{F}_{\mathcal{AC}}$ deletes the record $(qid, \mathcal{U}_j, a, \phi, \mathcal{V}_i, reqid)$ to indicate that $\mathcal{V}_i$ has issued the credential request.

Update information of openers:
1) Each opener now has $(sid, reqid, \phi, \mathcal{U}_j)$ of the user,

$\mathcal{U}_j$. It sends $(sid, reqid)$ to $\mathcal{F}_{\mathcal{AC}}$. If a request with $reqid$ is not present in $\mathcal{L}_{REQ}$, $\mathcal{F}_{\mathcal{AC}}$ aborts.

2) $\mathcal{F}_{\mathcal{AC}}$ stores $(reqid, \mathcal{U}_j, a, \mathcal{V}, \mathcal{O} \cup \mathcal{O}_k)$ in $\mathcal{L}_{ISS}$ which indicates that the opener $\mathcal{O}_k$ has now updated its private registry corresponding to $reqid$.

Presenting the credential (Credential Show) to $\mathcal{SP}$:

1) When $\mathcal{F}_{\mathcal{AC}}$ receives the service request $(sid, a = (a_1, \ldots, a_q), \varphi, P, \mathcal{SP})$ from a user $\mathcal{U}_j$ with a pseudonym P, it creates a fresh qid and saves $(qid, \varphi, \mathcal{SP}, a, P)$.
2) $\mathcal{F}_{\mathcal{AC}}$ sends (sid, qid) to the simulator $\mathcal{S}$ and $\mathcal{S}$ sends it back to $\mathcal{F}_{\mathcal{AC}}$ to simulate the real-world communication between the user $\mathcal{U}_j$ and the $\mathcal{SP}$.
3) If $P \in \mathbb{U}_p$ and a record $(reqid, \mathcal{U}_j, a, \mathcal{V}, \mathcal{O})$ is stored such that $|\mathcal{V}| \geq t_V$ if $\mathcal{U}_j$ is honest, or $|\mathcal{V}| \geq t'_V$ if $\mathcal{U}_j$ is corrupt, where $t'_V = t_V - \tilde{t}_V$ ($\tilde{t}_V$ is the number of corrupt validators), and $\varphi(a) = 1$, i.e. the attributes $a_1, \ldots, a_q$ satisfy the service policy, $\mathcal{F}_{\mathcal{AC}}$ creates and stores a new record with a fresh vid, (vid, a) in $\mathcal{L}_{VER}$. This tuple uniquely identifies a credential show.
4) $\mathcal{F}_{\mathcal{AC}}$ outputs (sid, φ, vid, P) to $\mathcal{SP}$.
5) $\mathcal{F}_{\mathcal{AC}}$ deletes the record $(qid, \varphi, \mathcal{SP}, a, P)$ to indicate that the service request has been taken care of.

Opening the credential:

1) Suppose $\mathcal{F}_{\mathcal{AC}}$ receives an open request (sid, vid) from an opener $\mathcal{O}_k$. If there is a credential with vid in $\mathcal{L}_{VER}$, $\mathcal{F}_{\mathcal{AC}}$ stores the record $(vid, \mathcal{O}' \cup \mathcal{O}_k)$. This indicates that the set of openers, $\mathcal{O}' \cup \mathcal{O}_k$, has agreed to open the credential.
2) $\mathcal{F}_{\mathcal{AC}}$ sends (sid, vid) to all the other openers and $\mathcal{S}$ to simulate the real-world communication of the openers.
3) $\mathcal{F}_{\mathcal{AC}}$ (and thus $\mathcal{O}_k$) waits until the condition $|\mathcal{O}'| \geq t_O$ is true, i.e. until a threshold number of openers agree.
4) If a record $(\mathcal{U}_j, reqid', a, \mathcal{V}, \mathcal{O})$ is present in $\mathcal{L}_{REQ}$ such that $|\mathcal{O} \cap \mathcal{O}'| \geq t_O$, i.e. if the threshold number of openers have updated their private registry corresponding to attribute a, set $reqid = reqid'$ else set $reqid = \perp$.
5) $\mathcal{F}_{\mathcal{AC}}$ outputs $(sid, vid, reqid)$ to $\mathcal{O}_k$.

$\mathcal{F}_{\mathcal{AC}}$ guarantees the following security properties.

1) *Unforgeability*. $\mathcal{F}_{\mathcal{AC}}$ ensures that the attributes $a = a_1, \ldots, a_q$ that satisfy φ have to be validated by at least t_V validators, i.e. at most one honest validator has to issue a partial credential.
2) *Authenticity and Blindness*. A validator never gets to see the attributes $a = a_1, \ldots, a_q$ and only learns that the attributes satisfy ϕ ensuring authenticity and blindness.
3) *Optional Unlinkability*. $\mathcal{F}_{\mathcal{AC}}$ does not impose any restriction on the value of the pseudonym when the user presents the credential to the $\mathcal{SP}$ which allows for optional unlinkability. If different pseudonyms are used then it provides unlinkability, else no.
4) *Accountability and Unanimity*. When at least a threshold number of openers agree, then the $reqid$ is revealed by the $\mathcal{F}_{\mathcal{AC}}$, thus providing accountability. All the openers open to the same $reqid$ which ensures unanimity.

B. REAL-WORLD PROTOCOL $\Pi_{\mathcal{AC}}$

We define the real-world protocol $\Pi_{\mathcal{AC}}$ corresponding to the ideal functionality $\mathcal{F}_{\mathcal{AC}}$. $\Pi_{\mathcal{AC}}$ makes use of the following ideal functionalities: $\mathcal{F}_{\mathcal{KG}}$ for key generation of validators and key registration of openers, $\mathcal{F}_{\mathcal{BC}}$ for broadcasting messages, $\mathcal{F}_{\mathcal{AUTH}}$ for transmitting authenticated messages, $\mathcal{F}_{\mathcal{NYM}}$ for pseudonymous communication and $\mathcal{F}_{\mathcal{RO}}$ for random oracle functionality, to securely realize $\mathcal{F}_{\mathcal{AC}}$ in the hybrid-world. These ideal functionalities are described in more detail in Appendix B. We assume that every entity in the system implicitly queries $\mathcal{F}_{\mathcal{KG}}$ to retrieve the public keys of the validators and the openers. $\Pi_{\mathcal{AC}}$ uses five types of storages: $\mathcal{L}_{\mathcal{U}_j,REQ}$ for each user $\mathcal{U}_j$ to store the information in the credential request which is to be used later by $\mathcal{U}_j$ after receiving the blind signatures from the validators, $\mathcal{L}_{\mathcal{V}_i,REQ}$ and $\mathcal{L}_{\mathcal{O}_k,REQ}$ for each validator $\mathcal{V}_i$ and each opener $\mathcal{O}_k$, respectively, to store the successful credential requests, $\mathcal{L}_{ISS}$ for each user to store the credentials, and $\mathcal{L}_{VER}$ for the $\mathcal{SP}$ to store the service requests.

Setup:

1) A validator $\mathcal{V}_i$ obtains their signing key pair (sk_i, pk_i) and the public key pk for credential verification from $\mathcal{F}_{\mathcal{AC}}$.
2) An opener $\mathcal{O}_k$ generates its public-private key pair (opk_k, osk_k) and requests $\mathcal{F}_{\mathcal{AC}}$ to register its public key.
3) The setup flag is set to true to indicate that all the validators and the openers have executed their setup initialization.

Credential request:

1) A user $\mathcal{U}_j$ with attributes $a = (a_1, \ldots, a_q)$ computes a commitment on the attributes, C_a (Eq. 1).
2) The user $\mathcal{U}_j$ then sends C_a to $\mathcal{F}_{\mathcal{RO}}$ and receives H (Eq. 1), where $\mathcal{F}_{\mathcal{RO}}$ ensures that the validators agree on a common randomness H.
3) $\mathcal{U}_j$ computes the commitment X_j on the individual attribute a_j (Eq. 2) which is used by the validators to blind sign the user attributes.
4) $\mathcal{U}_j$ also computes the opening shares for the openers and encrypts the k-th opening share for each $\mathcal{O}_k$ using the public key of the k-th opener(Eq. 5).
5) $\mathcal{U}_j$ computes $\{H_{j,l}\}j \in [q], l \in [t_O - 1]$ and π_k to help verify the validity of the k-th encrypted opening share, U_k (Eqs. 6, 7).
6) $\mathcal{U}_j$ picks $reqid \xleftarrow{\$} \mathbb{Z}_p$, stores $(sid, a, o_1, \ldots, o_q, \Lambda = (C_a, \{X_j\}_j \in [q], H), \phi, reqid)$ in $\mathcal{L}_{\mathcal{U}_j,REQ}$, where q is the number of attributes, o_j is a blinding factor (Eq. 2) used in the commitment X_j and Λ is a tuple that contains the commitments and H.
7) $\mathcal{U}_j$ sends $(\langle \Lambda, \pi_s, \{(U_k, \pi_k)\}_{k \in [n_O]}, \{H_{j,l}\}_{j \in [q], l \in [t_O - 1]},$

$\phi, reqid\rangle, \mathcal{P})$, a credential request, to $\mathcal{F}_{\mathcal{BC}}$ that sends the request to all the validators and openers.

8) Each validator $\mathcal{V}_i$ and opener $\mathcal{O}_k$ receives $(\langle\Lambda, \pi_s, \{(U_k, \pi_k)\}_{k\in[n_O]}, \{H_{j,l}\}_{j\in[q],l\in[t_O-1]}, \phi, reqid\rangle, \mathcal{U}_j, \mathcal{P})$ from $\mathcal{F}_{\mathcal{BC}}$, where the set $\mathcal{P}$ convinces the validators that all the openers were sent the credential request.
9) They send C_a to $\mathcal{F}_{\mathcal{RO}}$ and receives H'.
10) If $H' = H$ and the ZKPoKs sent π_s and $\{\pi_k\}_{k\in[n_O]}$ are valid, $\mathcal{V}_i$ stores $(sid, \mathcal{U}_j, reqid, C_a, \{X_j\}_{j\in[q]}, H, \mathcal{V}_i)$ in $\mathcal{L}_{\mathcal{V}_i,REQ}$ and $\mathcal{O}_k$ stores $(sid, \mathcal{U}_j, reqid, H, U_k, \mathcal{O}_k)$ in $\mathcal{L}_{\mathcal{O}_k,REQ}$ to indicate that the credential request has been successfully verified.

Credential issuance:

1) Each validator $\mathcal{V}_i$ now has $(sid, \mathcal{U}_j, reqid, C_a, \{X_j\}_{j\in[q]}, H, \mathcal{V}_i)$ in $\mathcal{L}_{\mathcal{V}_i,REQ}$ for a user, $\mathcal{U}_j$.
2) $\mathcal{V}_i$ computes the blind signature $\tilde{\sigma_i}$ using H and $\{X_j\}_{j\in[q]}$ (see *BlindSign* method in Sec. V-B).
3) $\mathcal{V}_i$ sends $(\langle reqid, \tilde{\sigma_i}\rangle, \mathcal{U}_j)$ to $\mathcal{F}_{\mathcal{AUTH}}$.
4) $\mathcal{U}_j$ receives $(\langle reqid, \tilde{\sigma_i}\rangle, \mathcal{V}_i))$ from $\mathcal{F}_{\mathcal{AUTH}}$ and unblinds $\tilde{\sigma_i}$ to σ_i (Eq. 10).
5) $\mathcal{U}_j$ updates the entry $(reqid, a, \mathcal{V} \cup (\mathcal{V}_i, \sigma_i), \sigma)$ in $\mathcal{L}_{ISS}$, where $\mathcal{V}$ is initialized to $\emptyset$ and σ is initialized with $\perp$.
6) If $|\mathcal{V}| \geq t_V$, $\mathcal{U}_j$ computes the credential σ using the *AggCred* method and σ_i's as input (Eq. 11) and updates the above entry to $(reqid, a, \mathcal{V}, \sigma)$.

Update information of openers:

1) Each opener $\mathcal{O}_k$ now has $(sid, \mathcal{U}_j, reqid, H, U_k, \mathcal{O}_k)$ in $\mathcal{L}_{\mathcal{O}_k,REQ}$ from a user, $\mathcal{U}_j$.
2) $\mathcal{O}_k$ decrypts the k-th opening ciphertext U_k to obtain μ_k and appends its local registry as follows: $Reg_k[reqid] = \mu_k$ (see Eq. 9).

Presenting the credential (Credential Show) to $\mathcal{SP}$:

1) $\mathcal{U}_j$ now has a record $(reqid, a, \mathcal{V}, \sigma)$ that contains the credential, σ.
2) $\mathcal{U}_j$ randomizes the credential σ as σ' and computes κ and ν (see *ProveCred* method in Sec. V-C).
3) $\mathcal{U}_j$ generates the ZKPoK π_v (Eq. 13) that convinces the verifier of $\mathcal{U}_j$'s knowledge of the attributes $a = (a_1, \ldots, a_q)$.
4) $\mathcal{U}_j$ picks a pseudonym P and sends the service request $(\langle\sigma', \varphi, \kappa, \nu, \pi_v\rangle, P, \mathcal{SP})$ to $\mathcal{F}_{\mathcal{NYM}}$ that sends the request to $\mathcal{SP}$.
5) $\mathcal{SP}$ on receiving $(\langle\sigma', \varphi, \kappa, \nu, \pi_v\rangle, P)$ from $\mathcal{F}_{\mathcal{NYM}}$ verifies σ' using κ, ν and the public key of the validators, pk (Eq. 14).
6) If the ZKPoK π_v is valid, $\mathcal{SP}$ picks a fresh vid to uniquely identify the credential presentation and records (vid, σ') in $\mathcal{L}_{VER}$.

Opening the credential:

1) Each opener $\mathcal{O}_k$ is asked to revoke the anonymity of the user corresponding to the session given by vid.
2) $\mathcal{O}_k$ finds the corresponding σ' in $\mathcal{L}_{VER}$ and computes $T_{reqid,k}$ (Eq. 15) for every $reqid$ stored in $\mathcal{O}_k$'s private registry, Reg_k. This helps to revoke the anonymity for a particular credential usage.
3) $\mathcal{O}_k$ sends the opening information to the other openers by sending $(\{T_{reqid,k}\}_{reqid\in Reg_k}, \mathcal{O} \setminus \mathcal{O}_k)$ to $\mathcal{F}_{\mathcal{BC}}$.
4) $\mathcal{O}_k$ waits till it receives $(\langle\{T_{reqid,l}\}_{reqid\in Reg_l}\rangle, \mathcal{O}_k, \mathcal{O}\setminus\mathcal{O}_k)$ from at least $t_O - 1$ number of openers, where $l \neq k$.
5) If the opening information for any $reqid$ satisfies Eq. 16, $\mathcal{O}_k$ outputs $reqid$, otherwise outputs $\perp$.

C. SIMULATOR DESIGN

We design a simulator $\mathcal{S}$ which interacts with an adversary $\mathcal{A}$ that controls $t_O - 1$ openers and $t_V - 1$ validators. The simulator, $\mathcal{S}$ is an algorithm that simulates both the honest parties that interact with a real-world adversary, $\mathcal{A}$ and the ideal world adversary that interacts with the ideal functionality, $\mathcal{F}_{\mathcal{AC}}$. We go on to show that the real-world adversary's view can be simulated by $\mathcal{S}$. The honest entity interaction with other honest entities is captured by the ideal functionality $\mathcal{F}_{\mathcal{AC}}$. We now describe the interaction between an honest entity and a corrupt entity. A simulator $\mathcal{S}$ does the setup by receiving the signing keys of the validators from $\mathcal{F}_{\mathcal{KG}}$ and generates the keys of the openers and registers the public keys of the openers with $\mathcal{F}_{\mathcal{KG}}$. The corrupt entities (adversaries) collect their respective keys from the simulator $\mathcal{S}$.

Honest $\mathcal{U}_j$ ***requests a credential from a corrupt*** $\mathcal{V}_i$. When $\mathcal{F}_{\mathcal{AC}}$ sends (sid, qid) to $\mathcal{S}$ in the credential request, $\mathcal{S}$ sends $(m, \mathcal{U}_j, \mathcal{P})$ to $\mathcal{A}$, where m is the credential request message that the honest $\mathcal{U}_j$ sends to the validators and the openers. When $\mathcal{F}_{\mathcal{AC}}$ sends $(sid, reqid, \phi, \mathcal{U}_j)$ to a corrupt validator $\mathcal{V}_i$, $\mathcal{S}$ runs a copy of $\mathcal{U}_j$ with the input (sid, a', ϕ) where a' is either set to a, if $\mathcal{U}_j$ stores a tuple $(sid, a, o_1, \ldots, o_q, \Lambda, \{(U_k, \pi_k)\}_{k\in[n_O]}, \{H_{j,l}\}_{j\in[q],l\in[t_O-1]}, \phi, reqid)$, else a' is chosen at random. $\mathcal{S}$ computes the simulated proofs π_s and $\{\pi_k\}_{k\in[n_O]}$, stores the tuple $(sid, a, o_1, \ldots, o_q, \Lambda, \{(U_k, \pi_k)\}_{k\in[n_O]}, \{H_{j,l}\}_{j\in[q],l\in[t_O-1]}, \phi, reqid)$ and sends the message $(\langle\Lambda, \pi_s, \{(U_k, \pi_k)\}_{k\in[n_O]}, \{H_{j,l}\}_{j\in[q],l\in[t_O-1]}, \phi, reqid\rangle, \mathcal{P})$ to $\mathcal{F}_{\mathcal{BC}}$. When $\mathcal{F}_{\mathcal{BC}}$ sends $(\langle\Lambda, \pi_s, \{(U_k, \pi_k)\}_{k\in[n_O]}, \{H_{j,l}\}_{j\in[q],l\in[t_O-1]}, \phi, reqid\rangle, \mathcal{U}_j, \mathcal{P})$, $\mathcal{S}$ forwards that message to $\mathcal{A}$.

Honest $\mathcal{U}_j$ ***sends opening information to corrupt*** $\mathcal{O}_k$. Here, the simulator $\mathcal{S}$ acts similar to "Honest $\mathcal{U}_j$ requests a credential from a corrupt $\mathcal{V}_i$". But here $\mathcal{F}_{\mathcal{AC}}$ sends $(sid, reqid, \phi, \mathcal{U}_j)$ to the corrupt opener $\mathcal{O}_k$ instead of a corrupt validator $\mathcal{V}_i$.

Corrupt $\mathcal{U}_j$ ***requests a credential from an honest*** $\mathcal{V}_i$. When an adversary $\mathcal{A}$ sends $(\langle\Lambda, \pi_s, \{(U_k, \pi_k)\}_{k\in[n_O]}, \{H_{j,l}\}_{j\in[q],l\in[t_O-1]}, \phi, reqid\rangle, \mathcal{P})$ to $\mathcal{F}_{\mathcal{BC}}$, S runs $\mathcal{F}_{\mathcal{BC}}$ with that message. $\mathcal{F}_{\mathcal{BC}}$ sends $(\langle\Lambda, \pi_s, \{(U_k, \pi_k)\}_{k\in[n_O]}, \{H_{j,l}\}_{j\in[q],l\in[t_O-1]}, \phi, reqid\rangle, \mathcal{A}, \mathcal{P})$ to $\mathcal{V}_i$. If $\mathcal{A}$ had already sent that message then $\mathcal{S}$ retrieves the stored tuple $(sid, \langle\Lambda, \{(U_k, \pi_k)\}_{k\in[n_O]}, \phi, reqid\rangle,$

$\langle a_1, \ldots, a_q, o, o_1, \ldots, o_q\rangle)$. Otherwise, $\mathcal{S}$ performs the following steps.

1) $\mathcal{S}$ verifies π_s and $\{\pi_k)\}_{k\in[n_O]}$ using ϕ, pk, and $(C_a, \{X_j\}_{j\in[q]}, H, \{U_k\}_{k\in[n_O]})$. Abort if any of the proofs are not correct.
2) $\mathcal{S}$ runs an extractor algorithm for the ZKPoK, π_s to extract the witness $(a_1, \ldots, a_q, o, o_1, \ldots, o_q)$.
3) If there is a tuple $(sid, \langle \Lambda = (C'_a, \{X'_j\}_{j\in[q]}, H'), \{(U_k, \pi_k)\}_{k\in[n_O]}, \phi, reqid\rangle, \langle a'_1, \ldots, a'_q, o', o'_1, \ldots, o'_q\rangle)$ stored such that $C'_a = C_a$ but $(a'_1, \ldots, a'_q) \neq (a_1, \ldots, a_q)$, $\mathcal{S}$ outputs failure.
4) Otherwise, $\mathcal{S}$ stores the tuple $(sid, \langle \Lambda = (C_a, \{X_j\}_{j\in[q]}, H), \pi_s, \{(U_k, \pi_k)\}_{k\in[n_O]}, \phi, reqid\rangle, \langle a_1, \ldots, a_q, o, o_1, \ldots, o_q\rangle)$ and sets $a \leftarrow (a_1, \ldots, a_q)$.
5) $\mathcal{S}$ sends (sid, a, ϕ) to $\mathcal{F}_{\mathcal{AC}}$. When $\mathcal{F}_{\mathcal{AC}}$ sends (sid, qid), $\mathcal{S}$ sends (sid, qid) back to $\mathcal{F}_{\mathcal{AC}}$.

Honest $\mathcal{V}_i$ ***issues a credential to a corrupt*** $\mathcal{U}_j$. When $\mathcal{F}_{\mathcal{AC}}$ sends (sid, qid) during the credential issuance phase, $\mathcal{S}$ sends $(m, \mathcal{U}_j)$ to $\mathcal{A}$, where m is the blind signature message that the honest $\mathcal{V}_i$ sends to the user, $\mathcal{U}_j$. When $\mathcal{F}_{\mathcal{AC}}$ sends $(sid, a, \phi, \mathcal{V}_i)$ to $\mathcal{U}_j$, $\mathcal{S}$ finds the stored tuple $(sid, \langle \Lambda = (C_a, \{X_j\}_{j\in[q]}, H), \pi_s, \{(U_k, \pi_k)\}_{k\in[n_O]}, \phi, reqid\rangle, \langle a_1, \ldots, a_q, o, o_1, \ldots, o_q\rangle)$ with $a = (a_1, \ldots, a_q)$. $\mathcal{S}$ computes the blind signature $\tilde{\sigma_i}$, stores the tuple $(sid, a_1, \ldots, a_q, \phi, \mathcal{V}_i)$ and sends the message $\langle reqid, \tilde{\sigma_i}\rangle$ to $\mathcal{A}$.

Corrupt $\mathcal{V}_i$ ***issues a credential to an honest*** $\mathcal{U}_j$. When an adversary $\mathcal{A}$ sends the message $(\langle reqid, \tilde{\sigma_i}\rangle, \mathcal{U}_j)$, $\mathcal{S}$ runs $\mathcal{F}_{\mathcal{AUTH}}$ with that message. When $\mathcal{F}_{\mathcal{AUTH}}$ sends $(\langle reqid, \tilde{\sigma_i}\rangle, \mathcal{V}_i, \mathcal{U}_j))$, $\mathcal{S}$ runs the copy of $\mathcal{U}_j$ with $reqid$. When the copy of $\mathcal{U}_j$ outputs $(sid, a, \phi, \mathcal{V}_i)$, $\mathcal{S}$ sends $(sid, \mathcal{U}_j, reqid)$ to $\mathcal{F}_{\mathcal{AC}}$. When $\mathcal{F}_{\mathcal{AC}}$ sends (sid, qid), $\mathcal{S}$ forwards it to $\mathcal{F}_{\mathcal{AC}}$.

Honest $\mathcal{U}_j$ ***shows a credential to a corrupt*** $\mathcal{SP}$. When $\mathcal{F}_{\mathcal{AC}}$ sends (sid, qid), $\mathcal{S}$ leaks the length of m to $\mathcal{A}$, where $m = \langle \sigma', \varphi, \kappa, \nu, \pi_v\rangle$ is the message sent by an honest $\mathcal{U}_j$ to $\mathcal{SP}$. When $\mathcal{F}_{\mathcal{AC}}$ sends (sid, φ, vid, P) to a corrupt $\mathcal{SP}$, $\mathcal{S}$ sets the message to be sent to $\mathcal{A}$ as follows.

1) $\mathcal{S}$ picks random $t_1 \xleftarrow{\$} \mathbb{Z}_p$, $t_2 \xleftarrow{\$} \mathbb{Z}_p$ and $t_3 \xleftarrow{\$} \mathbb{Z}_p$ and computes $\sigma' \leftarrow (t_1 G, (t_1 t_2)G)$, $\kappa \leftarrow (t_2 + t_3)G_2$ and $\nu \leftarrow (t_1 t_3)G$. $\mathcal{S}$ generates a simulated proof π_v.
2) $\mathcal{S}$ sends the message $(\langle \sigma', \varphi, \kappa, \nu, \pi_v\rangle, P, \mathcal{SP})$ to $\mathcal{A}$.

Corrupt $\mathcal{U}_j$ ***shows a credential to an honest*** $\mathcal{SP}$. When an adversary $\mathcal{A}$ sends the message $(\langle \sigma', \varphi, \kappa, \nu, \pi_v\rangle, P, \mathcal{SP})$, $\mathcal{S}$ runs $\mathcal{F}_{\mathcal{NYM}}$ on that message. When $\mathcal{F}_{\mathcal{NYM}}$ leaks the length of the tuple, $l(\langle \sigma', \varphi, \kappa, \nu, \pi_v\rangle)$, $\mathcal{S}$ forwards it to $\mathcal{A}$. When $\mathcal{F}_{\mathcal{NYM}}$ sends $(\langle \sigma', \varphi, \kappa, \nu, \pi_v\rangle, P, \mathcal{SP})$, $\mathcal{S}$ verifies σ' and the proof π_v and proceeds as follows.

1) $\mathcal{S}$ runs the extractor algorithm to extract the witness $(a_1, \ldots, a_q, r)$ from the proof π_v.
2) $\mathcal{S}$ parses σ' as (H', S') and runs the verification equation $e(H',\ \tilde{\alpha} + \sum_{j=1}^{q} a_j \tilde{\beta}_j) = e(S',\ \tilde{G})$. If the verification fails, $\mathcal{S}$ outputs failure.
3) $\mathcal{S}$ checks that there are at least $t_V - \tilde{t}_V$ tuples $(sid, a_1, \ldots, a_q, \phi, \mathcal{V}_i)$ stored for $t_V - \tilde{t}_V$ different validators, where $\tilde{t}_V$ is the number of corrupt validators. If not, $\mathcal{S}$ outputs failure.
4) $\mathcal{S}$ sends $(sid, \varphi, P, \mathcal{SP})$ to $\mathcal{F}_{\mathcal{AC}}$. When $\mathcal{F}_{\mathcal{AC}}$ sends (sid, qid), $\mathcal{S}$ forwards it to $\mathcal{F}_{\mathcal{AC}}$.

Honest $\mathcal{O}_k$ ***sends opening information to a corrupt opener*** $\mathcal{O}_l$. When $\mathcal{F}_{\mathcal{AC}}$ sends (sid, vid), $\mathcal{S}$ sends $(vid, m, \mathcal{O} \setminus \mathcal{O}_k)$ to $\mathcal{F}_{\mathcal{BC}}$ and $\mathcal{A}$, where m is a message that contains the opening information that an honest opener sends to a corrupt opener. When $\mathcal{F}_{\mathcal{AC}}$ sends (sid, vid) to a corrupt opener $\mathcal{O}_l$, $\mathcal{S}$ runs a copy of the opener $\mathcal{O}_k$ on the input (sid, vid). When the copy of $\mathcal{O}_k$ sends the message $(\langle\{T_{reqid,k}\}_{reqid\in Reg_k}\rangle, \mathcal{O} \setminus \mathcal{O}_k)$ to $\mathcal{F}_{\mathcal{BC}}$, $\mathcal{S}$ runs $\mathcal{F}_{\mathcal{BC}}$ on that message. When $\mathcal{F}_{\mathcal{BC}}$ sends $(\langle\{T_{reqid,k}\}_{reqid\in Reg_k}\rangle, \mathcal{O}_k, \mathcal{O} \setminus \mathcal{O}_k)$, $\mathcal{S}$ forwards it to $\mathcal{A}$.

Corrupt $\mathcal{O}_k$ ***sends opening information to an honest opener*** $\mathcal{O}_l$. When $\mathcal{A}$ sends the message $(\langle\{T_{reqid,k}\}_{reqid\in Reg_k}, vid\rangle, \mathcal{O} \setminus \mathcal{O}_k)$, $\mathcal{S}$ runs $\mathcal{F}_{\mathcal{BC}}$ on that message. When $\mathcal{F}_{\mathcal{BC}}$ sends $(\langle\{T_{reqid,k}\}_{reqid\in Reg_k}, vid\rangle, \mathcal{O}_k, \mathcal{O} \setminus \mathcal{O}_k)$, $\mathcal{S}$ sends it to $\mathcal{A}$. When adversary $\mathcal{A}$ responds, $\mathcal{S}$ sends (sid, vid) to $\mathcal{F}_{\mathcal{AC}}$. When $\mathcal{F}_{\mathcal{AC}}$ sends (sid, vid), $\mathcal{S}$ sends (sid, vid) back to $\mathcal{F}_{\mathcal{AC}}$.

Theorem VI.1. *When a subset of validators up to $t_V - 1$ and a subset of openers up to $t_O - 1$ are corrupt, Π_{AC} securely realizes $\mathcal{F}_{\mathcal{AC}}$ in the ($\mathcal{F}_{\mathcal{KG}}$, $\mathcal{F}_{\mathcal{BC}}$, $\mathcal{F}_{\mathcal{AUTH}}$, $\mathcal{F}_{\mathcal{NYM}}$, $\mathcal{F}_{\mathcal{RO}}$) -hybrid model if a) the non-interactive zero-knowledge proofs of knowledge scheme is weakly simulation extractable, b) the PS signature scheme is unforgeable, and c) the commitment and encryption schemes are hiding and binding.*

Proof. We show through a series of hybrid games that the environment $\mathcal{Z}$ cannot distinguish the real world from the ideal world. We denote the probability that $\mathcal{Z}$ distinguishes **Game** i from the real-world protocol as Pr[**Game** i].

Game 0: This is the real-world protocol. Therefore, Pr[**Game** 0] = 0.

Game 1: This game proceeds as **Game** 0, except that **Game** 1 runs the extractors for the non-interactive proofs of knowledge π_s, $\{\pi_k\}_{k\in[n_O]}$ and π_v. Under the weak simulation extractability property of NIZK proof system, we have that $|\Pr[\textbf{Game } 1] - \Pr[\textbf{Game } 0]| \leq \mathrm{Adv}^{\mathrm{ext}}_{\mathcal{A}}$, where $\mathrm{Adv}^{\mathrm{ext}}_{\mathcal{A}}$ represents the adversary's advantage of breaking the weak simulation extractability property. Weak simulation extractability [1] implies that if an adversary generates a new NIZK proof which it has not seen previously then the extractor, an efficient algorithm, can extract the witness. That is, an adversary cannot generate new proof unless he knows a witness.

Game 2: This game proceeds as **Game** 1, except that **Game** 2 outputs failure if two request messages were received with commitments C'_a and C_a, and proofs π'_s and π_s such that $C'_a = C_a$ but after extraction of the witnesses from π'_s and π_s, $(a'_1, \ldots, a'_q) \neq (a_1, \ldots, a_q)$. Under the binding property of the commitment scheme, we have that

$|\Pr[\mathbf{Game}\ 2] - \Pr[\mathbf{Game}\ 1]| \leq \mathrm{Adv}^{\mathrm{bin}}_{\mathcal{A}}$, where $\mathrm{Adv}^{\mathrm{bin}}_{\mathcal{A}}$ represents the adversary's advantage of breaking the binding property.

Game 3: This game proceeds as **Game** 2, except that, after extracting the witness $(a_1, \ldots, a_q, r)$ from the proof π_v, **Game** 3 outputs failure if σ' is not a valid signature. But σ' is always a valid PS signature on $(a_1, \ldots, a_q)$ and thus $|\Pr[\mathbf{Game}\ 3] - \Pr[\mathbf{Game}\ 2]| = 0$.

Game 4: This game proceeds as **Game** 3, except that **Game** 4 outputs failure if the adversary was not issued at least $t_V - \tilde{t}_V$ signatures on the messages $(a_1, \ldots, a_q)$. Under the unforgeability property of PS signatures in the random oracle model, we have that $|\Pr[\mathbf{Game}\ 4] - \Pr[\mathbf{Game}\ 3]| \leq \mathrm{Adv}^{\mathrm{unf}}_{\mathcal{A}} \cdot \frac{(n_V - \tilde{t}_V)!}{(t_V - 1 - \tilde{t}_V)! \cdot (n_V + 1 - t_V)!}$, where $\mathrm{Adv}^{\mathrm{unf}}_{\mathcal{A}}$ represents the adversary's advantage of forging the PS signatures in the RO model (see [31, Th. 4]).

Game 5: This game proceeds as **Game** 4, except that in **Game** 5 the ZKPoKs π_s, $\{\pi_k\}_{k \in [n_O]}$ and π_v that are sent to the adversary are replaced by simulated proofs. Under the zero-knowledge property, we have that $|\Pr[\mathbf{Game}\ 5] - \Pr[\mathbf{Game}\ 4]| \leq \mathrm{Adv}^{\mathrm{zk}}_{\mathcal{A}}$, where $\mathrm{Adv}^{\mathrm{zk}}_{\mathcal{A}}$ represents the adversary's advantage of breaking the zero-knowledge proofs.

Game 6: This game proceeds as **Game** 5, except that in **Game** 6 the values $\{X_j\}_{j \in [q]}$ and $\{U_k\}_{k \in [no]}$ in each request are replaced by random elements in $\mathbb{G}$ and $\tilde{\mathbb{G}} \times \tilde{\mathbb{G}}$. At this point, the proofs π_s and π_k's are simulated proofs of false statements. Since the values $\{X_j\}_{j \in [q]}$ and $\{U_k\}_{k \in [no]}$ are randomly distributed, this change does not alter the view of the environment. Thus we have $|\Pr[\mathbf{Game}\ 6] - \Pr[\mathbf{Game}\ 5]| = 0$.

Game 7: This game proceeds as **Game** 6, except that in **Game** 7 the commitments to the attribute $a = (a_1, \ldots, a_q)$ are replaced by commitments to random messages and the corresponding ElGamal encryption of opening shares are generated. Since the Pedersen commitment scheme and ElGamal encryption scheme are information-theoretically hiding, we have that $|\Pr[\mathbf{Game}\ 7] - \Pr[\mathbf{Game}\ 6]| = 0$.

Game 8: This game proceeds as **Game** 7, except that in **Game** 8, the values σ', κ, and ν are computed as follows. Pick random $t_1 \overset{\$}{\leftarrow} \mathbb{Z}_p$, $t_2 \overset{\$}{\leftarrow} \mathbb{Z}_p$ and $t_3 \overset{\$}{\leftarrow} \mathbb{Z}_p$. Set $\sigma' \leftarrow (t_1 G, (t_1 t_2) G)$, $\kappa \leftarrow (t_2 + t_3) G_2$ and $\nu \leftarrow (t_1 t_3) G$. These values follow the same distribution as the ones computed by the honest user in the real-world protocol. Therefore, we have that $|\Pr[\mathbf{Game}\ 8] - \Pr[\mathbf{Game}\ 7]| = 0$.

Game 9: This game proceeds as **Game** 8, except that **Game** 9 outputs failure if the openers either do not output a *reqid* or output more than one *reqid*.

Case 1: If the openers do not open the credential to any *reqid*, it implies the openers do not have the corresponding shares to open the credential. As every genuine credential request is issued by honest validators and the corresponding opening information is stored by the openers, the user must have used a credential that was not issued by honest validators.

Case 2: If the openers open a credential to multiple *reqid*'s, it implies the opening shares satisfy attributes of multiple sessions. As every credential is identified by a unique set of attributes, the adversary to succeed should generate a new signature (m', σ'), given the message-signature pair (m, σ').

Therefore, in both cases, the distinguishable probability of **Game** 9 from **Game** 8 is the advantage of the adversary winning a forgery game. Under the unforgeability property of PS signatures in the random oracle model, we have that $|\Pr[\mathbf{Game}\ 9] - \Pr[\mathbf{Game}\ 8]| \leq \mathrm{Adv}^{\mathrm{unf}}_{\mathcal{A}} \cdot \frac{(n_V - \tilde{t}_V)!}{(t_V - 1 - \tilde{t}_V)! \cdot (n_V + 1 - t_V)!}$, where $\mathrm{Adv}^{\mathrm{unf}}_{\mathcal{A}}$ represents the adversary's advantage of forging the PS signatures in the RO model. $\square$

TABLE 1: Software specifications of PoC

Particulars	Specification	Version
Blockchain	Ethereum	1.0
Ethereum emulator	Ganache	2.5.4
Smart contracts	Solidity	0.5.16
Cryptographic library	py_ecc	5.2.0

VII. IMPLEMENTATION

DTRAC uses the Ethereum blockchain for a tamper-resistant ledger. Table 1 shows the details of the experimental setup of our PoC implementation using Ganache, a private blockchain node for Ethereum application development. The smart contract library is implemented using Solidity v0.5.16. The client uses *web3.py* as the Python interface to interact with the blockchain. The underlying cryptographic elliptic curve library used is *py_ecc* due to its support of alt_bn128, a type III pairing curve available on Ethereum through a precompiled contract.

Smart Contract deployment. The smart contracts in Ethereum have a limitation that their maximum size has to be 24KB, beyond which it runs out of gas. DTRAC, therefore, uses five smart contracts – *Params*, *Request*, *Issue*, *Verify*, and *Opening* – to support all the required functionalities. The smart contracts *Params*, *Request*, and *Issue* are deployed by the validators, *Verify* is deployed by the $\mathcal{SP}$, and *Opening* is deployed by the openers. The *Params* smart contract stores the public keys of the validators, certifiers, and openers, and can be accessed by other smart contracts. A credential request is made by the user through the *Request* smart contract using the *RequestCred* method that verifies the credential request. As the credential request is verified using a smart contract method and not done off-chain, only genuine credential requests are sent to the validators and the *BlindSign* method need not verify the ZKPoK of the credential request. The validators send the blind signature to the user through the *Issue* smart contract using the *SendBlindSign* method. The *Verify* smart contract allows all the blockchain nodes to verify the service request of the user using the *VerifyCred* method. In the opening phase, each opener sends the opening infor-

TABLE 2: Parameters for loan application use case

Category	Value
Certifier	2
Validators (total, threshold)	(4,3)
Openers (total, threshold)	(4,3)
Identity attributes	Name, Age, Address
Income attributes	Income, Role, Company
Credential attributes	Name, Age, Income

TABLE 3: Gas consumptions in smart contract deployment

Smart contract	Gas consumption	Who deploys
Params	943,665	Validator(s)
Request	4,151,748	Validator(s)
Issue	223,994	Validator(s)
Verify	2,400,412	Service Provider
Opening	277,563	Opener(s)

mation to the other openers through the *SendOpeningInfo* method of the *Opening* smart contract.

Optimizations. The precompiled smart contract in EVM [1] that performs pairing checks and elliptic curve operations on alt_bn128, does not support addition and scalar multiplications in $\tilde{\mathbb{G}}$, where $(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T, e)$ is the bilinear group that we use in our scheme. Coconut has implemented a smart contract library for these operations, but it is computationally expensive and impractical on Ethereum due to the block gas limit. In our scheme, a scalar multiplication in $\tilde{\mathbb{G}}$ is replaced with a scalar multiplication in $\mathbb{G}$ and a pairing check, all of which consume less gas. For instance, let $G \in \mathbb{G}$, $\tilde{G} \in \tilde{\mathbb{G}}$ and r be randomly chosen from $\mathbb{Z}_p$. Instead of calculating $r\tilde{G}$ on the smart contract, $R = r\tilde{G}$ and r are given as inputs to the smart contract so that it can verify $e(G, R) = e(rG, \tilde{G})$ to confirm whether $R = r\tilde{G}$ or not. This reduces the gas cost at the expense of increasing the input size.

Mitigating potential attacks. Ethereum transactions are vulnerable to replay attacks and an adversary can access the service by replaying the transaction. To mitigate this we use a timestamp as a public input in the ZKPoK generation as part of the *ProveCred* method. Our scheme is secure against Sybil attacks because the Vcerts are linkable. A man-in-the-middle attack will not succeed because it is almost impossible for an eavesdropper to generate a valid ZKPoK for a fresh randomized signature. DDoS attacks are possible but expensive because of the Ethereum transaction fee. Impersonation attacks are possible if the certifier is malicious but difficult in the multi-certifier model with honest certifiers. We describe these attacks in detail in Appendix C.

PoC implementation. The feasibility of DTRAC is validated by considering the loan use case described in Sec. IV-E. Table 2 depicts the parameters configured for the loan use case. The certifiers, IdP and the employer provide an ID Vcert and an income Vcert, respectively. The user has to obtain the credential on a subset of attributes such as name and DoB from the set of identity attributes and income from the income Vcert. We consider four distributed validators and four distributed openers over the Ethereum blockchain with the threshold parameter set to three for both.

[1]EVM (Ethereum Virtual Machine) is a sandbox attached within each Ethereum full node for executing a smart contract bytecode

VIII. EVALUATION

The experiments were performed on Ubuntu Linux LTS 20.04 desktop computer with an Intel Xeon W-2133 CPU@3.60 GHz processor having 6 cores, 12 threads and 32GB RAM. We use the following two metrics to measure the performance overhead.

1) *Gas* refers to the fee to be paid by a user when executing a transaction on the Ethereum blockchain and it depends on the complexity of the operations involved. Every block in Ethereum has a gas limit (30M as of May 2022) which determines the number of transactions in a block.
2) *Execution time* refers to the time required to run each off-chain method in our system.

We evaluated our system for the loan application use case and bench-marked a) the deployment gas costs for the five smart contracts, b) the average gas costs for the execution of the smart contract methods, and c) the average execution times of the cryptographic primitives of all the four phases.

A. DEPLOYMENT OVERHEAD OF SMART CONTRACTS

Table 3 depicts the deployment gas consumption of *Params*, *Request*, *Issue*, *Verify* and *Opening* smart contract, deployed on a Ganache private node and also gives details of who pays for the deployment gas cost. This deployment is done only once for the loan application use case. Ideally, all the validators should collectively pay the gas fee for deploying the *Params*, *Request* and *Issue* smart contracts and all the openers should collectively pay for deploying the *Opening* smart contract, but the current implementation allows only one of the validators and one of the openers to pay the fee.

B. EXECUTION OVERHEAD OF CRYPTOGRAPHIC PRIMITIVES

Tables 4, 5, 6 and 7 depict the *average execution time* of the cryptographic primitives involved in various phases of the system over 20 iterations. Table 4 depicts the execution times of cryptographic primitives in the registration phase for the ID Vcert and income Vcert requests. One can see that there is a large gap between the execution times of *GenZKPoK* and *VerifyZKPoK* of the ID Vcert and income Vcert. This is due to the additional overhead for the employer in verifying the ID Vcert in zero-knowledge. Tables 5 and 6 measure the excecution times for the issuance and verification phase, respectively. In Table 5, the aggregation time measures the time to aggregate three partial credentials. For the opening phase, Table 7 shows the average execution time of comparing a credential against each record in the opener's registry. The execution time grows linearly with the number of users as the opener has to check the credential against all the $reqid$'s in the system until he finds the correct user.

TABLE 4: Execution time for methods in the Registration phase

	ID Vcert (ms)		Income Vcert (ms)	
Method name	**Mean**	**Std**	**Mean**	**Std**
GenCommitment	65.15	3.21	89.47	0.30
GenZKPoK	45.01	0.85	124.66	6.66
VerifyZKPoK	92.27	0.33	249.33	12.81
SignCommitment	22.01	1.68	21.26	2.04
VerifyVcerts	44.97	0.27	45.01	0.25

TABLE 5: Execution time for methods in the Issuance phase

Method name	Mean (ms)	Std (ms)	Location
PrepareCredRequest	8,911.58	58.05	Local
RequestCred	35,275.06	518.63	On-chain
BlindSign	159.11	1.34	Local
Unblind	22.05	4.79	Local
AggCred	23.40	0.25	Local

TABLE 6: Execution time for methods in the Verification phase

Method name	Mean (ms)	Std (ms)	Location
ProveCred	1,219.55	26.21	Local
VerifyCred	5,286.08	55.48	On-chain

TABLE 7: Execution time for methods in the Opening phase

Method name	Mean (ms)	Std (ms)	Location
PreOpening	3,048.28	35.67	Local
OpenCred	6,380.44	28.57	Local

Table 8 depicts the average gas consumption for the execution of smart contract methods over 20 iterations. For measuring the average gas consumption of the *SendOpeningInfo* method we consider the broadcasting of opening information of a single user/record in the opener's registry and this increases linearly with the number of users/records.

We have evaluated how the gas consumption of the *RequestCred* method varies with the number of openers, certifiers and attributes and how that of the *VerifyCred* method varies with the number of disclosed attributes. This evaluation is based on a set of random attributes and we give our observations below.

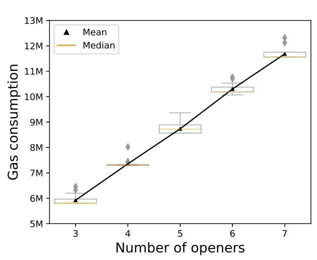

FIGURE 7: Gas consumption of RequestCred vs Number of openers

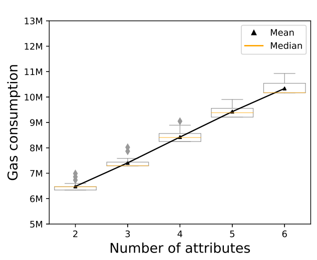

FIGURE 8: Gas consumption of RequestCred vs Number of attributes in the credential request

- *Number of openers vs Gas consumption.* Fig. 7 depicts how the gas consumption of the *RequestCred* method increases linearly with the increase in the number of

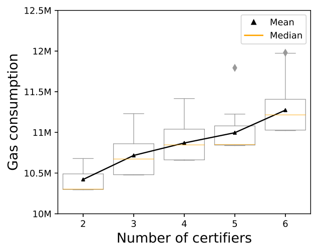

FIGURE 9: Gas consumption of RequestCred vs Number of certifiers

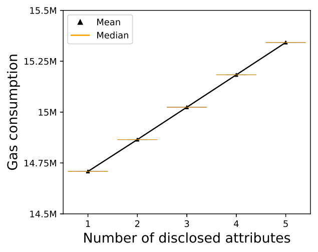

FIGURE 10: Gas consumption of *VerifyCred* vs Number of disclosed attributes

openers. Here we consider 2 certifiers, 4 validators and 4 openers with the threshold value set to 3 for both, 3 attributes in the credential and 3 attributes to be verified by each certifier. We observe that the mean is greater than the median and is close to the minimum gas consumption. i.e., for most of the transactions, the gas costs lie close to the minimum value.
- *Number of attributes vs Gas consumption.* Fig. 8 depicts how the gas consumption of the *RequestCred* method increases linearly with the increase in the number of attributes on which the credential is requested. In this setup, we consider 2 certifiers, 4 validators and 4 openers each with the threshold value set to 3 for both and 3 attributes to be verified by each certifier. We observe that the mean is greater than the median and is close to the minimum gas consumption. i.e., for most transactions, the gas costs lie close to the minimum value.
- *Number of certifiers vs Gas consumption.* Fig. 9 depicts how the gas consumption of the *RequestCred* method increases with the increase in the number of certifiers. In this setup, we consider 4 validators and 4 openers each with the threshold value set to 3 for both, 6 attributes for the credential request and 3 attributes to be verified by each certifier. We observe that the mean is greater than the median and close to the minimum gas consumption inferring that in most of the transactions the gas costs lie close to the minimum value.
- *Number of disclosed attributes vs Gas consumption.* Fig. 10 depicts how the gas consumption of the *VerifyCred* method increases linearly with the increase in the number of disclosed attributes. In this setup, we vary the number of disclosed attributes from 1 to 5 by keeping the number of attributes in a credential fixed, equal to 5. We observe that the standard deviation of the distribution of the gas consumption is very low. i.e., for most of the transactions the gas costs lie close to the mean.

C. COMPARISON WITH MULTIPLE COCONUT INSTANCES

We consider a service request that requires the user to present a credential, with multiple attributes certified by different certifiers, to a service provider. To achieve this functionality using the Coconut scheme, the user has to present multi-

TABLE 8: Gas consumption for execution of Smart contract methods

Method Name	Gas consumption	Smart Contract	Executed by	Frequency
RequestCred	6,089,587	Request	User	once for every credential request
VerifyCred	1,099,204	Verify	User	once for every service request
SendBlindSign	32,445	Issue	Validator	once for every credential request
SendOpeningInfo	41,599	Opening	Opener	once for every opening of a credential

ple credentials, one corresponding to each certifier, to the service provider, which implies running multiple instances of the Coconut scheme. However, in DTRAC, the user has to present only one credential to the $\mathcal{SP}$. Note that within the same instance, the verification process is the same for both DTRAC and the Coconut scheme. We present a detailed evaluation and comparison of the gas costs and execution times of the verification process of running one instance of DTRAC versus running multiple instances of Coconut for a) the loan application use case described in Sec. IV-E, and b) three scenarios we custom build: Scenario 1: two certifiers each attesting three randomly generated attributes, Scenario 2: three certifiers each attesting three random attributes, and Scenario 3: four certifiers each attesting three random attributes. The gas costs are compared when the service request is verified on-chain using smart contracts and the execution times are compared when the SP verifies the service request off-chain. For any service request, credential verification typically involves verifying 1) the signature of the validators on the credential (Eq. 14), and 2) the ZKPoK of the correctness of the undisclosed attributes (Eq. 13).

From Fig. 11, it is clear that DTRAC consumes significantly less gas to verify the service request compared to multiple instances of Coconut on the Ethereum blockchain. To understand why, we need to first understand how the verification of a service request happens over Ethereum. For a type III bilinear pairing group, $(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T, e)$, as explained in Sec. VII, the precompiled contracts in Ethereum do not support operations in $\tilde{\mathbb{G}}$, and we have to replace them with operations in $\mathbb{G}$ followed by a pairing check. This occurs in both the steps – signature verification and ZKPoK verification – of the service request verification. When we run multiple instances of Coconut, the two steps of verification, with the corresponding $\mathbb{G}$ operations and pairing checks, are performed for every instance, whereas in DTRAC, they are done only once. If we increase the number of instances by keeping the total number of attributes constant (which means we have more certifiers attesting the attributes), then the number of pairing operations and, accordingly, the ratio of the gas costs increase linearly.

Fig. 12 depicts that off-chain too the situation is similar – DTRAC takes significantly less execution time than Coconut to verify a service request of a credential that has attributes attested by multiple certifiers. This is again due to the pairing check operation, which is computationally intensive. Since DTRAC always has to verify only one credential, irrespective of the number of certifiers that attest to the attributes, it executes much faster than Coconut, which has to verify multiple credentials. Both the results show that DTRAC is more scalable as opposed to having multiple instances of Coconut and the performance gain increases linearly with the number of instances. Note that during the issuance phase when we have multiple certifiers attesting to multiple sets of attributes, the execution time of DTRAC is more since multiple Vcerts corresponding to each certifier have to be issued. But credential issuance typically happens only once per application while verification of a credential happens more frequently.

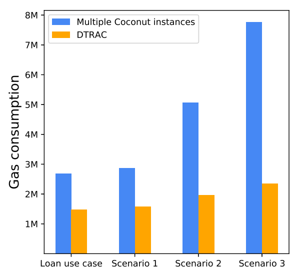

FIGURE 11: Gas consumption of *VerifyCred* Method: DTRAC vs Coconut's multiple instances

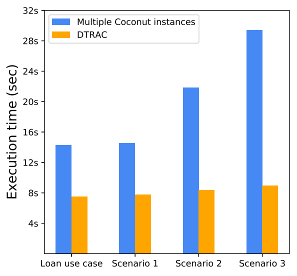

FIGURE 12: Execution time of *VerifyCred* Method: DTRAC vs Coconut's multiple instances

IX. RELATED WORK

'Anonymous credentials' was first coined by D.Chaum [15]. Idemix [4], based on CL signatures [12], is a well-known AC scheme but its size increases linearly with the number of attributes. PS signatures [29] resolves this issue with credentials that are constant and short-sized irrespective of the number of attributes involved. Garman *et al.* [20] addresses the limitation of a single trusted credential issuer by introducing decentralized anonymous credentials (DAC) where the issuance of credentials happens through a distributed ledger. It is computationally expensive due to its large proof size. Crypto-Book [26] provides threshold issuance of credentials but requires verification of t signatures, where t is the threshold parameter, by the service provider. Also, Crypto-Book is limited to identity authentication. The Coconut scheme [34] has a short-sized credential scheme and also supports threshold PS signatures over blockchains. But in the multi-certifier model, Coconut resorts to issuing multiple credentials and as we saw before that makes credential verification expensive. Versatile ABS scheme [3] supports multiple attribute providers, but, just like Coconut, provides multiple credentials on user attributes attested by different providers making credential verification expensive. CanDID

[27] has the same goal as this work but uses a different approach – the user is given a master credential using a decentralized mechanism from legacy credential systems, such as government identity providers, banks, etc. But it does not support threshold issuance.

There are several AC works [8], [11], [27], [36], [38], [39] that focus on credential revocation as opposed to anonymity revocation which is what DTRAC does. For example, CanDID refers to and matches the credential with a public revocation list using secure MPC techniques with fuzzy matching which are computationally intensive. An identity escrow scheme [24] allows the identity to be revealed by a trusted opener, but not all AC schemes support this, and typically the openers in such schemes are not distributed [10], [11]. Versatile ABS scheme [3] supports threshold opening of an anonymous credential, but in this scheme, every service request needs to contain additional opening information which makes credential verification inefficient. DTRAC enables anonymity revocation through a distributed set of openers in which less than a threshold number of openers can be corrupt.

X. CONCLUSION

We propose a decentralized threshold revocable anonymous credential scheme (DTRAC) that enables threshold credential issuance to support a multi-certifier communication model, i.e. having multiple certifiers validate disjoint sets of user attributes. To this threshold issuance scheme we integrate, for the first time, a threshold opening scheme that allows for the opening of a credential with multiple user attributes. We give a formal security analysis of this integration in the UC framework. We also provide a PoC implementation of our scheme over the Ethereum blockchain, making it the first threshold issuance scheme over Ethereum. We also present a detailed evaluation of the performance to show the feasibility of our scheme. We show that DTRAC significantly reduces the credential verification cost for the service provider and is more scalable than the state-of-the-art threshold issuance scheme, Coconut.

REFERENCES

[1] Shahla Atapoor and Karim Baghery. Simulation extractability in Groth's ZKSNARK. In Data Privacy Management, Cryptocurrencies and Blockchain Technology, pages 336–354. Springer, 2019.

[2] Foteini Baldimtsi and Anna Lysyanskaya. Anonymous credentials light. In Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, pages 1087–1098, 2013.

[3] Osman Biçer and Alptekin Kupcu. Versatile abs: Usage limited, revocable, threshold traceable, authority hiding, decentralized attribute based signatures. Cryptology ePrint Archive, 2019.

[4] Patrik Bichsel, Carl Binding, Jan Camenisch, Thomas Groß, Tom Heydt-Benjamin, Dieter Sommer, and Greg Zaverucha. Cryptographic protocols of the identity mixer library. Tech. Rep. RZ 3730, Tech. Rep., 2009.

[5] Dan Boneh and Matt Franklin. Identity-based encryption from the Weil pairing. In Annual International Cryptology Conference, pages 213–229. Springer, 2001.

[6] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In International Conference on the Theory and Application of Cryptology and Information security, pages 514–532. Springer, 2001.

[7] Stefan Brands. Rethinking public key infrastructures and digital certificates: building in privacy. Mit Press, 2000.

[8] Ernie Brickell and Jiangtao Li. Enhanced privacy id: A direct anonymous attestation scheme with enhanced revocation capabilities. In Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society, pages 21–30, 2007.

[9] Jan Camenisch, Manu Drijvers, Anja Lehmann, Gregory Neven, and Patrick Towa. Short threshold dynamic group signatures. In International Conference on Security and Cryptography for Networks, pages 401–423. Springer, 2020.

[10] Jan Camenisch and Anna Lysyanskaya. An identity escrow scheme with appointed verifiers. In Annual International Cryptology Conference, pages 388–407. Springer, 2001.

[11] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In Annual International Cryptology Conference, pages 61–76. Springer, 2002.

[12] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Annual International Cryptology Conference, pages 56–72. Springer, 2004.

[13] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In Annual International Cryptology Conference, pages 410–424. Springer, 1997.

[14] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In Proceedings 42nd IEEE Symposium on Foundations of Computer Science, pages 136–145. IEEE, 2001.

[15] David Chaum. Security without identification: Transaction systems to make big brother obsolete. Communications of the ACM, 28(10):1030–1044, 1985.

[16] Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. In Conference on the Theory and Application of Cryptographic Techniques, pages 186–194. Springer, 1986.

[17] David Gabay, Kemal Akkaya, and Mumin Cebe. Privacy-preserving authentication scheme for connected electric vehicles using blockchain and zero knowledge proofs. IEEE Transactions on Vehicular Technology, 69(6):5760–5772, 2020.

[18] Steven D Galbraith, Kenneth G Paterson, and Nigel P Smart. Pairings for cryptographers. Discrete Applied Mathematics, 156(16):3113–3121, 2008.

[19] Juan A Garay, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. Adaptively secure broadcast, revisited. In Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing, pages 179–186, 2011.

[20] Christina Garman, Matthew Green, and Ian Miers. Decentralized anonymous credentials. In Network and Distributed System Security Symposium (NDSS), 2014.

[21] Oded Goldreich. Foundations of cryptography: Volume 1, Basic Tools. Cambridge University Press, 2007.

[22] Antoine Joux. A one round protocol for tripartite Diffie–Hellman. In International Algorithmic Number Theory Symposium, pages 385–393. Springer, 2000.

[23] Aniket Kate, Yizhou Huang, and Ian Goldberg. Distributed key generation in the wild. IACR Cryptol. ePrint Arch., 2012:377, 2012.

[24] Joe Kilian and Erez Petrank. Identity escrow. In Annual International Cryptology Conference, pages 169–185. Springer, 1998.

[25] Anna Lysyanskaya, Ronald L Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In International Workshop on Selected Areas in Cryptography, pages 184–199. Springer, 1999.

[26] John Maheswaran, Daniel Jackowitz, Ennan Zhai, David Isaac Wolinsky, and Bryan Ford. Building privacy-preserving cryptographic credentials from federated online identities. In Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, pages 3–13, 2016.

[27] Deepak Maram, Harjasleen Malvai, Fan Zhang, Nerla Jean-Louis, Alexander Frolov, Tyler Kell, Tyrone Lobban, Christine Moy, Ari Juels, and Andrew Miller. Candid: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability. In 2021 IEEE Symposium on Security and Privacy (SP), pages 1348–1366. IEEE, 2021.

[28] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Annual International Cryptology Conference, pages 129–140. Springer, 1991.

[29] David Pointcheval and Olivier Sanders. Short randomizable signatures. In Cryptographers' Track at the RSA Conference, pages 111–126. Springer, 2016.
[30] David Pointcheval and Olivier Sanders. Reassessing security of randomizable signatures. In Cryptographers' Track at the RSA Conference, pages 319–338. Springer, 2018.
[31] Alfredo Rial and Ania M Piotrowska. Security analysis of coconut, an attribute-based credential scheme with threshold issuance. Cryptology ePrint Archive, 2022.
[32] Claus-Peter Schnorr. Efficient signature generation by smart cards. Journal of Cryptology, 4(3):161–174, 1991.
[33] Adi Shamir. How to share a secret. Communications of the ACM, 22(11):612–613, 1979.
[34] Alberto Sonnino, Mustafa Al-Bassam, Shehar Bano, Sarah Meiklejohn, and George Danezis. Coconut: threshold issuance selective disclosure credentials with applications to distributed ledgers. 26th Annual Network and Distributed System Security Symposium (NDSS), 2019.
[35] Markus Stadler. Publicly verifiable secret sharing. In International Conference on the Theory and Applications of Cryptographic Techniques, pages 190–199. Springer, 1996.
[36] Patrick P Tsang, Man Ho Au, Apu Kapadia, and Sean W Smith. Blacklistable anonymous credentials: blocking misbehaving users without ttps. In Proceedings of the 14th ACM Conference on Computer and Communications Security, pages 72–81, 2007.
[37] Paul Voigt and Axel Von dem Bussche. The EU General Data Protection Regulation (GDPR). A Practical Guide, 1st Ed., Cham: Springer International Publishing, 10(3152676):10–5555, 2017.
[38] Rupeng Yang, Man Ho Au, Qiuliang Xu, and Zuoxia Yu. Decentralized blacklistable anonymous credentials with reputation. Computers & Security, 85:353–371, 2019.
[39] Yanjiang Yang, Haibin Cai, Zhuo Wei, Haibing Lu, and Kim-Kwang Raymond Choo. Towards lightweight anonymous entity authentication for IoT applications. In Australasian Conference On Information Security and Privacy, pages 265–280. Springer, 2016.

APPENDIX A BACKGROUND

Here we review the necessary background information for the protocols described in Sec II-D and II-E.

A. BILINEAR GROUPS

The advent of pairings-based cryptography (PBC) [5], [22] led to short, efficient signatures with many versatile features. The randomizable signatures mentioned in Sec. II-D – PS signatures – rely on bilinear pairings, and we give an overview of the concept here.

Definition A.1 (Bilinear group)**.** *Let $\mathbb{G}$, $\tilde{\mathbb{G}}$ and $\mathbb{G}_T$ be cyclic groups of prime order p. A bilinear map e is a function $e : \mathbb{G} \times \tilde{\mathbb{G}} \to \mathbb{G}_T$, satisfying the following properties.*

1) *Bilinearity. For all $P \in \mathbb{G}$, $Q \in \tilde{\mathbb{G}}$ and $a, b \in \mathbb{Z}_p$, $e(aP, bQ) = e(P, Q)^{ab}$.*
2) *Non-degeneracy. For all $P \in \mathbb{G}$, $Q \in \tilde{\mathbb{G}}$ such that $P \neq 1_{\mathbb{G}}$, $Q \neq 1_{\tilde{\mathbb{G}}}$ and $e(P, Q) \neq 1_{\mathbb{G}_T}$.*
3) *Computability. The map e is efficiently computable.*

A bilinear group is the set comprising of $\mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T$ along with e that satisfies the above properties.

There are three types of pairings, and they differ in the structure of the underlying groups [18]. They are namely, type I where $\mathbb{G} = \tilde{\mathbb{G}}$; type II where $\mathbb{G} \neq \tilde{\mathbb{G}}$ but there exists an efficiently computable homomorphism, ϕ between $\mathbb{G}$ and $\tilde{\mathbb{G}}$; and type III where $\mathbb{G} \neq \tilde{\mathbb{G}}$ and there is no efficiently computable homomorphism between $\mathbb{G}$ and $\tilde{\mathbb{G}}$ in either direction. Type I pairings were used in the older PBC schemes and have now been mostly replaced by type III pairings because they are more efficient and are compatible with several computational hardness assumptions such as the decisional Diffie-Hellman assumption in $\mathbb{G}$ or $\tilde{\mathbb{G}}$ called the XDH assumption [6].

There are two versions of PS signatures. The security of the earlier version relies on the PS assumption to prove the EUF-CMA security of the scheme, which is an interactive assumption [29]. Due to rising concerns about interactive assumptions in the cryptographic community, the authors reassessed the security assumptions and slightly modified the signature scheme so that the security relies on q-type (q-MSDH) assumptions [30]. The earlier version of PS signatures is EUF-wCMA under the q-MSDH assumption, but the later version is EUF-CMA secure under the q-MSDH assumption. The Coconut threshold issuance scheme uses the earlier version of PS signatures.

B. SHAMIR SECRET SHARING

A secret sharing scheme, introduced by Shamir [33], is used to secure a secret in a distributed manner. In (t, n)-Shamir secret sharing, the original secret is split into n shares, out of which t or more shares are needed to reconstruct the secret. It is based on polynomial interpolation over finite fields. It satisfies the following two properties.

1) *Recoverability.* The secret can be reconstructed efficiently with the knowledge of at least t shares.
2) *Secrecy.* The secret is completely undetermined without the knowledge of at least t shares.

To generate the Shamir shares of a secret $m \in \mathbb{Z}_p$, we first generate a polynomial $\mathcal{P}$ of degree $t-1$ whose coefficients are chosen randomly from $\mathbb{Z}_p$ (see Eq. 3). The n shares, called Shamir shares, are generated by evaluating the polynomial at values 1 to n as $s_i = \mathcal{P}(i), i \in [n]$. Later any t of those shares can be used to reconstruct the original secret m using Lagrange coefficients. Let T be the set of indices of the shares which are used in the reconstruction protocol. Then the original secret can be reconstructed using $\Sigma_{j \in T} w_j s_j$, where w_j is the j-th Lagrange coefficient (see Eq. 12).

The Shamir secret sharing scheme is used twice in our proposed scheme: 1) the key generation phase of the threshold PS signature scheme (Sec. II-D) and 2) the generation of opening information (Sec. V-D).

APPENDIX B IDEAL FUNCTIONALITIES

In this section we present a brief overview of the ideal functionalities that are required to analyze the security of our scheme.

1) ***Ideal Functionality,*** $\mathcal{F}_{\mathcal{KG}}$: The ideal functionality $\mathcal{F}_{\mathcal{KG}}$ supports secure key generation for the validators $\mathcal{V} = \{\mathcal{V}_1, \ldots, \mathcal{V}_{n_V}\}$ and public key registration for the openers $\mathcal{O} = \{\mathcal{O}_1, \ldots, \mathcal{O}_{n_O}\}$. $\mathcal{F}_{\mathcal{KG}}$ assumes the security parameter k.

   Key generation of validators : When a validator $\mathcal{V}_i$ requests for keys, $\mathcal{F}_{\mathcal{KG}}$ generates a signing key pair $\{(sk_i, pk_i)\}_{i \in [n_V]}$ for the validators and a public key

pk for credential verification to support threshold issuance. $\mathcal{F_{KG}}$ sends $(sid, \{pk_i\}_{i\in[n_V]}, pk)$ to the simulator $\mathcal{S}$ to simulate the communication during key generation. When $\mathcal{S}$ prompts $\mathcal{F_{KG}}$, it outputs $(sid, \{(sk_i, pk_i)\}_{i\in[n_V]}, pk)$ to the validator $\mathcal{V}_i$.

***Registration of Openers keys*:** When an opener, $\mathcal{O}_k$ registers its public key opk_K, $\mathcal{F_{KG}}$ sends $(sid, \mathcal{O}_k, opk_k)$ to the simulator $\mathcal{S}$ to simulate the communication during key registration. When $\mathcal{S}$ prompts $\mathcal{F_{KG}}$, it sends the notification of successful registration to $\mathcal{O}_k$.

***Retrieval of public keys*:** When a party P queries for the public keys, $\mathcal{F_{KG}}$ sends $(sid, \{(sk_i, pk_i)\}_{i\in[n_V]}, pk, \{opk_k\}_{k\in[n_O]})$ to the simulator $\mathcal{S}$ and the party P.

The $\mathcal{F_{KG}}$ described above is different from $\mathcal{F_{KG}}$ in [31] in one way – here, along with the key generation of the validators, the openers register their public keys as well.

2) ***Ideal Functionality,*** $\mathcal{F_{BC}}$: The ideal functionality $\mathcal{F_{BC}}$ supports an ideal authenticated broadcast message transmission. When a sender T sends $(sid, m, \mathcal{R})$, where m is the message and $\mathcal{R}$ is a set of receivers, to $\mathcal{F_{BC}}$, it sends $(sid, m, T, \mathcal{R})$ to the simulator $\mathcal{S}$ and all the entities in the set $\mathcal{R}$. $\mathcal{F_{BC}}$ is different from the $\mathcal{F_{BC}}$ in [19] in the following way: here, along with the message m, the information regarding who are the receivers, i.e. elements of $\mathcal{R}$, are also sent to all the receivers.

3) ***Ideal Functionality,*** $\mathcal{F_{NYM}}$: The ideal functionality $\mathcal{F_{NYM}}$ [31] supports a secure idealized pseudonymous communication.

***Sending a message*:** When a sender T sends (sid, m, P, R), where m is the message, P is a pseudonym, and R is a receiver to $\mathcal{F_{NYM}}$, it leaks the message length to $\mathcal{S}$ to capture the information that is leaked during the communication between T and R, and sends (sid, m, P) to R. $\mathcal{F_{NYM}}$ hides the sender T from the receiver R.

***Replying with a message*:** When R replies with (sid, m, R, P) to $\mathcal{F_{NYM}}$, it leaks the message length to the simulator $\mathcal{S}$ for the same reason and sends (sid, m, P) to T.

4) ***Ideal Functionality,*** $\mathcal{F_{AUTH}}$: The ideal functionality $\mathcal{F_{AUTH}}$ [14] supports an ideal authenticated message transmission. When a sender T sends (m, R), where m is the message and R is a receiver, to $\mathcal{F_{AUTH}}$, it sends (m, T) to the simulator $\mathcal{S}$ and the receiver R.

5) ***Ideal Functionality,*** $\mathcal{F_{RO}}$: The ideal functionality $\mathcal{F_{RO}}$ [31] supports an ideal hash function. When a party P sends a message m, if (sid, m, H) is not stored, i.e. $\mathcal{F_{RO}}$ has not seen that message previously, it generates a fresh H and stores (sid, m, H). $\mathcal{F_{RO}}$ then outputs H to P.

APPENDIX C ATTACK MODEL

In order to demonstrate the security of the proposed scheme, we determine the capabilities and possible actions of an adversary. We consider the following attacks that lead to privacy leakage, credential misuse, service unavailability, etc.

1) *Impersonation Attack.* A certifier receives all the user attributes in the registration phase. A malicious certifier can make a new Vcert using these attributes and impersonate a user, and this cannot be prevented in our scheme. But if more than one certifier is required to issue an anonymous credential, this attack is unlikely as the malicious certifier will have to obtain valid Vcerts from all the other certifiers to be able to impersonate the user. Thus impersonation attacks can be mitigated in the proposed multi-certifier model.
2) *Man in the Middle Attack.* An adversary can eavesdrop on service requests over a blockchain which means the adversary has access to the randomized credential and the ZKPoK of the user's private data. Although an adversary can succeed in creating a new randomized signature from this information, he cannot generate a new ZKPoK corresponding to the fresh signature.
3) *Sybil Attack.* A user can request multiple credentials in an attempt to construct fake credentials. But as the certifier verifies each user before providing a Vcert and the Vcerts are linkable, such fake credential requests will be detected by the scheme.
4) *Denial of Service Attack.* Our scheme allows a user to request service as many times as she wants and this can be used by an adversary to cause a system shutdown. But Ethereum's transaction fees ensure that such an attack will be expensive.
5) *Replay Attack.* Since the requests to the $\mathcal{SP}$ are through the blockchain, the transaction data can be replayed and an adversary can gain access to the service. This attack can be prevented by adding a timestamp to the ZKPoK associated with the service request. This timestamp expires after a certain window and this window is determined by the $\mathcal{SP}$ [17]. The smart contract performs the proof execution only if the request is within this time period.

APPENDIX D ASYMPTOTIC ANALYSIS OF PHASES

We provide an asymptotic analysis of our scheme, with respect to the number of attributes, number of validators and number of openers. Table 9 depicts an analysis of the registration phase as a function of the number of attributes verified by each certifier, given by $attr$. Note that the analysis of the *GenZKPoK* and *VerifyZKPoK* methods are done assuming that the certifiers are independent. If a certifier requires that the ZKPoK of attributes of other Vcerts have to be verified then the complexity increases linearly with the number of attributes in the corresponding Vcerts.

Table 10 depicts the complexity analysis of the other phases as a function of the number of openers n_O, the number of validators n_V and the total number of attributes verified by all the certifiers, say $total_{attr}$. The complexity of the *PreOpening* and *OpenCred* methods are for a single credential and it increases linearly with the number of users.

TABLE 9: Computational complexity in the Registration phase

Method name	Complexity
GenCommitment	$\mathcal{O}(attr)M_1 + \mathcal{O}(attr)A_1$
GenZKPoK	$\mathcal{O}(1)M_1 + \mathcal{O}(1)A_1$
VerifyZKPoK	$\mathcal{O}(attr)M_1 + \mathcal{O}(attr)A_1$
SignCommitment	$\mathcal{O}(1)M_1 + \mathcal{O}(1)A_1$
VerifyVcerts	$\mathcal{O}(1)M_1 + \mathcal{O}(1)A_1$

A_1 and M_1 represents the addition and scalar multiplication operations in $\mathbb{G}_1$ respectively.

TABLE 10: Computational complexity for the Issuance, Verification and Opening phases

Method name	Average execution time
PrepareCredRequest	$\mathcal{O}(n_O total_{attr})M_1 + \mathcal{O}(total_{attr})A_1 + \mathcal{O}(n_O total_{attr})M_2 + \mathcal{O}(n_O total_{attr})A_2$
RequestCred	$\mathcal{O}(n_O total_{attr})M_1 + \mathcal{O}(total_{attr})A_1 + \mathcal{O}(n_O total_{attr})\mathcal{BP}$
BlindSign	$\mathcal{O}(total_{attr})M_1 + \mathcal{O}(total_{attr})A_1$
Unblind	$\mathcal{O}(1)M_1 + \mathcal{O}(1)A_1$
AggCred	$\mathcal{O}(n_V)M_1 + \mathcal{O}(n_V)A_1$
ProveCred	$\mathcal{O}(1)M_1 + \mathcal{O}(total_{attr})M_2 + \mathcal{O}(total_{attr})A_2$
VerifyCred	$\mathcal{O}(total_{attr})M_1 + \mathcal{O}(1)A_1 + \mathcal{O}(1)A_2 + \mathcal{O}(total_{attr})\mathcal{BP}$
PreOpening	$\mathcal{O}(n_O)\mathcal{BP}$
OpenCred	$\mathcal{O}(n_O)\mathcal{E}$

A_1, A_2 represent addition operations in $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively and M_1, M_2 represent the scalar multiplications in $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. $\mathcal{BP}$ represents the pairing operation and $\mathcal{E}$ represents the exponentiation operation in $\mathbb{G}_T$.

• • •

ADEEBA NAAZ completed her B.Tech in Computer Science and Engineering in 2017 from Ambedkar Institute of Technology and M.Tech in Computer Science and Engineering from the Indian Institute of Technology Hyderabad in 2022. Her research areas include Software-Defined Networking, Blockchains, Applied cryptography, and ZKP techniques.

T. V. PAVAN KUMAR B received his B.Tech degree in Computer Science and Engineering from the National Institute of Technology Warangal in 2019. He also earned an M.Tech degree in Computer Science and Engineering from the Indian Institute of Technology Hyderabad in 2022. His research interest includes Software-Defined Networking, Reinforcement Learning, and Privacy-preserving mechanisms over Blockchains.

MARIA FRANCIS completed her B.Tech in Computer Science and Engineering in 2007 from the National Institute of Technology Calicut and her Ph.D. in Computer Science from the Indian Institute of Science Bangalore in 2016. Currently, she is an Assistant Professor at the Indian Institute of Technology Hyderabad. Her research interests include Computer Algebra, Symbolic Computation, Pairings based cryptography, Lattice-based cryptography, and Privacy mechanisms over blockchains.

KOTARO KATAOKA received his B.A. in Environmental Information in 2002 from Keio University, and Master's and Ph.D. in Media and Governance from Keio University in 2004 and 2010 respectively. Currently he is Associate Professor at Department of Computer Science and Engineering, Indian Institute of Technology Hyderabad. His research interest includes Internet Architecture, Software-Defined Networking, Network Functions Virtualization, Blockchain, and Online Japanese Education.

• • •