CredAct: Privacy-Preserving Activity Verification for Benefits Schemes in Self-Sovereign Identity

***Abstract*—We propose *CredAct*, a user activity verification designed with data minimisation to protect privacy. Many Benefits Schemes, such as discount offers, loyalty programs, and incentive systems, require verification of user activity (e.g., buying healthy food, step counts) in their business processes. These service providers can collect a large amount of users' personal information, and often users do not have fine-grained control over the scope of data disclosure. In *CredAct*, we propose a Self-Sovereign Identity based framework implemented on blockchain that enables users participating in a benefits scheme to minimise data sharing during the submission and verification of data. We use a smart contract-based function along with a Zero-Knowledge Proof cryptographic commitment scheme, that forces the entities involved in the business process to collect or disclose only the required (minimum) data to fulfill the intended purpose. The evaluation shows that the system is feasible with minimal operational overheads compared to traditional cryptographic techniques. We also perform a qualitative privacy and security analysis considering relevant threats to *CredAct*.**

***Index Terms*—Blockchain, self-sovereign identity, incentives, responsible behaviour, activity verification, minimum disclosure, user privacy.**

1. Introduction

An individual engages in online activities tied to their identity. We use the term *"Identity activity"* to refer to recorded transactions associated with an entity's identity [1]. Some examples of identity activity include using services like a shopping website, online banking, or booking a table at a restaurant through an online portal.

While most of these identity activities are intended to be kept private, accessing certain services may require users to share their identity activities and their true identity for verification. For instance, a student may allow her university to share the list of enrolled courses and their credits with a public transport authority, who will then verify the student's full-time status to grant access to a student discount program. A customer of a health insurance provider may agree to share their grocery shopping history with the program provider, who will then verify the data according to their point-awarding criteria (e.g., purchasing healthy foods) [2]. We use "Benefits Schemes" as a general term that refers to incentives, rewards, or loyalty schemes designed to enhance overall engagement with the brand or service.

In current systems [3]–[5], the users give consent to the benefits scheme providers (e.g., health insurance program) to receive activity data directly from the data providers (e.g., supermarkets). The direct data sharing of these identity data satisfies the service provider's need to verify the data. However, the sharing is often done in an "all-or-nothing" manner, and users cannot customise the scope of information shared. This practice removes the users from exercising any data minimisation rights where data sharing is implemented by collecting only directly relevant and necessary data to accomplish a specified purpose.

Self-Sovereign Identity (SSI) platforms, a popular decentralised digital identity management scheme, offer an alternative framework that can support the conflicting needs of both parties: one seeks to access data for verification purposes, while the other aims to control sharing to minimise the disclosure of private information. Blockchain is the most common choice to implement SSI, which acts as a *neutral third-party*. Because of the trust properties in the architecture, such as traceability, verifiability, and integrity, blockchain-based SSI can provide a foundation for designing a system that supports conflicting needs.

To further articulate our research problem, let us take a motivating scenario. **Alice** is a university student. This semester, she studies full-time, and has enrolled in some STEM (Science, Technology, Engineering, and Mathematics) courses although they are not her major. Now she may be eligible for discount fares offered by public transport and a youth allowance program offered to students studying STEM courses by the state government. **Alice** must share her list of enrolled courses obtained from **her university** with the verifiers, that is, **transport authority** and **youth allowance provider**. The transport authority checks the total credits to determine the study load. The youth allowance provider checks the STEM courses approved for the allowance. Although Alice would like to enjoy the benefits, she is uncomfortable with sharing the full details of her enrolment records with multiple parties and wants to minimise data sharing.

In the context of SSI, these identity activity sharing and verification scenarios would involve three SSI roles, namely a *Holder* who is a user like Alice. An *Issuer* who provides a credential containing verifiable information about the user. In our scenario, the issuer is Alice's university, which provides the identity activity data containing the list of courses Alice has enrolled in. *Verifiers* who reads and verifies the credentials a holder presents for their business

purpose. In our scenario, the transport authority and youth allowance provider are verifiers.

Here, we highlight the data minimisation goal for Alice - how to obtain the student benefits (i.e., passing the verification requirements) without sharing the complete enrolment records. To achieve this goal, most SSI platforms suggest a well-known data minimisation technique called Selective Disclosure (SD) where Alice shares only the required data from the enrolment records [6]–[9]. In SSI, this is done with the help of the issuer, who will implement cryptography-based SD techniques when generating the credentials for the holder. However, we identify the following problems with the current approach: (i) since the issuer is involved in cryptographically signing the credentials containing the records, the user will have to reveal to the issuer which data in the records are to be disclosed. Zero Knowledge Proof (ZKP) techniques, for instance, use commitment schemes that require issuers to "commit" to any to-be-disclosed attributes. This could lead to the issuer identifying the intended use of the requested credential (e.g., Alice revealing only the courses allowed by the youth allowance program or Alice revealing the number of credits attribute). That is, the current approach assumes the issuer is a trusted party and ignores the privacy risk associated with the issuer potentially correlating the disclosed attributes in a credential with the user's intended actions. In our scenarios, Alice's repeated use of the credential to access various government services could contribute additional information for profiling her. (ii) Verifying certain activities often requires historical data logs, like records of courses completed or three-month daily step counts. Current selective disclosure methods for credentials, such as driver's licenses or student cards, are designed for singular records. Applying these methods as they are, such as signing individual attributes, is not well-suited for scenarios that require detailed historical data.

In this paper, we propose an SSI-based framework *CredAct* for identity activity verification in the context of *Benefits Schemes*. *CredAct* has a data minimisation technique that has the following technical design elements:

- **requirement policy based verification:** Verifier declares a "requirement policy" for activity verification, which specifies the required input and output for verification. This policy is encoded as a Requirement Function in a smart contract. Users who wish to have their activity verified will need to provide the required input to the function. The verifier needs to pre-define the least amount of data necessary for the verification process. Meanwhile, the user is responsible for supplying the relevant portion of her activity data as input.
- **ZKP-based cryptographic commitment scheme**: To coordinate verification of the data supplied by users without having to access the data itself, we use a cryptographic commitment scheme. We extend a well-known commitment scheme, Pedersen Commitment, by adding new features - First, it does not require issuers to sign commitments on any to-be-disclosed attributes. Instead, holders obtain activity data from issuers and generate a commitment on the data as a whole. The issuer only checks that the holder's commitment is on the correct activity data owned by the holder, and stores the commitment on the blockchain. Second, holders do not share the activity data with the verifiers. In lieu of the data, the holder only shares two proofs about the data; A proof of her knowledge of the data used to generate a commitment (*proof of record*), and another proof that shows the input supplied to the requirement function is indeed contained within the commitment (*proof of input*). This ensures both the validity of the commitment itself and the accuracy of the requirement function input in relation to the committed data.

These two technical elements are coordinated by a smart contract named **Requirement Verification (RV) smart contract** responsible for taking the user input to the function, executing the function, and verifying the user proofs according to the commitment scheme. This design leads to the following outcomes: (a) the verifiers never access the activity data directly, (b) since the issuers do not sign attributes to be disclosed per request by the users, the risk of issuers relating the credential and its intended use is reduced, (c) the idea of considering a credential as a whole during commitment generation, rather than individual attributes, has an added benefit of the scheme applying to broader credential formats.

CredAct is implemented on Ethereum. The evaluation results show that the scheme is feasible with minimal operational overheads compared to traditional cryptographic techniques. We also perform a qualitative privacy and security analysis by considering relevant threats.

2. Related Works

Traditional identity activity verification mechanisms usually follow a reputation-based [10], [11], feedback-based [12] or game-based mechanism [13], [14]. Usually, these mechanisms need a trusted centralised authority to verify the users' activity. Consequently, the mechanisms fail to provide public auditability and decision fairness [15]. As a response, there are some existing works in identity activity verification that use blockchain. We organise them into three categories: verifiable data-sharing systems, verifiable crowdsensing data and verifiable behavioural data.

Verifiable Data Sharing in P2P Systems. In this work, the focus is on verifying the authenticity or correctness of shared data in peer-to-peer settings. Naz et al. [16] propose a blockchain-based secure data-sharing platform leveraging IPFS. After each successful data sharing, the receiver reviews the data for a reward. Using the Watson Analyser, fake reviews are removed, and only valid reviews are stored on the blockchain. A similar decentralised data-sharing platform is proposed in [17], where the data owner receives manual feedback from its data receivers. Deepchain [18] presents a federated deep-learning framework that proposes an incentive-driven training data-sharing method to encourage trusted contributors. The shared training data is reviewed when used by the other participants.

Verifiable Crowdsensing Data. Mobile crowd sensing(MCS) leverages the widespread availability of mobile devices to gather, analyze, and share data collected by individuals in various locations. To validate the sensing data, Chen et al. [19] propose a blockchain-based solution based on geographical location, sampling time point and similarity match with neighbouring information. In another solution, Wang et al. [20] validate the data by the submission time and data accuracy, which is determined in terms of similarity among multiple submissions. TruCentive [21] is a game theoretic MCS platform where participants earn incentives by selling the sensing data (e.g., parking availability). They can further maximise their incentive if the data buyer can re-sell (e.g., re-selling the parking availability). Such a re-selling is only possible when participants contribute authentic data.

Verifiable Behavioural Data. In many incentive systems, blockchain is utilised as an immutable storage of users' activity data which is then verified by the incentive providers for reward. In Jaffe et al. [22], a cyclist must mount a GPS sensor on the bicycle to prove cycling activity. The sensor collects cycling data and stores the data on-chain. Yichen et al. [23] proposed a blockchain-based system to motivate the vehicles to help the other vehicles who need emergency road rights (e.g., at risk of missing flight schedule). They used an electronic radar module to monitor and record on-chain whether a vehicle has transferred its rights of way to another vehicle. Utomo et al. [24] proposed a blockchain-based system for trash bin providers. This system tracks a trash bin and its users via the presence of a smartwatch (Bluetooth) and measure the amount of trash thrown in the bin. These data are recorded on-chain to reward the bin provider for service. In Chen et al. [25], electric vehicles charging activities are tracked (e.g., arrival and leaving times, charging duration) to verify whether they are following the recommended charging patterns or not.

Overall, these have only focused on how blockchain could be used to track and store user activities in an immutable way for verification. They failed to address the need to maintain user privacy by minimising the activity data disclosure in the verification process.

3. *CredAct* Overview

Self Sovereign Identity Platform. Shown in Fig. 1, *CredAct* uses a prototype implementation of SSI [6] to support SSI related operations. SSI sets up a digital peer-to-peer network among identity issuers, holders and verifiers where blockchain manages verifiable identity registries to act as an intermediary of trust. To represent the SSI users' identity, W3C (World Wide Web Consortium) has standardised Decentralised IDentifier (DID) [26]. Each DID is associated with a DID DOcument (DDO), which contains a public key of the DID user, the communication protocols between two DIDs and service endpoints (e.g., HTTPS URLs) used for interactions. Any DID user can prove DID ownership using the associated private key.

The SSI platform in Fig. 1 is deployed on a permissioned blockchain to manage the identities of the entities involved in the scheme. It includes two layers: *service* and *registry*. The registry layer includes on-chain registry smart contracts. The service layer includes *DID Services* (for managing DIDs), *Issuer Services* (for managing issuer eligibility to sign credentials) and *Credential Services* (for managing credentials, including services to verify them).

Requirement Policies. A verifier provides a requirement policy that explicitly expresses the required holder's data, how the data is verified, and which authorities they trust in vouching for such data. Consider the following example that assesses a student's eligibility for a discounted fare on public transport.

Listing 1: An example of requirement policy

```
01: own es::EnrolmentStatement issued-by UniA
02: own sc::StudentCard issued-by UniA
03: reveal es.hash(Student_Number)
04: where (sc.hash(Student_Number) == es.hash(Student_Number)) as cond1 ∧
05: (es.enrolment_start_date ≥ Subtract(to_date, 2M)) as cond2 ∧
06: es.enrolment_status == 'full-time' as cond3
07: sign 'I agree with the general terms and conditions.'
```

Holders (a) reveal their student number from their enrolment statement (line 3), (b) have the same student number in both documents owned by the holder in line 1/line 2 (i.e., enrolment statement and student card) (line 4), (c) have the enrolment statement generated no more than a 2 months ago (line 5), (d) the enrolment status is 'full-time' (line 6) and (e) consent to the general terms and conditions (line 7).

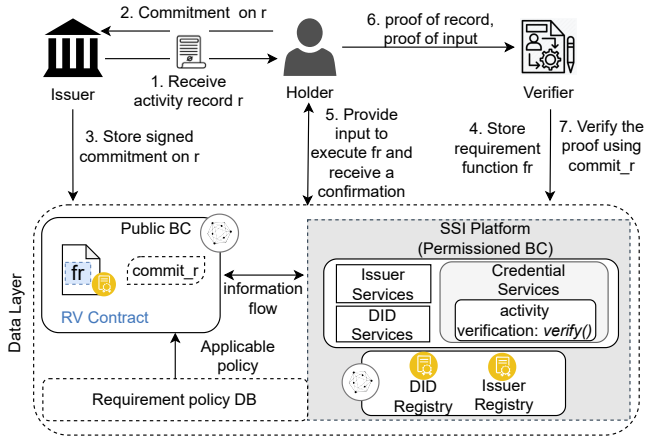

Figure 1: *CredAct* Architecture and Workflows

As shown in Fig.1, the data layer of *CredAct* is comprised of a requirement policy database (DB) provided by verifiers and public blockchain. The verifier develops a *Requirement Function* that codifies the requirement policies and includes it in a smart contract called *Requirement Verification (RV) Contract*.

Overall Workflow. The workflow illustrated in Fig.1 shows an overview of the commitment scheme and smart contract interactions: (Step 1) A Holder (e.g., Alice) receives the signed activity record `r` (e.g., enrolment statement) from the activity record issuer (e.g., university). (Step 2,3) The holder generates a commitment on `r`. The issuer verifies, signs and stores the commitment on-chain ($\texttt{Commit}_{\texttt{r}}$).

(Step 4) Verifier deploys the *RV Contract* with the requirement functions. (Step 5) Holders provide input to execute the functions (Step 6) If the function returns true, the holder generates and sends *proof of record* and *proof of input* to the verifier. (Step 7) Finally, the verifier calls *RV contract* to verify the proofs against $\texttt{Commit}_{\texttt{r}}$ on-chain.

4. ZkP-based Commitment Scheme

Since the holders are not sharing the activity records with the verifiers, *CredAct* employs a commitment scheme to allow the verifiers to verify the required input supplied by holders without having to access the records themselves. This section describes the cryptographic building blocks for commitment generation and verification in *CredAct*.

4.1. Pedersen Commitment Scheme

The Pedersen commitment [27] is an equivocable cryptographic commitment in which a holder first transmits an encrypted message (dummy commitment) to a verifier who does not possess the decryption key yet (referred to as Commit Phase). The key is transmitted later when the verifier needs to "equivocate" the message to reveal the actual message (referred to as Open Phase). We chose Pedersen because in *CredAct*, the commitment is stored in a public blockchain, and there is a risk of the commitment being reused by an adversary if a key can be guessed. In Pedersen, the commitment generation and verification are sufficiently difficult that an adversary cannot find the encryption key to recreate the commitment [27].

Thinking through our motivating scenario, Alice holds activity records $\texttt{r}$ to commit. If Pedersen is applied as-is, to equivocate the dummy commitment sent by Alice during the Commit Phase, Transport Authority requires $\texttt{r}$ (i.e., the activity record Alice used to generate the commitment) during the Open Phase. To allow Alice to complete the verification steps without sharing $\texttt{r}$ with the verifier, we propose a modified version of Pedersen commitment in which instead of sharing $\texttt{r}$, Alice shares **two proofs**, (i) *Proof of record* - which is a ZKP of Alice's knowledge about the committed $\texttt{r}$, (ii) *Proof of input* - which is auxiliary information Alice supplies to prove that the input to the requirement function is in fact contained within $\texttt{r}$. The verifier uses these proofs to equivocate the dummy commitment.

4.2. Proof of Record: ZKP of Activity Record

For the first proof, *proof of record*, we take the Non-Interactive Zero Knowledge Proof (NIZKP) scheme [27] as it allows two parties (e.g., holder and verifier) to verify a shared secret of their peer without needing a real-time interaction. In *CredAct*, Alice generates NIZKP on the $\texttt{r}$ to prove that she indeed has knowledge about the activity records as a way of showing the ownership of the record. To generate and check the correctness of NIZKP, we used the algorithms introduced in [27]. For space reasons, we omit the details and refer the readers to [27].

4.3. Proof of Input - Hash Generation

For the second proof, *proof of input*, we consider the following: ① Alice generates a hash of $\texttt{r}$, ② Alice generates *proof of input*, according to the hashing scheme used to generate the hash of $\texttt{r}$, ③ Later, Transport Authority generate the hash of $\texttt{r}$ themselves using the *proof of input*, ④ Transport Authority re-compute the commitment using the hash of $\texttt{r}$ and match it with the commitment on-chain.

The concept of "proof of input" is based on the verifier having the input provided by the holder for the requirement function. To demonstrate that this input is included in the activity records $\texttt{r}$, the holder must supply additional information. The verifier then uses this information, along with the input, to independently construct the hash of $\texttt{r}$. If the verifier is able to accurately verify the commitment of $\texttt{r}$ on the blockchain using this hash, it confirms that the input value originates from $\texttt{r}$.

On hash generation. We propose three approaches for the hash generation of activity record: Merkle tree-based, homomorphic hashing and homomorphic hiding. In the first approach, Merkle tree-based hash generation is considered [28]. However, such an approach requires additional computational costs for Merkle tree root construction/reconstruction process. From a theoretical point of view, the second approach, *homomorphic hashing* [29] and the third approach, *homomorphic hiding* [30] are time efficient than the Merkle tree proof because these do not involve the tree root generation process. However, the homomorphic hash generation process must consider word-level granularity to calculate the verifiable final hash. Compared to *homomorphic hashing*, the third approach, *homomorphic hiding*, shows better efficiency in calculating the final hash as this approach calculates the hash with document-level granularity. Later in the evaluation section (i.e., Section6), we will compare these approaches to select the most feasible hash generation approach with lower computation overhead.

Approach 1: Merkle tree-based. Holder constructs the Merkle-tree root as the hash of activity record $\texttt{hash}(\texttt{r})$.

Hash generation: The Merkle-tree construction works up the tree's root from the leaf node using Merkle tree hashing [28]. In our implementation, the leaf nodes are considered a word as the proposed solution is intended for a document (i.e., holder activity record). The leaf node values are hashed first, then the hash values of left and right child nodes are concatenated and hashed to generate the value of intermediate nodes. Hashing up the tree, the hashed root $\texttt{R} = \texttt{hash}(\texttt{r})$ is generated.

Proof of input: During the Commitment Open Phase, for a verifier to construct the tree, they would need the hash values of other nodes, in addition to the hash value of the input, which they already possess. For instance, Fig 2 shows a Merkle tree where x1....x8 represents the leaf nodes. The non-leaf nodes R1.....R7 store the hash of their children. Considering x5 as the input, the verifier would need the hash values $\{\texttt{x6}, \texttt{R7}, \texttt{R2}\}$ and their positions in the tree to

complete the tree. Thus, these values are shared with the verifiers as *proof of input*.

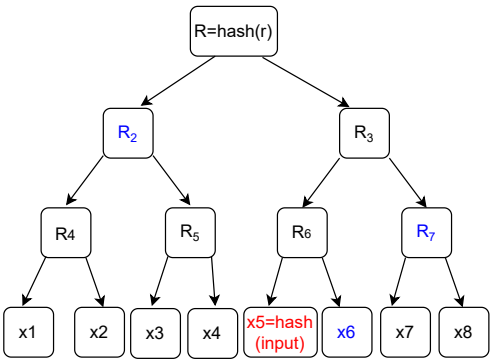

Figure 2: Illustration of Proof Verification for Merkle Tree Approach

Approach 2: Homomorphic hashing. In standard hashing, the holder would take all the previous and new data and compute a new hash. This may be computationally intensive if the data amount is significant. In such a scenario, the homomorphic hashing method is a solution where a holder computes the new hash using the existing hash and the new data [29]. This is more efficient, as it is not necessary to process all data into a new hash; instead, one can use the hash of existing data for the computation.

Hash generation: To initialise the homomorphic hash generation process, a nil hash value is taken. Then, to add a hash, the current hash value (i.e., the nil hash value) is added to the new one $\texttt{hash}(\texttt{r}_\texttt{i})$. In practice, two bytes (16 bits) are taken at a time from each hash value (i.e., nil hash and $\texttt{hash}(\texttt{r}_\texttt{i})$), added together, and then put the addition back into the current value of $\texttt{hash}(\texttt{r})$ where $\texttt{hash}(\texttt{r}) = \texttt{hash}(\texttt{r}_\texttt{i}) + \texttt{hash}(\texttt{r}_{\texttt{i}+1}) + ... + \texttt{hash}(\texttt{r}_\texttt{input}) + ... + \texttt{hash}(\texttt{r}_\texttt{n})$.

Proof of input: During the Commitment Open Phase, a verifier can add the hash values $\texttt{hash(input)}$ and $\texttt{hash}(\texttt{r}/\texttt{input})$ to verify whether the $\texttt{hash}(\texttt{r}) = \texttt{hash(input)} + \texttt{hash}(\texttt{r}/\texttt{input})$ where $\texttt{hash}(\texttt{r}/\texttt{input}) = \texttt{hash}(\texttt{r}_\texttt{i}) + \texttt{hash}(\texttt{r}_{\texttt{i}+1}) + ... + \texttt{hash}(\texttt{r}_\texttt{n})$. Since verifiers need $\texttt{hash}(\texttt{r}/\texttt{input})$, along with $\texttt{hash(input)}$ they already have, to complete the hash, $\texttt{hash}(\texttt{r}/\texttt{input})$ is the *proof of input*.

Approach 3: Homomorphic hiding. Homomorphic hiding is a construction of a hash function such that the hash of a composite block can be computed from the hashes of the individual blocks [30]. With a construction like this, we could distribute a list of individual hashes to verifiers, and they could use those to verify the distributed blocks as a composite as they arrive. In homomorphic hiding, the multiplication of two hashes will generate the same hash if their sum is hashed. The observation that $\texttt{g}^{\texttt{r}_0} * \texttt{g}^{\texttt{r}_1} = \texttt{g}^{\texttt{r}_0+\texttt{r}_1}$ (e.g., $2^3 * 2^2 = 2^5$).

Hash generation: In this approach, $\texttt{hash}(\texttt{r})$ is computed as $\texttt{g}^\texttt{r}$ (i.e., $\texttt{hash}(\texttt{r}) = \texttt{g}^\texttt{r}$ where $r = hash(input) + hash(r/input)$). Here, g is a random number.

Proof of input: Since r is the summation (or combination) of input and r/input (i.e., the complete activity record r except the *input*), therefore, during verification, verifier can check if $\texttt{hash}(\texttt{r}) = \texttt{g}^{\texttt{hash(input)}} * \texttt{g}^{\texttt{hash}(\texttt{r}/\texttt{input})}$. Since verifiers need $\texttt{hash}(\texttt{r}/\texttt{input})$, in addition to $\texttt{hash(input)}$ they already have, to complete the hash, $\texttt{hash}(\texttt{r}/\texttt{input})$ values along with g as the agreed values are the *proof of input*.

4.4. Proposed Commitment Generation Scheme

The holder holds the hash of activity record $\texttt{hash}(\texttt{r})$ to commit to, and the verifier receives the required hashes to open the commitment later. In this commitment protocol, g and h are the agreed values between the holder and the verifier.

Commit phase

- verifier samples a value h and sends h to the holder.
- holder generates $\texttt{hash}(\texttt{r})$ using a hash generation technique described in Section 4.3
- holder samples an opening value d, computes the commitment $\texttt{C} = \texttt{g}^\texttt{d}\texttt{h}^{\texttt{hash}(\texttt{r})}$, and sends C to the verifier.

Open phase

- holder sends the d, *proof of input* and *proof of record* to the verifier
- verifier verifies the NIZKP using *proof of record* and computes the $\texttt{hash}(\texttt{r})$ using *proof of input*.
- verifier checks whether $\texttt{g}^\texttt{d}\texttt{h}^{\texttt{hash}(\texttt{r})}$ matches the on-chain commitment C and either accepts if they match or rejects if they do not.

5. *CredAct* Construction

This section describes a specific construction of *CredAct*.

5.1. Identity Activity Sharing Workflows

For activity verification, *CredAct* includes seven algorithms named *Generation()*, *Setup()*, *Commitment()*, *Register()*, *Execute()*, *Prove()* and *Verify()*. We illustrate the communication workflows of *CredAct* along with the algorithms in Fig. 3 over four distinct phases, namely: Requirement function-based query creation, Commitment generation, Commitment verification, and Discount approval. We assume all entities are registered in the system using DIDs.

Phase-1: Requirement function-based query creation. The main algorithm used in this phase is Generation and Setup. Using the Generation algorithm, the verifier generates the required keys for *RV Contract* deployment. These keys are used to prove/verify the contents of *RV Contract* in the ledger. Next, each verifier generates and publishes the requirement functions $\texttt{f}_\texttt{r}$ in the contract using the Setup algorithm.

Phase-2: Commitment generation. The main algorithm used in this phase is Commitment and Register. Each holder computes a commitment C following the *commit phase* described in Section 4.4 and shares $(\texttt{C}, \texttt{r})$ with the issuer. Next, using the Register algorithm, the issuer first verifies that C is computed correctly from r. If verified, the issuer signs and stores the C on-chain.

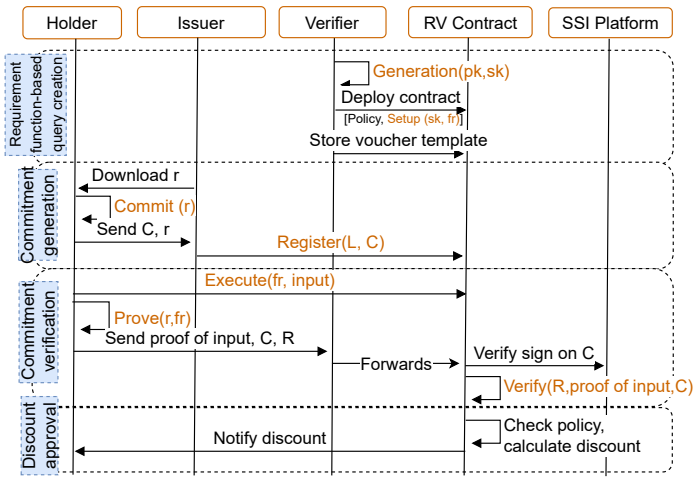

Figure 3: Communication Workflows

Phase-3: Commitment verification. The main algorithm used in this phase is `Execute`, `Prove`, and `Verify`. Each holder calls the *RV contract* to execute $\mathtt{f}_\mathtt{r}$ providing the relevant *input*. Next, using the `Prove` algorithm, the holder generates his *proof of record* to confirm his knowledge of the committed `r`. Finally, the holder sends *proof of record* `R`, *proof of input* and (`C`) to the verifier to open the commitment for verification. Using *RV Contract*, the verifier i) verifies the issuer's signature on `C`, ii) verifies `R` and iii) re-computes `C` (i.e., in the *open phase* described in Section 4.4) to determine whether it is the same as that stored on-chain.

Phase-4: Discount approval. The holder must deposit a certain amount to *RV Contract* when claiming for discount. The deposited amount is refunded to those who submit valid claims. Imposing such a claim fee will make any denial of service attack expensive for the attackers.

5.2. Used Claim Revocation

If any dishonest holder places the same claim of activity multiple times (e.g., claiming the same list of enrolled courses multiple times), *CredAct* can detect this multiple usage. *CredAct* uses the cryptographic accumulator scheme [31] for used claim revocation [32]. In this scheme, a finite set of claims is accumulated into a concise value of constant size. Each time a new claim is added to the accumulator, the accumulated value is updated [33]. The mechanism to invalidate a particular claim is to revoke the associated claim identifier. The claim identifier is a unique value calculated by hashing an identical attribute from the holder activity record (e.g., a hash of the student number shown in List. 1) and issuer signature. The revocation mechanism includes three steps. First, the verifier defines each claim identifier. Second, when the holder presents an activity claim the verifier generates the claim identifier and adds it to its revocation list as an accumulated value. Third, the next time the holder attempts to place the same claim, verification fails to pass through the revocation list because the claim identifier appears on the revocation list.

6. Evaluation and Discussion

To evaluate *CredAct*, we first discuss the privacy and security properties. Second, we demonstrate the scheme's feasibility by implementing concrete instances of *CredAct* and measuring its performance.

6.1. Security and Privacy Analysis

This section presents our qualitative security and privacy analysis of *CredAct*.

Security Analysis. Here we discuss some possible threats to *CredAct* and solutions. We consider that any holder and verifier who is part of *CredAct* can be as an adversary. Issuers (e.g., university) are assumed to be non-malicious in carrying out their duties such as signing the activity records to show that the holder owns the activity record (e.g., in the real world, a student's enrolment statement is signed by the university). However, for privacy analysis, we assume that issuers could be curious and can track the holders' behaviours such as information they disclose in credentials, which could be used to profile the holders.

- **Verifier Attack:** In this attack, the verifier is assumed to be honest but curious. Such a verifier accesses the holder's personal or activity records, they can use the information for undefined purposes, or distribute the data unwillingly or with intent. *CredAct* mitigates this attack by minimising the amount of shared data with verifiers, and hiding the identity behind their DID (when DID is used as pseudonym).
- **Public Ledger Attack:** An adversary can know that a discount claim is generated by analysing the holder's transactions (e.g., sending input to a requirement function). A holder generating discount claims using an one-off identifier like DID can reduce the risk of correlating the holder transactions to the *RV contract* [34].
- **Replay Attack:** We prevent proofs from being replayed by adding a timestamp as input during the proof generation process. The timestamp is public input, and therefore, it is submitted to the *RV contract* when verifying the proof. The contract then verifies the current time. The proof is valid if the current time is within the proof validity period. Otherwise, it is denied.
- **Denial of Service (DoS) Attack:** In *CredAct*, the holder can send discount claims as many as they are eligible. However, this requires a deposit fee payment in advance, which will be refunded if their claim is not fake. Therefore, DoS attempts to make fake claims would be an expensive attack that will not be attractive.
- **Multiple Spending Attack:** To avoid multiple claims for the same discount offer, we have used a dynamic accumulator to revoke the already used claim.

Privacy Analysis. This subsection provides a privacy analysis of *CredAct*.

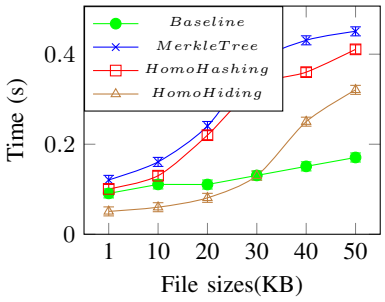

Figure 4: Commitment Gen. Time

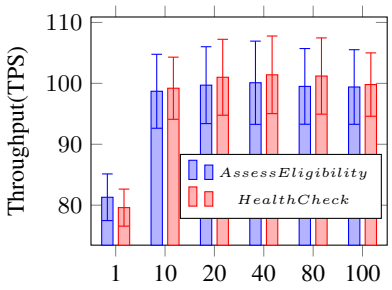

Figure 5: TPS for Function Exe.

TABLE 1: Performance of Requirement Function Execution

Requirement functions	# of inputs	function exe.(sec)
SNoMatch	1(256bit string)	0.0001252
EnrolCheck	1(256bit string)	0.0002371
WalkCheck	30(32bit int)	0.000815
WalkCheck	365(32bit int)	0.001508

Data minimisation. *CredAct* support the holders exercise data minimisation right by (i) requirement function-based querying of personal data to facilitate limited personal data collection and usage by the verifier, (ii) sharing of proofs about the shared data rather than the actual data.

Holder privacy and data verifiability. *CredAct* includes techniques for both preserving the holder's privacy as well as data verifiability. Using ZKP and *proof of input*, holders can prove the correctness of the input without revealing any additional information from their activity records.

Unlinkability in holder activity. As a part of identity protection, we want an issuer to be unable to learn about the holder activity (i.e., which holder is going to place a discount claim). Unlike the traditional commitment scheme, we generated the commitment on complete activity records rather than a particular attribute of the record so that the issuer cannot anticipate holder data-sharing intentions by looking at the committed data.

6.2. Implementation

We implemented *CredAct* scheme with all the interactions from Figure 3. We used Ethereum as a public blockchain network. Smart contracts were written using solidity to define the logic for executing requirement functions, performing transactions and storing the data on the blockchain. To interact our code with the smart contract, we imported the Application Binary Interface (ABI) that creates a JSON file after compiling a smart contract.

We deployed the smart contract in a Goerli test network (or briefly testnet) via the Hardhat development environment[1]. Goerli is an Ethereum testnet where we interacted with our main script with the deployed smart contract. The main script is based on nodeJS[2]. We created a Goerli wallet address against the holders' private key so that they can interact with the smart contract using their wallet. We used the Goerli faucet to deposit ether into the created wallet.

CredAct implementation has three versions for three hash generation approaches: Merkle tree proof, homomorphic hashing and homomorphic hiding. As an activity record, we considered a string array in our implementation. This can be a PDF file, for example, an enrollment statement uploaded to the student portal.

For the business logic of the smart contract, we implemented two use case scenarios of typical benefit schemes: *AccessEligibility* implements our motivating scenario where students need to prove their eligibility to access government services. It implements two functions: *SNoMatch* for checking if student numbers match, and *EnrolCheck* for checking enrolment status. Each function contains a single "if-statement". *HealthCheck* implements a scheme that requires users to submit daily step counts to get extra reward points. It implements one function *WalkCheck* for adding and checking step count numbers.

6.3. Performance Results

We deployed the blockchain node with the smart contract on a Linux Virtual Machine with 4 cores and 16GB RAM. We performed tests on *commitment generation* phase and *commitment verification* phase. These two phases are expected to contribute the most in computation time and overall blockchain performance since they involve cryptographic techniques and interact frequently with blockchain. We conducted experiments to validate the practicality of applying cryptographic techniques in verifying the activity records. We first compared the proposed commitment generation techniques with a traditional one to understand the time cost. Second, we showed the computation overhead for executing the on-chain requirement functions. Finally, we measured the throughput for requirement function execution to evaluate the blockchain performance.

Performance of Commitment Generation. To understand the performance of commitment generation in *CredAct*, we consider the traditional *Pederson Commitment* scheme as the baseline system. Our *CredAct* will consider all three approaches- Merkle tree, Homomorphic hashing and Homomorphic hiding techniques in the commitment generation process. We compared only the *commit phase* to avoid repetition since the *open phase* includes similar computation.

Fig. 4 shows the result where the X-axis represents the file sizes and the Y-axis represents the commitment generation time in seconds. We vary the number of file sizes (i.e., the activity records) from 1 KB to 50 KB. Each

[1]Hardhat, https://hardhat.org/
[2]Node.js, https://github.com/nodejs/node

data point measured is the average time taken over 30 runs. Although the graph depicts an increasing trend with higher file sizes, the baseline system remains almost stable in this graph compared to the proposed approaches. Among the three approaches, the Merkle tree approach takes a longer time(i.e., 45ms for 50KB of file size), and the homomorphic hiding approach takes only 32ms for 50 KB of file size. As we discussed in Section 4.3, the Merkle tree approach is computationally expensive for the construction/reconstruction of a tree. For the homomorphic hashing approach, the *commit phase* needs to calculate the hash of each word and then add the hashes to calculate the final hash. However, hash calculation is less complex in the homomorphic hiding approach than in the other two approaches. This leads to lower commitment generation time for the hiding approach.

On-chain Computation Overhead. We calculate the time to execute the requirement functions in the smart contract to determine the computation overhead incurred by on-chain interactions. For our tests, the requirment functions of two applications *AccessEligibility* and *HealthCheck* are implemented and deployed as a single *RV Smart Contract*.

Table 1 summarises the functions in applications, the number of inputs required in each function, and the performance results for each function. Since function *SNoMatch* includes a single "if-statement", the performance of function execution is high. Although *SNoMatch* and *EnrolCheck* both contain a single "if-statement" and one input equivalently, the function execution time of *EnrolCheck* takes twice compared to *SNoMatch*. The reason behind this is the *EnrolCheck* calculates the difference between two dates before executing the "if-statement". On the other hand, the function execution time is prominent in *WalkCheck* due to the higher input data type size. While comparing the same function *WalkCheck*, with different inputs (e.g., walking report for 30 days and 365 days), the execution times are significant for a higher number of inputs. In *WalkCheck* with 30 inputs, the time increases by 6 times compared with *SNoMatch*.

TABLE 2: Cost estimation

Transactions	Gas	Cost(Ether)	Cost($)
Deploy contract	793282	0.000793282	1.433
Store commitment	38772	0.000038772	0.070
Execute function	45413	0.000045413	0.082
Verify claim	87462	0.000087462	0.158

we have considered a gas price of 1 gwei (1 gwei= $10^{-9} ETH$) and 1 ETH = $1807(accurate at the time of writing, May 2023).

Performance of Blockchain. To understand blockchain performance in *CredAct*, we measured the throughput for requirement function execution. We performed this test using the *RV contract*, invoking it to execute the requirement functions. We have compared both applications by sending 10000 discount claims for each through JMeter, an application designed to load test functional behavior [35]. The throughput is measured by modifying the thread number and using 30 iterations on each data point.

Fig. 5 shows the transaction throughput in terms of the number of claim generations completed per second. From 10,000 generated claims passing through varying threads, we achieved the result shown in Fig. 5. The x-axis of the graph defines the number of threads, i.e., from 1 to 100 threads, whereas the y-axis defines the throughput (i.e., transactions per second(TPS)) ranging from 80 to 110 TPS to find the maximum throughput of the scheme. The results show that the scheme throughput consistently reaches 100 TPS with its highest TPS of 101 in this graph for *Health Check* application (i.e., the *WalkCheck* requirement function). For *Assessing Eligibility* application(i.e., *EnrolCheck* and *SNoMatch* requirement functions), the range of rising throughput is between 98-100 TPS.

Cost Estimation. The execution of *RV contract* code, like transactions, incurs costs measured as gas in the Ethereum platform. Gas cost is based on the complexity of the computation and the amount of data exchanged and stored. We store the commitment and requirement functions on-chain to lower the costs of invoking smart contracts, while the requirement policy DB is stored off-chain.

To understand the gas expenditure of *CredAct* scheme's on-chain components, we called the methods of the deployed *RV contract* daily on Goerli for 7 days (from 19 to 25 April 2023). Although we have used Goerli, a public test net where gas has no real value, we calculate the average fee cost of each type of transaction in gas units. Table 2 shows the cost of *CredAct* scheme's on-chain transactions. To sum up, the total result is 964929 units, around $1.743 at the time of writing[3]. This exchange rate can vary depending on the value of Ethereum.

7. Conclusion

This paper proposed an SSI-based activity data verification scheme called *CredAct*, focusing on minimised activity data sharing. We proposed a requirement function-based query to minimise the activity data collection by the verifiers. Secondly, we presented a ZKP-based cryptographic commitment that facilitates the holders to hide the use of identity data from the issuer, and supply verifiable data to the service provider without having to reveal the complete data. We demonstrated the feasibility of *CredAct* by implementing an instantiation of it using a scenario where university students are required to share their enrolment data with verifiers to obtain access to student services. The evaluation results of *CredAct* showed that the system is feasible with minimal operational overheads compared to traditional cryptographic techniques. We also perform a qualitative privacy and security analysis considering the possible threats to *CredAct*.

References

[1] Alessandro Acquisti. *Privacy and Security of Personal Information*, pages 179–186. Springer US, Boston, MA, 2004.

[3]as computed on 19th May 2023

[2] Your guide to supermarket rewards cards. https://www.news.com.au/compare-money/financial-wellbeing/supermarket-rewards/117F5AEC. Accessed: 2023-07-20.

[3] H. Sato, Y. Okabe, and M. Nakamura. User identification of pseudonyms without identity information exposure-a scenario in access federations. *Jour. of Information Processing*, 25:788–795, 2017.

[4] L. Stockburger, G. Kokosioulis, A. Mukkamala, R. Mukkamala, and M. Avital. Blockchain-enabled decentralized identity management: The case of self-sovereign identity in public transportation. *Blockchain: Research and Applications*, 2(2):100014, 2021.

[5] NSW Goverment. Apply for a concession opal card as a tertiary or tafe student, 2023. Accessed: August 21, 2023.

[6] R. Mukta, J. Martens, H. Paik, Q. Lu, and S. S. Kanhere. Blockchain-based verifiable credential sharing with selective disclosure. In *IEEE TrustCom*, pages 959–966, 2020.

[7] TT. Tran and HD. Le. Iu-smartcert: A blockchain-based system for academic credentials with selective disclosure. In *Future Data and Security Engineering. Big Data, Security and Privacy, Smart City and Industry 4.0 Applications*, pages 293–309. Springer Singapore, 2021.

[8] R. Tian, L. Kong, B. Zhang, X. Li, and Q. Li. Authenticated selective disclosure of credentials in hybrid-storage blockchain. In *IEEE 28th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 330–337, 2023.

[9] D. Chadwick, D. Longley, M. Sporny, O. Terbu, D. Zagidulin, and B. Zundel. Verifiable credentials implementation guidelines 1.0, 2019. https://w3c.github.io/vc-imp-guide/#selective-disclosure.

[10] B. Mortazavi and G. Kesidis. Model and simulation study of a peer-to-peer game with a reputation-based incentive mechanism. In *Information Theory and Applications Workshop*. Citeseer, 2006.

[11] R. Jurca and B. Faltings. An incentive compatible reputation mechanism. In *EEE International Conference on E-Commerce*, pages 285–292, 2003.

[12] Q. Jiang, C. Men, and Z Tian. A credit-based congestion-aware incentive scheme for dtns. *Information*, 7(4), 2016.

[13] S. Luo, Y. Sun, Y. Ji, and D. Zhao. Stackelberg game based incentive mechanisms for multiple collaborative tasks in mobile crowdsourcing. *Mobile Networks and Applications*, 21(3):506–522, 2016.

[14] Q. Liu, M. Liu, Y. Li, and M. Daneshmand. A novel game based incentive strategy for opportunistic networks. In *IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2015.

[15] Deloitte. Making blockchain real for customer loyalty rewards programs. Technical report, Deloitte center for Financial services, 2016.

[16] M. Naz, F. Al-zahrani, R. Khalid, N. Javaid, A. Qamar, M. Afzal, and M. Shafiq. A secure data sharing platform using blockchain and interplanetary file system. *Sustainability*, 11(24), 2019.

[17] V. Jaiman, L. Pernice, and V. Urovi. User incentives for blockchain-based data sharing platforms. *PLOS ONE*, 17(4):e0266624, 2022.

[18] Jiasi Weng, Jian Weng, Jilian Zhang, Ming Li, Yue Zhang, and Weiqi Luo. Deepchain: Auditable and privacy-preserving deep learning with blockchain-based incentive. *IEEE Transactions on Dependable and Secure Computing*, 18(5):2438–2455, 2021.

[19] S. Chen, B. Li, L. Rui, J. Wang, and X. Chen. A blockchain-based creditable and distributed incentive mechanism for participant mobile crowdsensing in edge computing. *Mathematical Biosciences and Engineering*, 19(4):3285–3312, 2022.

[20] L. Wang, Z. Cao, P. Zhou, and X. Zhao. Towards a smart privacy-preserving incentive mechanism for vehicular crowd sensing. *Security and Communication Networks*, 2021:5580089, 2021.

[21] B. Hoh, T. Yan, D. Ganesan, K. Tracton, T. Iwuchukwu, and J. Lee. Trucentive: A game-theoretic incentive platform for trustworthy mobile crowdsourcing parking services. In *IEEE Conference on Intelligent Transportation Systems*, pages 160–166, 2012.

[22] C. Jaffe, C. Mata, and S. Kamvar. Motivating urban cycling through a blockchain-based financial incentives system. In *International Joint Conference on Pervasive and Ubiquitous Computing and International Symposium on Wearable Computers*, page 81–84. ACM, 2017.

[23] Y. Ding, J. Li, K. Tei, and S. Honiden. Blockchain-based cooperative incentive system for emergency road right transferring. In *IEEE 10th Global Conference on Consumer Electronics (GCCE)*, pages 736–739, 2021.

[24] S. Utomo, T. Koshizuka, and N. Koshizuka. Blockchain-based incentive system for public trash bin. In *IEEE 9th Global Conference on Consumer Electronics (GCCE)*, pages 168–172, 2020.

[25] X. Chen, T. Zhang, W. Ye, Z. Wang, and Herbert H. Iu. Blockchain-based electric vehicle incentive system for renewable energy consumption. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 68(1):396–400, 2021.

[26] D. Reed, M. Sporny, D. Longley, C. Allen, R. Grant, and M. Sabadello. Decentralized identifiers (dids) v1.0; core architecture,data model and representations, 2020. https://w3c.github.io/did-core/.

[27] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In J. Feigenbaum, editor, *Advances in Cryptology*, pages 129–140. Springer, 1992.

[28] U. Chelladurai and S. Pandian. Hare: A new hash-based authenticated reliable and efficient modified merkle tree data structure to ensure integrity of data in the healthcare systems. *Journal of Ambient Intelligence and Humanized Computing*, 2021.

[29] K. Lewi, W. Kim, I. Maykov, and S. A. Weis. Securing update propagation with homomorphic hashing. *IACR Cryptol. ePrint Arch.*, 2019:227, 2019.

[30] M.N. Krohn, M.J. Freedman, and D. Mazieres. On-the-fly verification of rateless erasure codes for efficient content distribution. In *IEEE Symposium on Security and Privacy*, pages 226–240, 2004.

[31] Y. Ren, X. Liu, Q. Wu, L. Wang, and W. Zhang. Cryptographic accumulator and its application: A survey. *Security and Communication Networks*, 2022:5429195, 2022.

[32] C. Paquin and L. Nguyen. U-prove designated-verifier accumulator revocation extension. Technical Report MSR-TR-2015-40, Microsoft Corporation, 2015.

[33] Tolga Acar, Sherman Chow, and Lan Nguyen. Accumulators and u-prove revocation. In *Financial Cryptography and Data Security*, pages 189–196. Springer, 2013.

[34] Y. Kortesniemi, D. Lagutin, T. Elo, and N. Fotiou. Improving the privacy of iot with decentralised identifiers (dids). *Journal of Computer Networks and Communications*, 2019:8706760, 2019.

[35] J. Wang and J. Wu. Research on performance automation testing technology based on jmeter. In *International Conference on Robots & Intelligent System (ICRIS)*, pages 55–58, 2019.