

자기주권형 신원인증 시스템의 클러스터 노드 구성에 따른 성능 평가

김 정 현*, 강 창 훈*, 우 중 수*, 홍 원 기°

Performance Evaluation of Self-Sovereign Identity Systems Based on Cluster Node Configuration

Jeongheon Kim*, Changhoon Kang*, Jongsoo Woo*, James Won-Ki Hong°

요 약

자기주권형 신원인증 패러다임은 기존의 중앙 기관이 모든 신원정보를 통제하는 방식의 위험성에서 벗어나, 개인이 직접 신원정보를 관리하고, 특정 서비스 이용을 위해 원하는 정보만을 제출할 수 있다는 점에서 보안성이 뛰어나다. 그러나 여전히 사용자 대부분은 중앙 집중적 방식의 높은 사용자의 편의성과 속도에 익숙해져 있으며, 이는 자기주권형 신원인증 시스템의 가장 큰 걸림돌이 되고 있다. 본 논문에서는 자기주권 신원인증 시스템을 구현하기 위한 오픈소스 프로젝트인 A Cloud Agent - Python (ACA-Py)과 이에 특화된 하이퍼레저 인디(Hyperledger Indy) 블록체인을 활용하여 전체적인 시스템을 구성하였다. 또한, ACA-Py를 클러스터 형태로 구성하여, 클러스터 내부의 노드 개수 변화에 따른 실제 자기주권 신원증명 시스템의 자격증명 발급 및 검증 등의 성능을 평가한다. 실험 결과 노드가 1개인 경우 부하 분산 실패로 높은 응답시간과 에러율이 발생한 반면, 3개 또는 5개 노드로 구성된 클러스터는 에러 없이 우수한 성능을 보였다.

Key Words : Self-Sovereign Identity, Performance evaluation, Hyperledger Aries

ABSTRACT

The self-sovereign identity (SSI) paradigm offers high security by allowing individuals to manage their own identity information and selectively disclose specific details required for certain services, rather than relying on a centralized authority to control all identity information. However, most users are still accustomed to the high convenience and speed of centralized methods, which remains a significant obstacle for SSI systems. In this paper, we construct a comprehensive SSI system using A Cloud Agent - Python (ACA-Py), an open-source project designed for SSI implementation, and Hyperledger Indy blockchain. Additionally, we configure ACA-Py in a clustered form and evaluate the performance of credential issuance and verification within the SSI system as the number of nodes in the cluster changes. In our experiment, clusters with a single node experienced high response times and error rates due to load balancing failures, whereas clusters with 3 or 5 nodes showed excellent performance without any errors.

※이 논문은 경찰청이 지원한 ‘경찰건강 스마트관리(www.kipot.or.kr)’사업의 지원을 받아 수행된 연구결과입니다. [과제명: 경찰관 맞춤형 건강관리 서비스를 위한 지능형 빅데이터 통합플랫폼 개발 / 과제번호: 220222M01]

• First Author : Pohang University of Science and Technology (POSTECH), Department of Computer Science and Engineering, kjheon1118@postech.ac.kr

° Corresponding Author : POSTECH, Department of Computer Science and Engineering, jwkhong@postech.ac.kr

* POSTECH, Department of Computer Science and Engineering, {chkang, woojs}@postech.ac.kr

논문번호 : KNOM2024-02-02, Received November 18, 2024; Revised December 10, 2024; Accepted December 22, 2024

I. 서 론

전통적인 신원정보 관리 방식은

기업 또는 정부 등의 중앙 기관이 전체적인 정보를 저장하며 관리한다. 이러한 중앙 집중적 방식은 데이터 관리가 쉽다는 장점이 있으나, 데이터가 유출될 경우 대규모 피해가 발생한다는 단점이 있다. 또한, 개인정보 보호의 중요성이 주목받는 추세에 의해 자기주권 신원(Self-Sovereign Identity, SSI)이 등장하였다^[1].

자기주권 신원은 개인이 자신의 신원정보를 중앙 기관에 위탁 관리하지 않으며, 개인이 신원정보에 대한 통제권을 보유하는 패러다임을 뜻한다. 중앙 기관이 다량의 개인정보를 보관하지 않기에 데이터 유출의 위험성이 적으며, 개인이 원하는 서비스에만 개인정보 동의를 통해 신원정보를 제공할 수 있다. 이때, 서비스 이용에 필요한 특정 신원정보만을 제공할 수 있을 뿐 아니라, 영지식 증명(Zero-Knowledge Proof)을 활용하여 신원정보 자체를 제공하지 않고도 서비스가 요구하는 조건을 만족한다는 사실을 증명할 수 있다^[2].

이러한 자기주권 신원을 구현하기 위해 탈중앙화 식별자(Decentralized Identifier, DID)가 사용되며, 해외뿐 아니라 국내에서도 DID 기술을 활용한 자기주권 신원 생태계를 넓혀가고 있다^[3]. 마이크로소프트는 W3C가 정의한 DID 표준을 기반으로 하는 Entra Verified ID 서비스를 출시하였다^[4]. 해당 ID를 기반으로 다양한 자격증명을 발급하여 검증할 수 있도록 하는 서비스이다. 국내에서는 SKT가 타 주요 통신사 및 금융기관과 협력하여 DID 기반 모바일 전자증명 서비스인 Initial을 개발하였다^[5]. 신원증명뿐 아니라, 정부의 전자증명서 발급 및 고객센터 서류 제출 및 검증 등의 서비스를 제공하고 있다.

DID는 DID Method를 통해서 생성 및 관리되며, 다양한 서비스들이 독자적으로 DID Method를 구현하여 활용한다^[6]. 글로벌 공개 SSI 네트워크인 Sovrin 네트워크를 운영하는 Sovrin Foundation에서는 did:sov 메소드를 정의하여 사용한다^[7]. 앞에서 소개한 initial 서비스는 did:ssw 메소드를 정의하였다^[8]. 누구나 DID Method를 정의하고 구현할 수 있기에, 현재 DID Method는 200개를 넘어가고 있다^[9].

다양한 DID Method를 활용할 수 있는 오픈소스

프레임워크로는 OpenWallet Foundation에서 관리하는 A Cloud Agent - Python (ACA-Py)가 존재한다^[10]. ACA-Py는 SSI를 DID에 기반한 검증가능한 자격증명(Verifiable Credential, VC)[11]을 통해 구현하고, 자격증명 발급, 검증, 폐기 등의 기능을 제공하는 에이전트이다. Sovrin Network^[12], Findy Agency^[13], IDUnion^[14], IdRamp^[15], Initial^[5] 등 다양한 자기주권형 신원인증 시스템들이 ACA-Py를 기반으로 운영되고 있다.

ACA-Py 소스코드의 동시성 관리에 관한 부분을 분석해보면, 각 기능별로 하나의 ACA-Py 에이전트가 할당되어 시스템이 구성된 경우 시스템이 동시에 처리할 수 있는 요청 건수가 적도록 제약이 걸려있다. 즉, 더 많은 에이전트를 시스템에 추가할수록 소프트웨어적으로 제한되어 있는 동시 처리 작업의 최대 개수를 늘릴 수 있다.

본 연구에서는 ACA-Py를 기반으로 한 자기주권형 신원인증 시스템을 구성하여 자격증명 발급 및 검증과정의 성능을 측정한다. 시스템을 구성하고 있는 ACA-Py는 각 역할 군(발급자, 소유자, 검증자)별로 여러 개의 컨테이너가 배치되어 클러스터를 이루며, 이를 통해 더 많은 수의 요청을 적절히 분산하여 동시에 처리할 수 있도록 하였다. 이때, 클러스터를 이루고 있는 컨테이너의 개수에 따라 전체적인 시스템의 성능 변화를 측정 및 평가하였다.

본 논문의 구성은 다음과 같다. 2장에서는 자기주권형 신원인증 시스템에 관한 연구 배경과 ACA-Py의 성능 평가에 관련된 연구를 소개한다. 3장에서는 전체적인 자기주권형 신원인증 시스템의 구조 및 ACA-Py 클러스터에 관해서 설명한다. 4장에서는 실험 및 결과에 대해서 분석한다. 5장에서는 결론 및 향후 연구에 관하여 기술한다.

II. 연구 배경 및 관련 연구

2.1. 연구 배경

자기주권형 신원증명 시스템은 개인이 자신의 신원 정보에 대한 주권을 가지고 관리하며, 특정 서비스를 선택하여 신원정보를 제공할 수 있는 특징을 가지고 있다. 이러한 특성으로 인해 자기주권형 시스템은 발급자(Issuer), 소유자(Holder), 검증자(Verifier), 저장소(Registry)가 핵심 요소로 구성된다^[16]. 해당 네 가지 구성 요소의 구조와 상호작용 방식을 통해 신원정보를 중앙 기관의 개입 없이 관

리할 수 있다.

발급자는 자격증명(신분증, 증명서, 자격증, 면허 등)을 발행하는 믿을 수 있는 주체이다. 발급자는 소유자로부터 자격증명 발급에 대한 요청을 받을 수 있으며, 이에 대한 자체적인 유효성 검증을 거친다. 유효성이 확인되면 DID에 기반을 둔 자격증명을 발행하게 되는데, 이때 발급자는 자신의 비밀키를 활용하여 서명을 생성하고, 이를 자격증명에 포함한다. 발급자는 자신을 식별하기 위한 DID 문서와 공개키를 공개적이며 위변조가 어려운 저장소에 등록해 놓음으로써 특정 자격증명이 해당 발급자로부터 발급된 것임을 증명할 수 있다.

소유자는 자격증명을 소유하고 관리하는 개인이나 조직으로, 자신의 자격증명을 전자 지갑 등을 사용하여 안전하게 보관하고, 필요한 경우 검증자에게 부분적으로 제출할 수 있다. 검증자에게 제출하는 경우, 자격증명은 검증가능한 프레젠테이션(Verifiable Presentation)으로 구성되며, 발급자의 서명에 더해 소유자의 비밀키를 활용한 서명까지 포함하여 검증자에게 제출함으로써 특정 프레젠테이션이 해당 소유자로부터 제출된 것임을 증명할 수 있다.

검증자는 소유자가 제출한 자격증명(프레젠테이션)이 신뢰할 수 있는지 확인할 수 있다. 해당 자격증명이 지정된 발급자로부터 발급된 것이 맞는지 발급자 서명의 유효성을 통해 검증할 수 있다. 또한, 특정 자격증명에는 유효기간 등이 설정되어 있거나, 검증자가 제공하는 서비스의 조건을 충족시키지 못하는 경우(e.g. 서비스의 나이 제한이 19세이지만 제출된 자격증명에 기록된 정보는 19세 미만인 경우) 등을 고려하여 서비스 제공 여부를 결정할 수 있다.

저장소는 발급자, 소유자, 검증자가 DID 및 여러 메타데이터를 포함하는 DID 문서를 안전하게 저장하고 관리하는 공간이다. 많은 수의 자기주권형 신원인증 시스템이 저장소를 분산원장을 가지는 블록체인을 채택하고 있으며^[17], 이를 통해 DID 및 여러 데이터의 위변조를 막음으로써 자기주권 신원인증 시스템의 신뢰성을 높일 수 있다.

2.2. 관련 연구

Muhle et al. 은 자기주권 신원인증 시스템을 구축하기 위해서는 전체적으로 어떠한 요소들이 필요한지 설명하였다^[26]. Soltain et al. 은 ACA-Py 및 Hyperledger Indy 블록체인을 사용하는 Sovrin

Network과 다양한 자기주권 신원인증 시스템을 소개하였다^[27]. Jolocom^[28]은 Ethereum을 기반으로 동작하는 오픈소스 자기주권 신원인증 플랫폼으로, 생성된 DID는 IPFS^[29]에 저장된다. 또한 SelfKey^[30]와 LifeID^[31]는 영지식 증명을 사용하여 최소한의 필요한 데이터만 공유하도록 자기주권 신원인증 시스템을 구현하였다.

Siqueira et al. 은 ACA-Py에 기반을 둔 자기주권 신원인증 시스템을 구축하여 실제 성능을 측정하였다^[18]. 이때, 발급자와 검증자는 1개의 ACA-Py 노드로 구성하였고, 소유자는 2개의 노드로 구성하였다. 이를 기반으로 ACA-Py 노드 간의 연결, 자격증명의 발급 및 검증과정에 걸리는 시간을 측정하였으며, throughput과 서버의 CPU 사용량을 측정하였다. 동시에 발급 및 검증 요청을 하는 스레드가 많아질수록 throughput이 증가하였으나, 80회의 동시 요청 횟수가 넘어갈수록 에러율이 증가하는 것을 확인할 수 있었다. 해당 논문을 제외하면 ACA-Py의 실제 성능을 측정하는 연구를 확인할 수 없었다.

본 논문에서는 위에서 소개한 관련 연구에서 더 나아가, ACA-Py를 기반으로 자기주권 신원인증 시스템을 구축하고 이를 실제 운용한다고 가정했을 때, 이 시스템의 성능 개선을 위한 방법의 제안 및 검증을 목표하고 있다. ACA-Py 에이전트의 수를 클러스터 구성으로 늘리는 방법을 통해 시스템의 성능이 실제로 얼마나 향상될 수 있는지 검증한다.

III. 시스템 구조

본 장에서는 ACA-Py를 기반으로 구성한 자기주권형 신원인증 시스템의 전체 구조를 설명하고, ACA-Py를 클러스터 형태로 구성한 이유를 설명한다. 또한, 시스템에서 사용하는 자격증명 발급 과정과 검증과정에 대하여 설명한다.

3.1. ACA-Py 기반 자기주권형 신원인증 시스템 구조

일반적으로 자기주권형 신원인증 시스템은 발급자, 소유자, 검증자가 상호작용하는 구조를 이루며, 본 논문에서 구현한 시스템도 이와 같은 형태를 따른다. 시스템의 전체 구조는 그림 1과 같이, 리버스 프록시(Reverse Proxy), ACA-Py 컨트롤러, ACA-Py 클러스터, 하이퍼레저 인디 블록체인으로 구성된다. 리버스 프록시는 사용자들의 요청을 받아들이며 해당

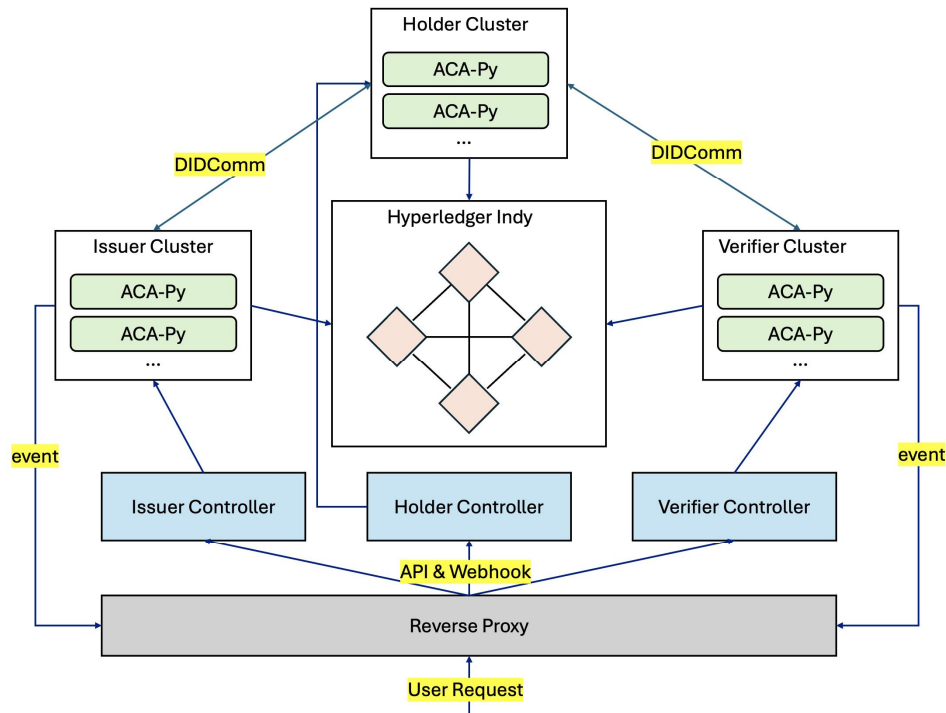


그림 1. 자기주권형 신원증명 시스템 구조
Fig. 1. Self-Sovereign Identity System Architecture

요청을 처리하여야 하는 ACA-Py 컨트롤러에 전달하는 역할을 한다. 또한, 컨트롤러가 ACA-Py 클러스터에 요청을 전달하면, ACA-Py는 작업을 처리한 후 해당 결과를 event 형태로 다시 리버스 프록시에 전달하여 event에 대응되는 컨트롤러에 전달하는 역할을 한다.

컨트롤러는 리버스 프록시로부터 전달된 사용자의 요청을 처리할 뿐 아니라, 자격증명 발급 또는 ACA-Py 간의 connection을 생성하는 과정에 대한 state를 자동으로 감지하여 추후의 과정을 자동으로 진행하며, 이에 관한 결과를 사용자에게 전달해주는 역할을 한다.

ACA-Py는 DID 발급, DID 및 자격증명 관리를 위한 지갑(Wallet) 생성, ACA-Py 간의 연결 생성, 자격증명 발급, 자격증명 검증의 역할을 수행한다. 이때, 발급자나 검증자의 ACA-Py와는 달리 소유자의 ACA-Py는 하나의 ACA-Py에서 여러 개의 지갑을 발급하고 자격증명을 관리할 수 있도록 하는 다중 테넌시(Multi-tenancy) 모드로 동작한다. 기본적으로 하나의 ACA-Py는 하나의 지갑을 생성하여 관리하기에 발급자 또는 검증자는 이에 적합하지만, 다양한 사용자들을 관리하여야 하는 소유자 ACA-Py는 이에 적합하지 않다. 이때, 본 논문에서는 소유자가 따로 지갑을 생성할 수 있는 모바일

어플리케이션을 사용하지 않는 경우를 가정한다. 그림 2와 같이 소유자의 ACA-Py를 다중 테넌시 모드로 동작시켜 하나의 tenant가 하나의 사용자를 관리하도록 설정하였으며, 같은 ACA-Py 내에 존재하는 tenant 간에는 같은 inbound와 outbound transport를 사용하도록 하여, 하나의 ACA-Py가 다수의 사용자를 관리할 수 있도록 구성하였다.

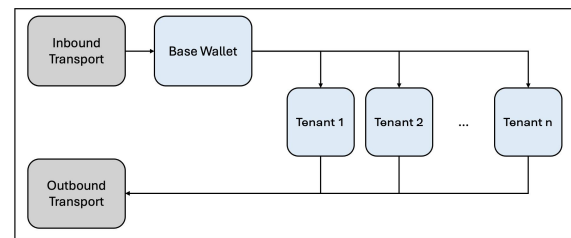


그림 2. 다중 테넌시 ACA-Py 구조
Fig. 2. Multi-tenancy ACA-Py Structure

DID 및 메타데이터의 저장소로는 하이퍼레저 인디(Hyperledger Indy)^[19] 블록체인을 사용하였다. 하이퍼레저 인디는 SSI를 구현하기 위해 설계된 오픈소스 분산원장 프로젝트로, 블록에 대한 합의를 수행하지 않고 트랜잭션 단위로 합의를 수행하여 분산원장에 기록한다는 특징이 있다. 또한, Aries 프로토콜을 통해 DIDComm 프로토콜을 지원하므로, ACA-Py와 상호 운용이 가능하다. 본 시스템에

서는 브리티시 컬럼비아에서 개발한 VON (Verifiable Organizations Network)^[20] 네트워크를 활용하여 4개의 indy 노드로 운영되는 하이퍼레저 인디 블록체인을 구성하였다.

3.2. ACA-Py 클러스터 구성

ACA-Py의 실제적인 성능을 평가했던 논문^[18]에 의하면, 동시에 80명 이상의 사용자가 동시에 자격증명의 발급 및 검증과정을 진행할 시, 점진적으로 에러 발생률이 올라가는 것을 확인할 수 있었다. 이러한 에러가 발생하는 원인으로는 자격증명 발급 및 검증과정은 모두 ACA-Py 간의 상호작용이 있어야 진행될 수 있으나, 각 ACA-Py 하나당 outbound transport를 처리할 수 있는 건이 200건이 한계치이며, 하나의 호스트에서는 오직 50건이 한계치이기에 발생하는 것으로 예상된다^[10].

이를 해결하기 위해서 본 시스템에서는 각 ACA-Py 역할 군마다 여러 개의 ACA-Py 노드로 구성된 클러스터를 구성하였다. 성능 평가를 위하여 클러스터의 구조를 변경하여 테스트를 진행한다. 하나의 클러스터가 가질 수 있는 최대 ACA-Py 노드의 개수는 5개이며, 최소 개수는 1개로 구성하여 차이를 두었다.

3.3. 자격증명 발급 및 검증과정

ACA-Py를 통해 자격증명 발급 및 검증을 위해서는 다양한 과정을 거쳐야 한다. 가장 먼저 소유자 ACA-Py에 연결된 사용자의 지갑을 생성하여야 한다. ACA-Py에서는 현재 askar 타입 지갑과 indy 타입 지갑을 모두 지원하고 있으나, indy 지갑을 사용하고 관리하기 위한 Indy SDK의 사용종료가 예정되어 있기에, 본 시스템에서는 askar 타입 지갑을 사용하였다. 다중 테넌트 모드로 동작하는 ACA-Py에서 지갑을 생성하게 되면, 지갑의 고유 id와 Bearer token이 생성되며, 이는 지갑과 관련된 기능들을 사용하는 데 필요하다.

지갑을 생성한 후, 자격증명을 발급하기 위해서는 발급자 ACA-Py와의 연결을 생성하는 과정이 선행되어야 한다. 본 시스템에서는 ACA-Py 간의 DID 교환을 통해 연결을 생성하는 Aries RFC 0023 (DID Exchange Protocol v1)^[21]을 사용하였다. 기존 ACA-Py 간의 연결을 생성하기 위해서는 Aries RFC 0160 (Connection Protocol)^[22]을 사용하여야 했으나, DID Exchange 프로토콜이 더 효율적이고 높은 보안성을 제공하여 Connection 프로토콜에 대

한 지원이 종료될 예정이다. 자격증명 발급을 담당하는 ACA-Py와 자격증명 검증을 담당하는 ACA-Py 모두 소유자 ACA-Py 내의 테넌트 별로 연결이 필요하다. 그림 3에서는 발급자와 검증자가 자신의 공개 DID를 전달하면, 소유자가 이를 기반으로 DID Exchange 요청을 전송하며, 발급자와 검증자는 이에 대한 반응을 전송하여 연결이 생성되는 과정을 설명하고 있다.

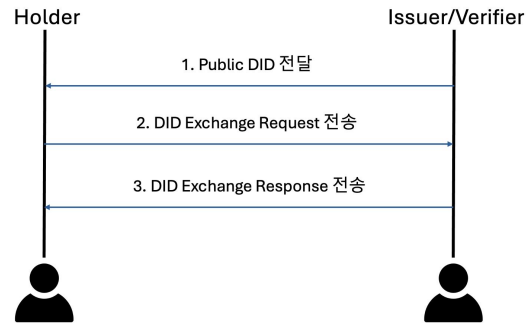


그림 3 DID 교환 과정
Fig. 3. DID Exchange Process

자격증명 발급 과정은 Aries RFC 0453 (Issue Credential Protocol 2.0)^[23]을 통해 진행되며 총 5단계의 과정으로 진행된다. 그림 4를 통해서 확인할 수 있듯이 자격증명 발급 과정은 소유자가 자격증명 발급에 대한 proposal을 발급자에게 전송하여 제안한다. 그 뒤의 offer, request, issue 모두 해당 proposal에 정의된 내용을 기반으로 진행된다. 이처럼 단계별로 발급 과정을 진행함으로써, 각 당사자에게 유연성과 제어권을 제공하여 상호 신뢰와 보안을 강화할 수 있다.

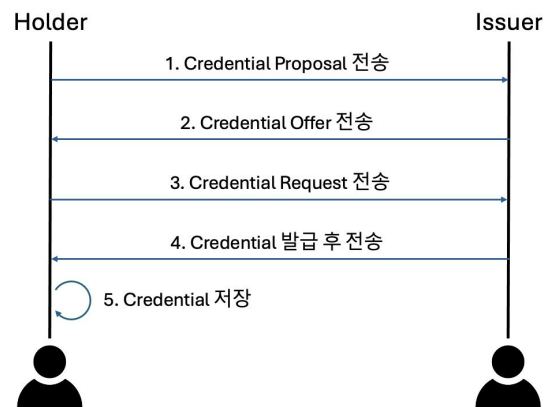


그림 4. 자격증명 발급 과정
Fig. 4. Verifiable Credential Issuance Process

자격증명 검증과정은 Aries RFC 0454 (Present Proof Protocol 2.0)^[24]을 통해 진행되며, 발급 과정과 유사하게 총 5단계의 과정으로 진행된다. 소유자가 자격증명 검증에 대한 proposal을 검증자에게 전

송하여 제안하며, 검증자는 이를 확인 후 해당 자격 증명 검증에 대한 presentation을 요청한다. 소유자가 해당 presentation을 제출하면, 발급자 서명, 소유자 서명 등의 과정을 통해 자격증명의 유효성을 확인한 뒤, 검증 결과를 소유자에게 전달한다.

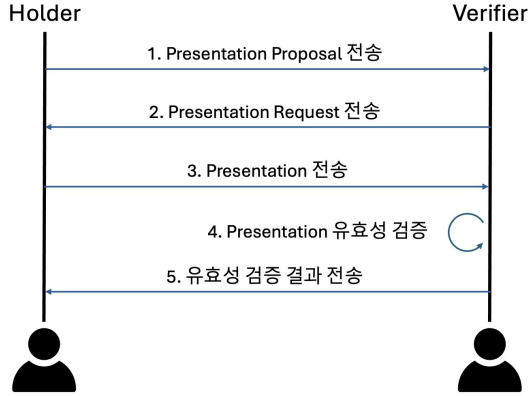


그림 5. 자격증명 검증 과정
Fig. 5. Verifiable Credential Verification Process

IV. 실험 및 분석

4.1. 실험 환경

자격증명 발급을 위해서는 해당 자격증명이 어떤 특성들을 포함할지 미리 정의하여야 하며, 이를 스키마(schema)라고 부른다. 또한, 스키마는 여러 발급자 사이에서 공통으로 사용될 수 있으므로, 해당 스키마에 추가로 발급자의 공개키 및 메타데이터를 포함하는 것을 자격증명 정의(credential definition)라고 한다. 자격증명 정의를 통해 특정 발급자가 특정 스키마에 따라 자격증명을 발급할 때, 해당 자격증명에 대한 발급자 정보 및 검증 방법을 지정할 수 있다. 스키마와 자격증명 정의 모두 하이퍼레저 인디 블록체인에 저장되어 신뢰성을 높일 수 있다. 본 논문에서는 실험을 위해 간단한 구조의 스키마를 사용하며, 해당 스키마의 구조는 표 1에서 확인할 수 있듯이 이름, 이메일 주소, 날짜로 구성된다.

표 1. 자격증명 스키마 구조
Table 1. Verifiable Credential Schema Structure

Schema Attributes	Attribute details
name	사용자의 이름
email	사용자의 이메일 주소
date	자격증명 발급요청 일자

본 연구에서는 ACA-Py 클러스터의 노드 개수를

조절함으로써 성능 변화를 테스트하고 분석하는 것을 목표로 한다. 이를 위해 ACA-Py 클러스터들(발급자, 소유자, 검증자)의 노드 개수를 1개, 3개, 5개로 설정하고, 동시에 자격증명 발급 및 검증을 요청하는 사용자의 수를 점차 늘리는 시나리오를 통해 테스트를 진행한다.

이때, ACA-Py 노드의 환경설정은 같은 클러스터 내부의 ACA-Py는 같은 설정 정보를 가지며, 세부적인 설정 정보는 표 2에서 확인할 수 있다. 각 노드는 비슷한 환경으로 설정되어 있으나, 소유자는 다중 테넌시 모드로 동작한다는 차이점이 있으며, ACA-Py 간의 연결을 생성할 때 공개적인 DID를 통해 신뢰를 확보하여야 하는 발급자와 검증자는 공개적인 초대를 가능하도록 설정하였다. 빠른 작업이 진행될 수 있도록 발급자는 소유자의 여러 단계에 걸친 자격증명 발급을 자동화할 수 있도록 설정하였으며, 반대로 검증자는 소유자의 자격증명 검증을 자동화할 수 있도록 설정하였다.

표 2 ACA-Py 환경 변수
Table 2. ACA-Py Environment Variables

Environment	Issuer	Holder	Verifier
version	1.0.0	1.0.0	1.0.0
db	postgres	postgres	postgres
wallet type	askar	askar	askar
multi-tenancy	false	true	false
did method	sov	sov	sov
public invites	true	false	true
auto accept invites	true	true	true
auto accept credentials	true	false	false
auto accept presentations	false	false	true

본 구현에서는 자기주권형 신원인증 시스템을 구성하는 모든 요소(블록체인, 컨트롤러, ACA-Py, 리버스 프록시)는 하나의 서버 내에서 동작하도록 구현하였다. 서버의 CPU는 AMD EPYC 7443P 2.75GHz 24 Core를 사용하였으며, RAM은 64GB

표 4. 노드 1개 구성 클러스터 실험 결과
Table 4. Node #1 Cluster Evaluation Result

API Request	Cluster - node #1					
	20 users		50 users		100 users	
	response time (ms)	error rate (%)	response time (ms)	error rate (%)	response time (ms)	error rate (%)
Step - Create wallet	100	0	150	0	272	0
01 - Holder create wallet	100 (5-274)	0	150 (4-777)	0	272 (5-1338)	0
Step - Connect issuer/holder	723	0	8596	22.35	24225	78.22
01 - Issuer invitation	9 (3-90)	0	10 (3-204)	0	73 (2-536)	0
02 - Holder receive invitation	714 (117-1295)	0	8586 (105-30272)	22.35	24152 (360-31133)	78.22
Step - Issue credential	1485	0	4215	32.85	6823	98.98
01 - Holder send proposal	314 (65-527)	0	2062 (5-30115)	27.61	4697 (5-30746)	90.1
02 - Holder send request	627 (200-983)	0	1194 (11-10157)	30	1986 (9-30869)	98.98
03 - Holder store credential	544 (108-916)	0	959 (10-2609)	32.85	140 (9-1027)	98.98
Step - Connect verifier/holder	746	0	1638	0	1264	0
01 - Verifier invitation	7 (3-42)	0	8 (3-88)	0	27 (3-544)	0
02 - Holder receive invitation	739 (110-1167)	0	1630 (105-3631)	0	1237 (125-2882)	0
Step - Verify credential	798	0	1279	33.83	149	97.22
01 - Holder send proposal	303 (54-573)	0	562 (4-1692)	25.19	49 (4-616)	66.67
02 - Holder send presentation	495 (156-745)	0	717 (20-2081)	33.83	100 (12-321)	97.22

표 5. 노드 3개 구성 클러스터 실험 결과
Table 5. Node #3 Cluster Evaluation Result

API Request	Cluster - node #3					
	20 users		50 users		100 users	
	response time (ms)	error rate (%)	response time (ms)	error rate (%)	response time (ms)	error rate (%)
Step - Create wallet	38	0	93	0	161	0
01 - Holder create wallet	38 (5-430)	0	93 (4-552)	0	161 (4-1507)	0
Step - Connect issuer/holder	201	0	667	0	1405	0
01 - Issuer invitation	7 (3-76)	0	10 (2-211)	0	15 (2-459)	0
02 - Holder receive invitation	194 (111-423)	0	657 (112-1752)	0	1390 (101-2634)	0
Step - Issue credential	587	0	1435	0	2749	0
01 - Holder send proposal	112 (63-238)	0	288 (65-710)	0	605 (68-1510)	0
02 - Holder send request	288 (207-562)	0	633 (215-1239)	0	1112 (200-1840)	0
03 - Holder store credential	187 (115-329)	0	514 (108-1167)	0	1032 (112-1840)	0
Step - Connect verifier/holder	203	0	703	0	1543	0
01 - Verifier invitation	7 (3-75)	0	7 (3-117)	0	20 (2-458)	0
02 - Holder receive invitation	196 (112-383)	0	696 (115-1560)	0	1523 (117-2520)	0
Step - Verify credential	312	0	765	0	1535	0
01 - Holder send proposal	96 (53-232)	0	279 (55-686)	0	632 (55-1144)	0
02 - Holder send presentation	216 (152-415)	0	486 (149-1018)	0	903 (158-1504)	0

표 6. 노드 5개 구성 클러스터 실험 결과
Table 6. Node #5 Cluster Evaluation Result

API Request	Cluster - node #5					
	20 users		50 users		100 users	
	response time (ms)	error rate (%)	response time (ms)	error rate (%)	response time (ms)	error rate (%)
Step - Create wallet	21	0	52	0	136	0
01 - Holder create wallet	21 (5-71)	0	52 (4-739)	0	136 (4-4621)	0
Step - Connect issuer/holder	156	0	581	0	798	0
01 - Issuer invitation	4 (2-15)	0	245 (2-1036)	0	282 (3-835)	0
02 - Holder receive invitation	152 (99-315)	0	336 (117-985)	0	516 (102-1413)	0
Step - Issue credential	471	0	784	0	1193	0
01 - Holder send proposal	85 (59-197)	0	169 (70-434)	0	260 (57-1479)	0
02 - Holder send request	239 (186-341)	0	364 (205-786)	0	534 (193-1331)	0
03 - Holder store credential	147 (105-255)	0	251 (106-589)	0	399 (119-1245)	0
Step - Connect verifier/holder	160	0	336	0	642	0
01 - Verifier invitation	4 (3-43)	0	21 (2-687)	0	66 (3-488)	0
02 - Holder receive invitation	156 (96-331)	0	315 (109-794)	0	576 (106-1505)	0
Step - Verify credential	253	0	455	0	679	0
01 - Holder send proposal	69 (47-179)	0	156 (51-398)	0	246 (54-985)	0
02 - Holder send presentation	184 (142-299)	0	299 (161-590)	0	433 (160-1116)	0

이며, OS는 Ubuntu 18.04를 사용하였다.

실험은 Postman^[25]의 API들을 연속적으로 호출할 수 있는 Collection 기능, 가상의 동시 접속 사용자 수를 설정할 수 있는 Performance Test 기능을 통해 진행하였다. 1개, 3개, 5개의 노드로 이루어진 클러스터별로, 각각 20명, 50명, 100명의 동시 접속 사용자 수를 설정하여 성능을 테스트하였다.

표 3. 성능 평가 지표

Table 3. Performance Evaluation Metrics

Evaluation Metrics	Details
response time	단계별 요청 처리 시간
throughput	초당 요청 처리 횟수
CPU usage	서버의 CPU 사용률
error rate	요청 에러 확률

이때, 성능을 평가하기 위한 지표는 표 3에서 볼 수 있듯이, 클러스터 내 노드의 변경에 따라 여러 단계로 나누어져 있는 지갑 생성, ACA-Py 간 연결 생성, 자격증명 생성, 자격증명 발급 과정의 단계별 요청 시간을 측정한다. 또한, 초당 요청 처리 횟수를 측정하여 최적의 설정을 확인하고, 전체적인 ACA-Py 클러스터들이 어느 정도의 CPU를 사용하는지 확인한다. 마지막으로, ACA-Py 자체의 한계로 인해 클러스터 내 노드의 개수가 적을 때 많은 요청이 들어오게 된다면 작업을 처리하지 못해 에러가 발생하게 되는데, 어느 정도의 에러가 발생하는지 측정한다.

4.2. 성능 분석

표 4, 5, 6은 각각 클러스터 내의 ACA-Py 노드가 1개, 3개, 5개로 구성되었을 때의 실험 결과를 나타낸다. 지갑 생성, ACA-Py 노드 간의 연결 생성, 자격증명 발급, 자격증명 검증과정 내의 세부 단계들로 나누어 응답시간의 최소, 최대, 평균을 측정함과 동시에, 에러율까지 같이 측정하였다. 전체 단계들 중 자격증명 발급에서 항상 가장 긴 응답시간이 소요되었고, 노드가 1개인 표 4의 경우, 더 많은 노드가 있을 때는 발생하지 않던 에러도 꽤 높은 비율로 발생하는 것을 확인할 수 있다.

응답시간의 경우, 3가지 경우의 클러스터 모두 동시에 요청하는 사용자의 수가 늘어날수록 응답시간도 같이 증가하는 경향을 띠고 있는 것을 알 수 있었다.

그림 6에서 볼 수 있듯이, 오직 노드 1개로만 구성된 클러스터의 경우, 다른 클러스터들보다 훨씬 긴 응답시간이 필요했다. 또한, 노드 3개로 구성된

클러스터보다 노드 5개로 구성된 클러스터가 조금이지만 더욱 빠른 응답시간을 나타내는 것을 확인할 수 있었다. 노드 1개로 구성된 경우가 응답시간이 오래 걸리는 원인으로는 부하를 분산하지 못하는 점뿐 아니라, 표 4에서 에러율을 확인하면 더욱 쉽게 설명할 수 있다.

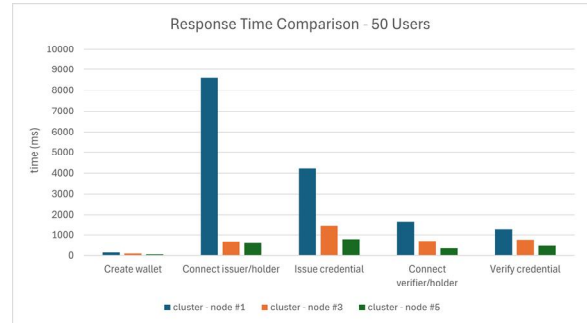


그림 6. 동시 사용자 50명 응답시간 비교

Fig. 6. Response time comparison for 50 users cases

노드 1개 클러스터의 경우 동시 사용자가 늘어날수록 에러율이 급격하게 상승하였다. 100 명이 동시에 요청하였을 때 특정 단계에서 98% 이상이 에러를 발생시킨 것을 확인할 수 있다. 그러나 노드가 3개 또는 5개인 클러스터들은 단 하나의 에러도 발생시키지 않고 요청을 문제없이 처리한 것을 확인할 수 있다. 이는 ACA-Py 노드 1개가 자체가 처리할 수 있는 한계점이 명백히 존재한다는 것을 의미한다.

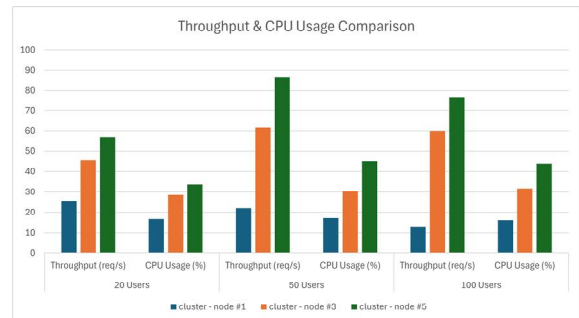


그림 7. 요청 처리량 & CPU 사용량 비교

Fig. 7. Throughput & CPU Usage Comparison

다음은 그림 7에서 클러스터별로 동시 사용자 수가 변화할 때, 전체 throughput과 서버의 CPU 사용량을 측정하였다. 3가지 클러스터 모두 throughput이 높을수록 CPU 사용량이 많기에, throughput과 CPU 사용량은 비슷한 경향을 띠고 있다는 것을 확인할 수 있다. 5개의 노드로 구성된 클러스터의 throughput 및 CPU 사용량이 가장 높았으며, 1개의 노드로 구성된 클러스터는 가장 낮았다. 이는 하나

의 ACA-Py가 처리할 수 있는 요청에 한계가 있다는 것을 뜻한다. 다만 3개, 5개 노드로 구성된 클러스터 모두 100명의 사용자를 동시에 처리할 때보다, 50명의 사용자를 동시에 처리할 때의 성능이 미세하게 더 높은 것을 확인할 수 있었다. 이는 요청의 분산이 적절히 이루어지지 못해 발생한 차이와 더불어, 50명의 사용자가 요청할 때 ACA-Py가 처리할 수 있는 적절한 수의 요청이 들어왔던 것으로 생각된다.

V. 결론 및 향후 연구

자기주권 신원증명의 패러다임은 자신의 신원정보를 직접 관리할 수 있으며, 원하는 정보만을 제공하는 등의 방법을 통해 보안성을 높이는 측면에서 굉장히 혁신적인 방법이다. 그러나, 아직 사용자 대부분이 특정 중앙 기관들이 직접 신원정보를 관리하는 중앙 집중적 방식을 사용하고 있는 이유는 서비스를 사용할 때의 편의성과 속도 때문이다. 많은 기존 서비스들이 구글 또는 네이버를 통한 간편 로그인을 지원하는 등 편의성 및 속도 측면에서 앞서고 있고, 따라서 자기주권 신원증명 방식도 이러한 격차를 줄이기 위한 노력이 중요하다.

본 연구에서는 현재 자기주권 신원증명을 구현하기 위해 주로 사용되는 오픈소스 프로젝트인 ACA-Py 노드의 실제 성능을 평가하였다. ACA-Py와 하이퍼레저 인디 블록체인을 통해 전체적인 시스템을 구현하였으며, ACA-Py의 노드들을 클러스터로 구성하여 클러스터의 구성을 다르게 하여 실제 성능을 평가하였다.

향후 연구에서는 동시 접속 사용자 수가 더 늘어나는 경우 시스템 운영을 위해 어느 정도 수의 노드가 필요한지 측정해볼 계획이다. 추가로, ACA-Py 기준에 사용했던 자격증명 발급 및 검증과정을 변경하여 ACA-Py 간의 연결을 생성하지 않아도 되는 connectionless 방법의 자격증명 발급 및 검증과정을 구현하여 기존 방법과의 성능을 비교해볼 예정이다. 또한, 본 연구에서는 모든 시스템의 구성 요소들이 하나의 서버에 위치하였는데, 발급자, 소유자, 검증자 역할별로 물리적으로 서버를 분리하여 전체적인 성능 변화를 측정해볼 예정이다.

References

[1] Tobin, A., & Reed, D. (2016). The inevitable rise

of self-sovereign identity. The Sovrin Foundation, 29(2016), 18.

- [2] Dieye, Mohameden, et al. "A self-sovereign identity based on zero-knowledge proof and blockchain." I EEE Access 11 (2023): 49445-49455.
- [3] Avellaneda, Oscar, et al. "Decentralized identity: W here did it come from and where is it going?." I EEE Communications Standards Magazine 3.4 (20 19): 10-13.
- [4] Introduction to Microsoft Entra Verified ID. Availab le at "https://learn.microsoft.com/en-us/entra/verified -id/decentralized-identifier-overview", Accessed at: 2024-11-09
- [5] Initial. Available at "https://www.skntenterprise.com/pr oduct/detail/209", Accessed at: 2024-11-09
- [6] Decentralized Identifiers (DIDs) v1.0, Available at "https://www.w3.org/TR/did-core/", Accessed at: 20 24-11-09
- [7] Sovrin DID Method Specification. Available at "htt ps://sovrin-foundation.github.io/sovrin/spec/did-metho d-spec-template.html", Accessed at: 2024-11-09
- [8] SSW DID Method Specification. Available at "http s://sktston.github.io/ssw-did/did-method-spec.html", Accessed at: 2024-11-09
- [9] DID Method Lists. Available at "https://diddirectory. com/", Accessed at: 2024-11-09
- [10] ACA-Py - A Cloud Agent - Python Github. A vailable at "https://github.com/openwallet-foundation /acapy", Accessed at: 2024-11-09
- [11] Sedlmeir, Johannes, et al. "Digital identities and v erifiable credentials." Business & Information Syst ems Engineering 63.5 (2021): 603-613.
- [12] Sovrin Network Homepage. Available at "https://sov rin.org/overview/", Accessed at: 2024-11-09
- [13] Findy Agency Homepage. Availabe at "https://findy- network.github.io/", Accessed at: 2024-11-09
- [14] IDUnion Homepage. Available at "https://idunion.or g/", Accessed at: 2024-11-09
- [15] IdRamp: Identity Verification Orchestration Hompa ge. Available at "https://idramp.com/", Accessed a t: 2024-11-09
- [16] Verifiable Credentials Data Model v1.1. Available at "https://www.w3.org/TR/vc-data-model/", Access ed at: 2024-11-09
- [17] Ferdous, Md Sadek, Farida Chowdhury, and Madin

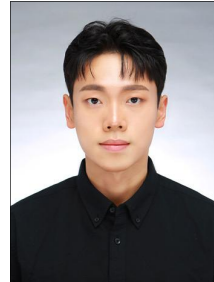
- i O. Alassafi. "In search of self-sovereign identity leveraging blockchain technology." IEEE access 7 (2019): 103059-103079.
- [18] Siqueira, Alexandre, Arlindo F. Da Conceição, and Vladimir Rocha. "Performance Evaluation of Self-Sovereign Identity Use Cases." 2023 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS). IEEE, 2023.
- [19] Bhattacharya, Manas Pratim, Pavol Zavarsky, and Sergey Butakov. "Enhancing the security and privacy of self-sovereign identities on hyperledger indy blockchain." 2020 International Symposium on Networks, Computers and Communications (ISNCC). IEEE, 2020.
- [20] BC Gov - Von Network Github. Available at "https://github.com/bcgov/von-network", Accessed at: 2024-11-09
- [21] Aries RFC 0023: DID Exchange Protocol 1.0. Available at "https://github.com/hyperledger/aries-rfcs/blob/main/features/0023-did-exchange/README.md", Accessed at: 2024-11-09
- [22] Aries RFC 0160: Connection Protocol. Available at "https://github.com/hyperledger/aries-rfcs/blob/main/features/0160-connection-protocol/README.md", Accessed at: 2024-11-09
- [23] Aries RFC 0453: Issue Credential Protocol 2.0. Available at "https://github.com/hyperledger/aries-rfcs/blob/main/features/0160-connection-protocol/README.md", Accessed at: 2024-11-09
- [24] Aries RFC 0454: Present Proof Protocol 2.0. Available at "https://github.com/hyperledger/aries-rfcs/blob/main/features/0454-present-proof-v2/README.md", Accessed at: 2024-11-09
- [25] Postman Homepage. Available at "https://www.postman.com/", Accessed at: 2024-11-10
- [26] Mühle, Alexander, et al. "A survey on essential components of a self-sovereign identity." Computer Science Review 30 (2018): 80-86.
- [27] Soltani, Reza, Uyen Trang Nguyen, and Aijun An. "A Survey of Self Sovereign Identity Ecosystem." Security and Communication Networks 2021.1 (2021): 8873429.
- [28] Jolocom. Available at "https://stories.jolocom.com/", Accessed at: 2024-11-14
- [29] Benet, Juan. "Ipfes-content addressed, versioned, p2p file system." arXiv preprint arXiv:1407.3561 (2014).
- [30] SelfKey. Available at "https://selfkey.org/", Accessed at: 2024-11-14
- [31] LifeID. Available at "https://lifeid.io/", Accessed at: 2024-11-14

김 정 현 (Jeongheon Kim)



2020~현재 포항공과대학교 컴퓨터공학과 통합과정
2017 포항공과대학교 IT융합공학
<관심분야> 블록체인 및 암호화폐, 머신러닝, 딥러닝

강 창 훈 (Changhoon Kang)



2019~현재 포항공과대학교 컴퓨터공학과 통합과정
2015 포항공과대학교 컴퓨터공학과 학사
<관심분야> 블록체인 및 암호화폐, 딥러닝

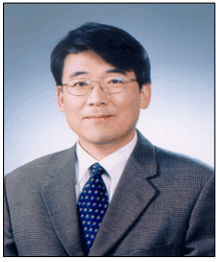
우 종 수 (Jongsoo Woo)



2020~현재 포항공과대학교 컴퓨터공학과 연구교수
2020~현재 포스텍 크립토티블록체인연구센터 공동센터장
2018~현재 DGIST 이사장
2016~2019 포스코교육재단 이사장
2014~2016 산업과학기술연구원 원장
2011~2014 포스코 기술연구원 원장

<관심분야> 암호화폐, 블록체인

홍 원 기 (James Won-Ki Hong)



2020~현재 포스텍 크립토블록
체인연구센터 공동센터장

2012~2014 KT 종합기술원,
원장

2008~2012 포항공과대학교, 정
보전자융합공학부장

2008~2010 포항공과대학교, 컴
퓨터공학과 주임교수

2007~2010 포항공과대학교, 정보통신연구소 연구소
장

2007~2011 포항공과대학교, 정보통신대학원장

1995~현재 포항공과대학교, 컴퓨터공학과 교수

1992~1995 Univ. of Western Ontario, 연구교수

<관심분야> 네트워크 트래픽 모니터링, 네트워크
및 시스템 관리, SDN/NFV, IoT, 무크기반 교육,
블록체인 및 암호화폐, Vmeeting 화상회의시스템