Demo: Decentralized Product Lifecycle Management Using Blockchain and Digital Twins

Ranjit Kannappan*,†, Julien Hatin*, Emmanuel Bertin*,†, Noel Crespi†
*Orange Innovation, 14000 Caen, France
†Samovar, Telecom SudParis, Institut Polytechnique de Paris, France
(ranjit.kannappan, julien.hatin, emmanuel.bertin)@orange.com, noel.crespi@telecom-sudparis.eu

***Abstract*—Managing a product throughout its various lifecycle stages is essential for all stakeholders involved. Efficient management is supposed to streamline process, enhance product quality, increase customer satisfaction, and improve profitability. The current architecture faces challenges that impact trust among stakeholders because of its centralized architecture. This paper presents a blockchain-based solution to enhance trust and transparency in product lifecycle management. Using Asset Administration Shell (AAS) as a standardized digital representation, we model and share product digital twins via IPFS. The proposed architecture demonstrates how smart contracts and IPFS enable the creation, sharing, and traceability of digital twins throughout the product lifecycle, fostering collaboration among stakeholders. This is illustrated through the use case of an electric vehicle, where digital twins are created for critical components like the EV battery and tire and enabling tracking across different lifecycle stages. Additionally, we evaluate the implementation for scalability, latency, and cost efficiency.**

***Index Terms*—Blockchain, Smart Contract, IPFS, Digital Twin, Asset Administration Shell**

I. Introduction

The product lifecycle spans from conception to recycling, involving multiple stakeholders who generate data to optimize processes and reduce carbon emissions [1]. While Product Lifecycle Management (PLM) tools bridge silos between functions like design and manufacturing [2], they lack inter-organizational collaboration [3], limiting access for SMEs due to high costs [4]. Data exclusivity, the lack of common syntax and semantics further hinder collaboration and innovation among stakeholders.

The solution we demonstrate is a decentralized product lifecycle system that utilizes digital twin to represent the product virtually. Digital twins play an important role to achieve lifecycle tracking. With the recent development of the Asset Administration Shell, a semantic standardization of digital twin to facilitate interoperability and information exchange[5], we are able to achieve secure distributed sharing with no single entity playing the part of monopoly. Our system utilizes Interplanetary file system (IPFS) to store lifecycle data in a decentralized manner.

In this paper, we present our implementation of decentralized lifecycle of product that uses a sample AAS file to create a digital twin and share them between lifecycle parties. Two types of digital twin are created in this demonstration. They are as follows

1) **Component digital twin**: These twins are digital representation of individual component of a product. Each component digital twin contains detailed information about the component, such as design data, performance data and lifeycle information depending on the type of sub model that is being used. several sub model templates are provided by IDTA, the creator of AAS. We utilize a sample AAS that is provided with 4 sub models in it. More sub models can be added as necessary using our system into the digital twin of the component.
2) **Product digital twin**: These twins contains a group of component digital twins. Several component digital twin make up a product and this product is represented by product digital twin. This twin can be generated by an assembler who receives multiple components to be assembled physically.

After creation of a digital twin, component digital twins are shared using two approach. One where sender shares the component twin to the receiver and another where the receiver requests the data to be shared. Both creation and sharing are authorized and recorded in blockchain using smart contracts. The data of the components are encrypted and stored in IPFS. The encrypted hash is shared to the appropriate parties.

II. Decentralized lifecycle Approach

A. System Architecture

In our proposed high level architecture Fig.1, we leverage several technologies to demonstrate the creation and sharing of digital twin of products in a decentralized manner.

Communication and Asset layer contains AAS sub model templates which are used to create AAS file for a component of a real asset. The first party in the lifecycle (often the manufacturer) initiates the creation by either creating AAS file externally using Aasx package explorer[6] and present the file as an input or that party can create a digital twin using empty aas file and then add submodel using our system. Creation of digital twin using aas file and adding subcomponents individually is recorded in the blockchain.

Storage Layer contains IPFS to manage off chain storage. IPFS is chosen because of its decentralized nature with content addressing for data integrity and cost efficiency. Data is encrypted and stored in IPFS. Features such as IPFS manifest and PubSub are utilized for structured storage and key sharing respectively.

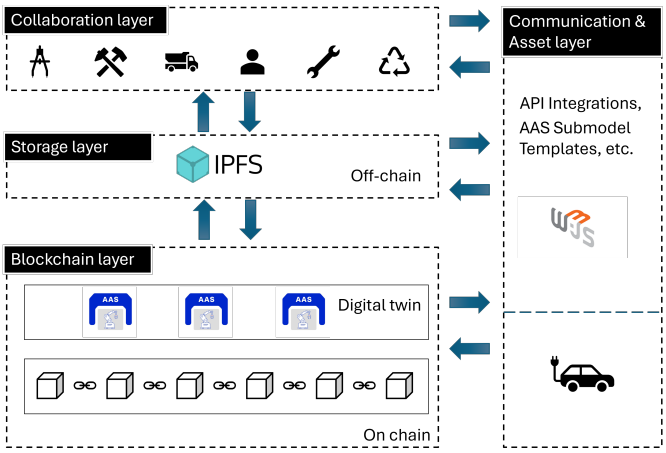

Fig. 1. Architecture of the implemented system

Blockchain Layer implements the Ethereum blockchain. Parity Ethereum (development mode) is used to run a local node, Truffle deploys smart contracts, and Web3.js interacts with the contracts. The following contracts are deployed:

- **ContractRegistry.sol**: Registers and retrieves contracts, creating instances of component digital twins.
- **Authorization.sol**: Manages attribute-based access control (ABAC) for secure data sharing and ownership tracking.
- **Signature.sol**: Handles transaction signatures for data sharing, including signature requests and retrieval.
- **ComponentDT.sol**: Manages digital twins of individual components, including ownership updates and subcomponent additions.
- **ProductDT.sol**: Aggregates component twins to represent the final product, managing data sharing and lifecycle processes.
- **MasterContractRegistry.sol**:manages the creation and tracking of ProductDT Contract instances. It allows authorized user to create new ProductDT instances using a set of addresses of componentDT instances

Collaboration Layer includes multiple stakeholders (designers, manufacturers, supply chain, users, maintainers, recyclers) as shown iN Fig.1. Stakeholders create and share digital twins, interact with contracts, and record all actions on the blockchain, ensuring transparency and collaboration.

B. Twin Creation & Sharing

When a digital twin is created using aas file (Fig.2), we parse the aas file which is in the format of aasx that has XML structure. For each submodel present in this file, encryption is done using AES-256 symmetric encryption and a key is produced. Encrypted data is stored in IPFS and a content identifier is generated(Orange CID in Fig.2). The content identifier is used to locate and retrieve the content. The key of the encrypted data and the generated CID is encrypted again using public key of the owner and stored in manifest (green keys in Fig.2). Manifest in IPFS is used to organize and store structured data. It is often used to store a set of CIDs, which in our case, encrypted key and CID is stored together in a manifest. Manifest CID is generated and is kept by the owner.

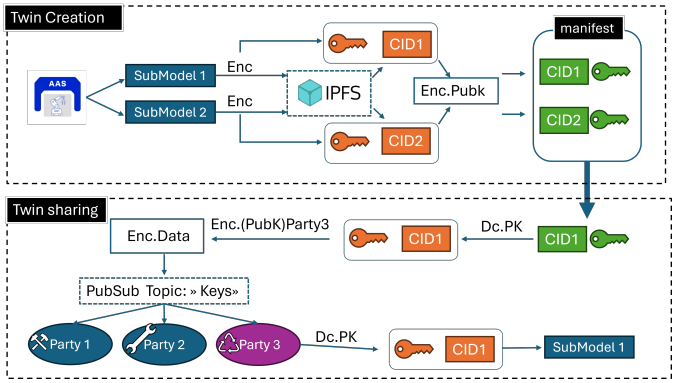

Fig. 2. Twin Creation and Sharing using sub models (Green represent the encrypted key and CID, while orange represent the decrypted/original key and CID)

In our system, data sharing is achieved through two approaches. The first is **First Party Data Sharing**, where the owner initiates sharing (Fig.2). The owner retrieves the CID and encrypted key (green key, Fig.2), decrypts them, and re-encrypts the batch (key and CID) using the receiver's public key ((pubK)Party 3). The encrypted batch (approximately 300 bytes) is published via IPFS PubSub under the topic "Keys." Subscribed parties receive the batch, and the intended receiver decrypts it to access the submodel. Encryption and decryption use sec256k1 and AES-256-GCM. Successful sharing is recorded in the blockchain via the Authorization contract.

The second is **Second Party Data Sharing**, where the receiver requests access to a submodel by interacting with the Signature contract to generate a signature request. The owner signs the request, and after verification, the first-party sharing process is triggered. The transaction is then recorded in the blockchain.

III. Implementation

This section details the implementation process, including the setup of the blockchain and IPFS, encryption mechanisms for secure data storage, and the workflows for creating and sharing digital twins using smart contracts. For our implementation, we developed smart contracts in Solidity [7] and deployed them on a local Ethereum blockchain. Parity Ethereum was used for this purpose, running in development mode with an instant seal engine and a single node on a local machine. The machine is equipped with 8 GB of memory, an Intel Core i7 processor, Intel HD Graphics 3000, and runs on Ubuntu 20.04.6 LTS. The contracts were compiled and deployed using the Truffle suite. IPFS node was setup and running in parallel to parity blockchain in the same machine.

Submodels are encrypted using the AES-256-GCM symmetric cryptographic algorithm, ensuring secure encryption and decryption with a 256-bit symmetric key and a random 16-byte initialization vector (IV), making ciphertext unique even

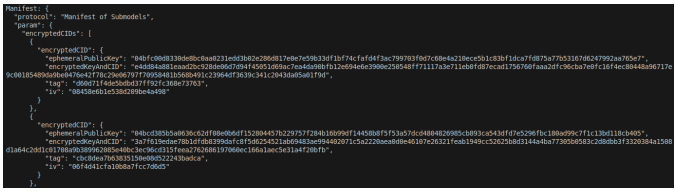

Fig. 3. Manifest of key and CID that are encrypted together

for identical data. An authentication tag protects the integrity of encrypted data by detecting unauthorized modifications. For sharing, a hybrid encryption mechanism combines Elliptic Curve Diffie-Hellman (ECDH) for key exchange and AES-GCM for data encryption. The symmetric key is encrypted using a shared secret derived from an ephemeral private key and the recipient's public key via ECDH, achieving forward secrecy. The manifest (Fig 3) includes the encrypted CID, key, ephemeral public key, IV, and tag for decryption. Only the owner can decrypt the submodels from the manifest using their private key, maintaining an organized and secure list of CIDs pointing to submodels.

Fig. 4. Re-encrypted components sent as message through PubSub topic-"Keys"

The sender chooses the submodels to share and re-encrypts the key & CID using private key of recipient. Two submodels "Nameplate" and "Document" are selected by the sender and the re-encrypted batch of these submodels are shown (Fig.4). Sender sends this encrypted batch through the PubSub channel in the topic : "Keys". Other parties can subscribe to this topic, all the message shared in this topic is visible to all parties(refer Fig.5).Only the recipient for whom the message is meant can decrypt the batch using their private key. This ensures a secure key sharing in a open channel.

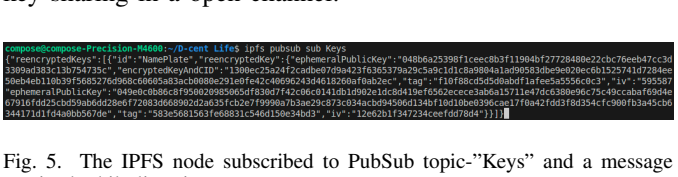

Fig. 5. The IPFS node subscribed to PubSub topic-"Keys" and a message received while listening.

For both First party and second party data sharing, this sharing mechanism is the same. In the first party data sharing, the sender manually selects the submodels to share to the reciever. For the second party data sharing, the reciever creates a request to recieve the submodels. Reciever interacts with the smart contract Signature.sol to creates a signature request with the senders address for required submodels (refer Fig.6)and *AccessRequested* event is emited after request creation. In front end, sender receives the request and the message contains hash of the component.

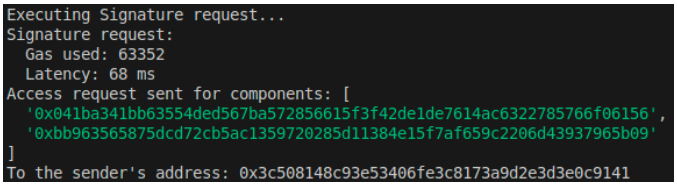

Fig. 6. Receiver creates a signature request and access request is sent to the senders address.

The message is then signed using the private key of the sender for each component(refer Fig.7). The signature along with its submodels are sent back to *CollectSignature* function of Signature contract. The collectSignature function is used to collect and verify digital signatures for the requested components. For each component, it calculates a messageHash using the keccak256 hash of the receiver and the component, then verifies the signature by extracting the signer's address from the signature using *ecrecover* function. If all signature are valid, the request is marked as signed and emits an event which executes the sharing algorithm from the front end of sender.

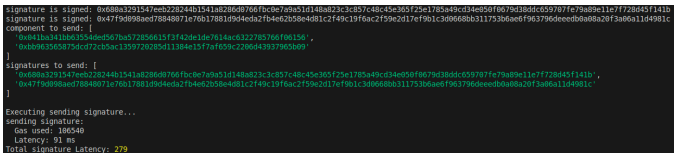

Fig. 7. The sender signs the request for each component requested and sends the signature to the smart contract.

IV. Evaluation

We evaluate our implementation in terms of deployment costs, latency, and scalability under varying concurrent request loads. The one-time deployment costs of all the smart contracts for the implementation using Truffle framework is summarized in Table I with a gas price of 20 gwei. Authorization contract and Contract Registry has the highest cost with 0.022 and 0.023 eth respectively. Nested data structures, such as attributes and roles, incur higher cost. In order to add or remove attributes, for loop is used to loop through array which increases gas costs as well. Including other contracts, the total deployment costs comes to 0.113 eth or 356.57 Euro. The same gas consumption is used to calculate the estimated cost on L2 (Arbitrum), reducing expenses by 95%.

The Component Twin creation process is the most resource-intensive one, with the highest latency of 167 ms (Table II). It involves creating a smart contract instance for two components using sample aasx files, representing a Electric vehicle battery and a vehicle tire with 4 submodels: Nameplate, Document,

TABLE I
COSTS OF DEPLOYMENT OF SMART CONTRACTS

Contract	Cost (Eth)	Cost (L1)	Est.Cost (L2)
Migration	0.005	15.78 €	0.07€
Authorization	0.022	69.42€	0.32€
ComponentDT	0.016	50.49€	0.23€
ContractRegistry	0.023	73.58€	0.33€
ProductDt	0.008	25.24€	0.12€
MasterContractRegistry	0.021	66.26€	0.30€
Signature	0.021	66.26€	0.30€
Total Cost	**0.113**	**356.57€**	**1.61€**

Note: The cost (L1) represents our implementation cost in Layer 1 blockchain, while Est. Cost (L2) represents the estimated cost in the Arbitrum Layer 2 blockchain.

Service, and Identification. Twin sharing, the least expensive process, shares 2 submodels (Nameplate and Document) in 65 ms, measured from retrieval to the execution of the *addAttribute* function in the authorization contract. Product Twin creation involves interacting with the MasterContractRegistry to link Component DT contracts into a Product DT. The total signature process for sharing 2 submodels takes 265 ms, covering signature request, sign & verify, and data-sharing execution, with costs of 0.43 and 0.33, respectively, for request and retrieval. The estimated costs in the L2 blockchain, as shown in Table II, are drastically cheaper compared to running the implementation on a public blockchain like Ethereum.

TABLE II
LATENCY AND COST EVALUATION

Process	Lat. (ms)	Gas	Cost (L1)	Est. Cost (L2)
Component Twin Creation	167	681242	41.40 €	0.20 €
Twin Sharing	64	75393	4.59 €	0.02 €
Product Twin Creation	65	490175	29.82 €	0.14 €
Signature Request	51	138352	8.42 €	0.04 €
Sign and Verify	65	106562	6.48 €	0.03 €

Twin creation and twin sharing processes are evaluated for latency under varying concurrent request loads using a multi-threaded approach with the *worker-threads* module. Each thread handles a subset of requests, and latency is measured per batch. Results show that as concurrent requests increase, latency decreases. For Twin Creation, latency drops from 21.112 ms (1 request) to 0.918 ms (10,000 requests), and for Twin Sharing, from 45.426 ms to 0.83 ms. This demonstrates the system's scalability and efficiency in handling high volumes of operations, such as creating multiple component twins or assembling them into product twins.

V. DEMONSTRATION SCENARIO

The proposed system is demonstrated through the lifecycle of an electric vehicle (EV) as a use case. The demonstration begins with the creation of digital twins for key electric vehicle components, such as the battery and tires, using Asset Administration Shell(AAS). The manufacturer initiates the process by generating component digital twins, encrypting

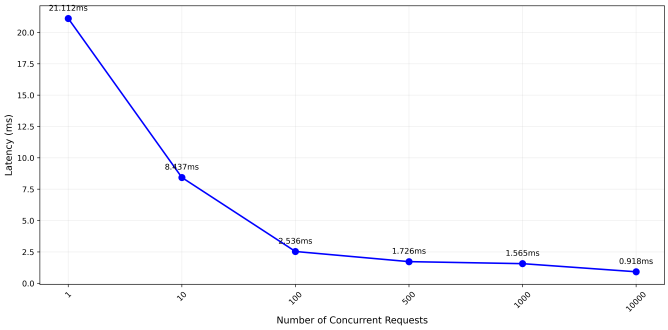

Fig. 8. Latency vs concurrent requests for Twin Creation

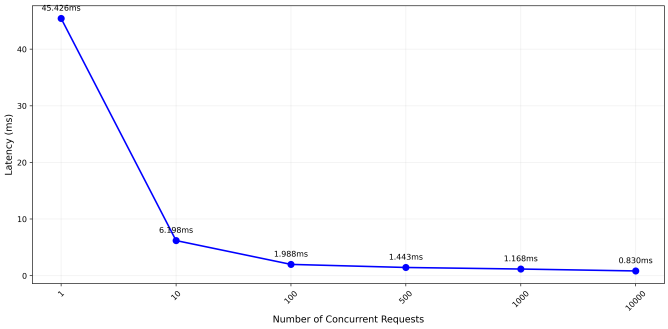

Fig. 9. Latency vs concurrent requests for Twin Sharing

their data, and storing it on IPFS. Next, the sharing process is showcased, where the manufacturer securely shares the digital twin of a battery with a service provider for maintenance tracking. Following this, the recycler demonstrates a second-party sharing scenario by requesting access to the tire's digital twin. The recycler interacts with the blockchain to send an access request to the vehicle user, who reviews and approves the request. Once approved, the system securely shares the encrypted data with the recycler. All interactions, including twin creation, sharing, and access requests, are recorded on the blockchain.

VI. CONCLUSION

In conclusion, the proposed decentralized product lifecycle management system utilizes blockchain, smart contracts, and IPFS to create and share digital twins securely and transparently. By utilizing Asset Administration Shell (AAS) for semantic standardization and implementing efficient encryption mechanisms, the system ensures interoperability and data integrity. The evaluation demonstrates scalability, with reduced latency under high concurrent requests, and highlights the cost-effectiveness of the approach, making it suitable for real-world lifecycle applications in collaborative and distributed environments. The electric vehicle use case, such as tracking batteries and tires through their lifecycle, showcases the system's ability to support sustainability by enabling efficient data sharing, maintenance tracking, and recycling efforts.

References

[1] J. Stark, "Product lifecycle management (PLM)," in *Product Lifecycle Management (Volume 1): 21st Century Paradigm for Product Realisation*, J. Stark, Ed., Cham: Springer International Publishing, 2022, pp. 1–32, ISBN: 978-3-030-98578-3. DOI: 10.1007/978-3-030-98578-3_1.

[2] M. W. Grieves, "Product lifecycle management: The new paradigm for enterprises," *Int. J. Product Development*, vol. 2, no. 1/2, p. 71, 2005.

[3] M. Hayat and H. Winkler, "Blockchain-based decentralized product lifecycle management: A framework for production network," in *2023 15th IEEE International Conference on Industry Applications (INDUSCON)*, 2023, pp. 1346–1351. DOI: 10.1109/INDUSCON58041.2023.10374585.

[4] S. de Oliveira and A. Soares, "A plm vision for circular economy," *IFIP Advances in Information and Communication Technology*, vol. 506, pp. 591–602, 2017. DOI: 10.1007/978-3-319-65151-4_52.

[5] P. I. 4.0 and ZVEI, "Details of the asset administration shell—part 1: The exchange of information between partners in the value chain of industrie 4.0 (version 1.0)," Federal Ministry for Economic Affairs and Energy, Berlin, Germany, Nov. 2018.

[6] *Aas package explorer*, 2023. [Online]. Available: https://github.com/eclipse-aaspe/package-explorer.

[7] E. Foundation, *Solidity — ethereum's smart contract language*, https://soliditylang.org/, 2024.