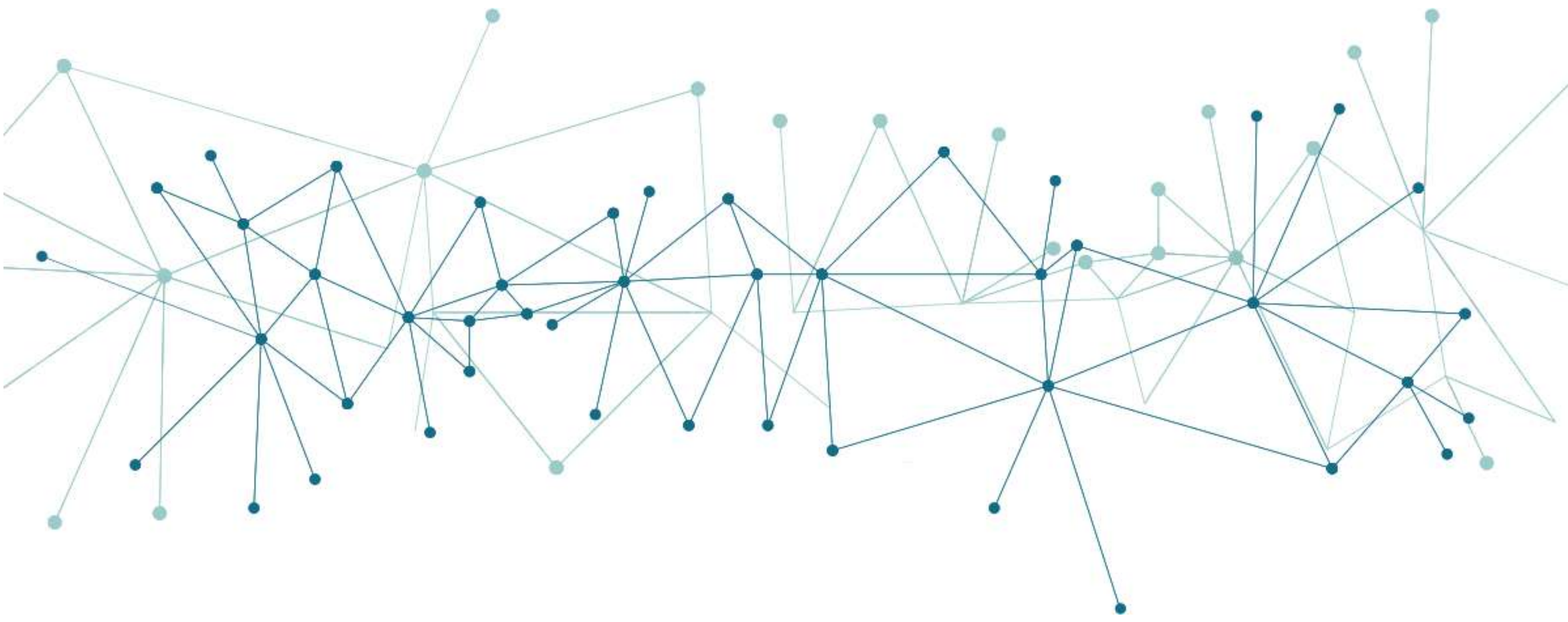
The eDREAM project is co-founded by the EU's Horizon 2020 innovation programme under grant agreement No 774478

enabling new Demand REsponse Advanced, Market oriented and secure technologies, solutions and business models

DELIVERABLE: D5.4 Blockchain platform for secure and distributed management of DR programs V2

Authors: Vincenzo Croce, Giuseppe Raveduto, Tudor Cioara, Claudia Pop, Ioan Salomie



Imprint

Blockchain platform for secure and distributed management of DR programs V2, May 2020

Contractual Date of Delivery to the EC:	31.05.2020
Actual Date of Delivery to the EC:	31.05.2020
Author(s):	Vincenzo Croce (ENG), Giuseppe Raveduto (ENG), Tudor Cioara (TUC), Claudia Pop (TUC), Ioan Salomie (TUC)
Participant(s):	ENG, TUC, E@W, ASM, EMOT
Project:	enabling new Demand Response Advanced, Market oriented and secure technologies, solutions and business models (eDREAM)
Work package:	WP5 – Blockchain-enabled decentralized network control optimization and DR verification
Task:	Task 5.1 - Secure energy data handling via blockchain/ledger
Confidentiality:	public
Version:	1.0

Legal Disclaimer

Copyright

Table of Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

API	Application Programming Interface
BIP	Bitcoin Improvement Proposal
CKD	Child Key Derivation
DAO	Data Access Object
DEP	Distributed Energy Prosumer
DLT	Distributed Ledger Technology
DR	Demand Response
DSO	Distributed System Operator
eDREAM	enabling new Demand Response Advanced, Market oriented and secure technologies, solutions and business models
HD	Hierarchical-Deterministic
IoT	Internet of Things
JSON	JavaScript Object Notation
P2P	Peer-to-Peer
PERT	Program Evaluation and Review Technique
PoA	Proof-of-Authority
PoW	Proof-of-Work
REST	Representational State Transfer
RPC	Remote procedure call
URI	Uniform Resource Identifier
UTXO	Unspent Transaction Output
WP	Work Package

Executive Summary

In this deliverable, we present the work done in the direction of further developing the eDREAM 2nd tier blockchain-based energy data and transaction storage which was presented in detail in D5.1 [1] with new scalability and privacy features by the integration of Zero-knowledge proofs on one hand and by extending the storage for using it with Bitcoin-like public blockchains.

Considering that all energy transactions are duplicated and shared across the network peer nodes, it is important to provide solid ways of ensuring the energy data privacy, thus we have implemented and integrated on top of the distributed energy ledger solution based on zero knowledge proofs. It provides privacy over the monitored values, while at the same time allowing the Smart Contracts to validate on chain the prosumer's activity. To ensure this, our solution replaces the monitored energy values of a prosumer, with a zero-knowledge proof that is generated by that prosumer. The proof is registered and validated on chain, being further used by the prosumer associated smart contract to evaluate and financially settle the prosumer's activity. The mechanisms for computing the digital fingerprints remain the same, the fingerprints being registered in the prosumer smart contract, sealing them in an immutable log and offering security features and the possibility of later validation. The implemented zero knowledge proof solution was described in the context of the decentralized flexibility management and control use case, the validation function estimating the deviation generated by a prosumer based on the monitored energy values from the flexibility requested. The evaluation use case result shows that our solution can ensure the privacy of prosumer energy data stored in the blockchain energy storage and can detect inconsistencies in data, e.g. tampering with the proof at the edge node level, prior to the energy transaction being signed and deployed on chain.

One of the main advantages of Bitcoin-like public blockchains as a data storage solution is notarization providing data timestamping, data integrity verification, and proof of data ownership. To benefit on these features for energy transactions we have proposed the extension of the energy storage solution for using it with Bitcoin-like public blockchains, without Turing-complete smart contracts support. On initialization two addresses for the sender and receiver are derived from the same parent key and the first address is funded with an initial transaction. The monitored energy values are periodically aggregated and digitally fingerprinted and hash-linked back to the prosumer using the algorithms defined in D5.1 [1]. A new energy transaction is created on the Bitcoin-like public blockchain featuring the sender and receiver addresses and the hash value generated for the monitored energy data will be stored in the OP_RETURN field. Finally, the transaction is signed, included in a block and broadcasted becoming thus verifiable.

1 Introduction

1.1 Purpose

This report provides an overview about the evolution of the secure and distributed blockchain-based data storage solution already defined in Task 5.1, describing the new features designed and developed to extend the 1st version of the platform. We have worked to ensure energy data privacy implementing and integrating Zero-knowledge proofs on top of the 2nd tier distributed energy data storage solution and to define a way to extend the 2nd tier solution to be used in public blockchains without support for Turing-complete smart contracts, supporting use cases in which greater security or greater transparency may be required.

1.2 Relation to other activities

WP5 uses some of the outputs from WP2 in terms of requirements and use cases as well as the outputs from WP3 in terms of energy demand/generation forecasting and prosumers' baseline assessment.

T5.1 in particular, defines the energy data storage platform on top of which the smart contracts-based solution for decentralized DR management (T5.2) and the DR verification and financial settlement solution (T5.3) are built (Figure 1).

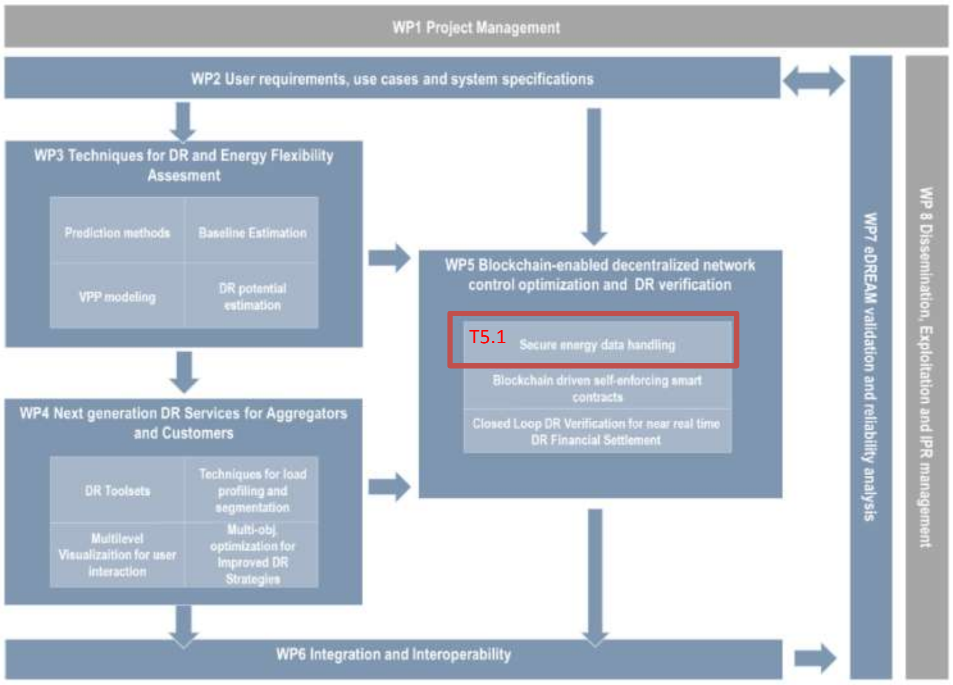

WP1 Project Management
WP2 User requirements, use cases and system specifications
WP3 Techniques for DR and Energy Flexibility Assesment
Prediction methods
Baseline Estimation
VPP modeling
DR potential estimation
WP4 Next generation DR Services for Aggregators and Customers
DR Toolsets
Techniques for load profiling and segmentation
Multilevel Visualization for user interaction
Multi-obj. optimization for Improved DR Strategies
WP5 Blockchain-enabled decentralized network control optimization and DR verification
T5.1
Secure energy data handling
Blockchain driven self-enforcing smart contracts
Closed Loop DR Verification for near real time DR Financial Settlement
WP6 Integration and Interoperability
WP7 eDREAM validation and reliability analysis
WP 8 Dissemination, Exploitation and IPR management

Figure 1. eDREAM PERT chart showing WP5 in relation to other work packages

1.3 Structure of the document

The remainder of the document is organized as follows:

- Section 2 presents the eDREAM privacy solution for energy data, based on Zero-knowledge proofs;
- Section 3 describes how it is possible to integrate the 2nd tier solution with Bitcoin blockchain for specific use cases in which greater security or greater transparency may be requested;
- Section 4 concludes the report.

2 eDREAM energy data privacy solution

The transparency of the blockchain based system is most of the times a desired feature, that brings openness and audit capabilities. However, in the energy domain the prosumer monitored energy data are sensitive and rather private information thus, a private solution is required. In the decentralized approaches, energy data are registered and stored locally in blockchain using prosumers' digital identities and then replicated and shared to all the network peers for validation. Considering that all energy transactions are duplicated and shared across the network peer nodes, it is imperative to provide solid ways of protecting this data. To ensure the energy data privacy in eDREAM we have implemented and integrated, on top of the 2nd tier distributed energy ledger, a solution based on zero knowledge proofs.

2.1 Zero-knowledge proofs

Zero-knowledge proofs have been added to digital currencies, such as Zerocoin [2] [3], to create anonymous Bitcoin transactions without the need of third parties. The Zero-Knowledge proof mechanisms, mainly succinct for Non-Interactive Zero-Knowledge proofs (zkSnarks) [4], are used to ensure the transactions' validation without revealing actual information about the parties involved. The Zero-Knowledge proofs require that a Verifier could easily check that the Prover is in possession of a secret, without revealing the actual secret to the Verifier. Consider a simple scenario, where the network is in possession of a hash value H. An actor wants to prove to the network that it holds the secret s, that hashed offers the value of H. In the traditional system, the actor would need to reveal the secret to the network. ZkSnarks aims to enhance this model, by allowing the actor to prove ownership over the data without disclosing any information regarding the secret. In this sense, zkSnarks introduces a proving function, that can be used by the actor to issue a *proof* showing that the private secret is indeed corresponding to the public information H, and a verification function that can be executed by any participant in the network in order to validate whether the proof corresponds to the public information H. The ZCash implements a zkSnarks mechanism as an improvement of the Bitcoin system that ensures the privacy of the transactions. It requires any transaction to be locked by a key, called a *commitment* and unlocked by a participant that holds the secret and who can generate the relevant proof, called the *nullifier*. The commitment is issued off-chain by the sender of the transaction, having as secret information the value and the receiver's public key. Similarly, the nullifier is computed off-chain by the receiver, proving that he owns the necessary information to spend the locked value. The commitment and the nullifier are registered on-chain. The commitments are registered in the commitment tree, and each time a transfer is required, a nullifier for one of these commitments needs to be issued and then registered in the nullifier set as future proof for avoiding double-spending attacks. The verifiers of the nullifier are all mining nodes that need to validate the integrity of the transactions.

All the information stored as energy transactions on the distributed ledger is public. Thus, to assure privacy preservation in a Smart Grid use case, mechanisms such as zero knowledge proofs can be employed. Zero Knowledge Proofs method allows one party, named the *verifier* to check if the other party, named the *prover*, has a secret information without the prover to divulge the information. In our view the Zero Knowledge proof mechanisms, mainly the zkSnarks, can be adapted to ensure the privacy of transactions involving energy monitored values. zkSnarks relies on three different functions that allows this functionality:

- *G- key generator*: a key generator function having as input the program function F used to validate the secret. As output, the generator issues two keys: the *proving key* (pk) used by the prover to run the program *F* and to issue the proof that it holds the correct secret, and the *verification key* (vk) used by the verifier to validate that the public value is indeed corresponding to the proof validating the undisclosed secret.

$$G(F) => (pk, vk)$$

- *P - Prover*: a prover function that is used by the prover to generate the proof that it indeed holds the secret. The prover must provide as input the proving key, pk, generated by $G,$ the public information H and the secret S which will be checked according to the validation rules provided by F. If the program F is successful, the proof is generated:

$$P(pk, H, S) => proof$$

- *V- Verifier*: the verification function is used by any player to check the integrity of the public information. The function receives as input the verification key, vk, together with the public information H that will be validated against the *proof* published by the prover:

$$V(sk, H, proof) => true/false$$

In eDREAM we will use this for the prosumer monitored energy data. Thus, instead of registering the monitored energy value for evaluation on chain (thus becoming visible for any other peer), our solution proposes that each prosumer to use ZK functions to hide the monitored values. The proof generated by the ZK Prover function is registered on chain where a Verifier function is used to check the validity of the proofs registered by the prosumers.

2.2 eDREAM 2nd Tier Energy Ledger Overview

The blockchain based energy data and transaction storage implemented in eDREAM was presented in detail in D5.1 [1]. To make this report self-contained in this sub-section we will present a short overview of the scalable 2nd tier solution which combines real-time energy data stored off chain in a distributed database with their digital fingerprint stored on blockchain. The Zero-Knowledge Proof will be integrated on top for adding energy data privacy features.

In our proposed hybrid solution, the real time energy data collected from IoT metering devices during a predefined time interval are hash-linked back at prosumer level in the order in which they had been sampled and then are stored off chain in a distributed database. As a result, a single energy transaction is created and signed for the entire interval by the prosumer. The transaction will be published on blockchain containing the average energy value registered and the associated hash fingerprint generated.

On blockchain, the average energy value registered will be used for further validation and business logic assessment, while the digital fingerprint transaction will be linked hashed-back with the digital fingerprints of energy transactions generated for previous time intervals. Whenever historical data is requested, the digital fingerprint of the off-chain stored data will be computed and checked against the on-chain registered fingerprint. In case the hashes coincide, it is concluded that the off-chain real-time registered data has not been tampered with. Otherwise, further inquiries can be made to detect the exact interval where the data has been modified, thus leading to a tamper evident system, where no changes can go unnoticed.

Figure 2 presents the layered architecture of our 2nd tier solution integrating Zero-Knowledge proofs:

- On the prosumer layer the energy data is fetched from a sensor associated with a device identifier and a measurement type, the hashing algorithms are run and the connections to store the raw values are implemented;
- On the off-chain layer scalable storage capabilities are provided and used for the real time data received from the sensors. A queue-based asynchronous messaging system is used to ensure the data sequence and system operation even in case of fluctuations in the monitoring data sampling rate;

- The blockchain layer stores the energy transactions associated with intervals of monitored energy data in blocks.

For each prosumer we have considered that its associated energy metering device provides monitored energy data over a time interval τ. The interval τ is split in N smaller disjoint intervals T_i, where each interval T_i is delimited by a start time and an end time $T_i = (T_S^i, T_E^i]$:

$$\tau = \cup_{i=1}^{N} T_i \; and \; \cap_{i=1}^{N} T_i = \emptyset$$

In each time interval T_i there can be L ordered discrete timestamps in which sensors are sampling new data:

$$T_i = \{t_k | t_k < t_{k+1}, \forall k = 0..L-1\}$$

We denote the monitored value sent by a sensor at a timestamp t_k as $M(t_k)$. From the sensor level, the monitored data value $M(t_k)$ is sent to the edge device at timestamp t_k. The edge device will forward this information directly to the asynchronous messaging system that stores the new sample data in the off-chain storage system in a sequential manner. At edge device level we had defined an online hashing algorithm that for each interval T_i will compute the digital fingerprint of all the monitored data received from the sensor at each discrete timestamp from that interval. At the end of interval T_i, the edge device will sign and register the digital fingerprint of the monitored data on blockchain, thus ensuring an immutable log of this value. The blockchain will implement a decentralized storage algorithm that is responsible to store and compute a hash of period P, composed of all the digital fingerprints received for each interval T_i.

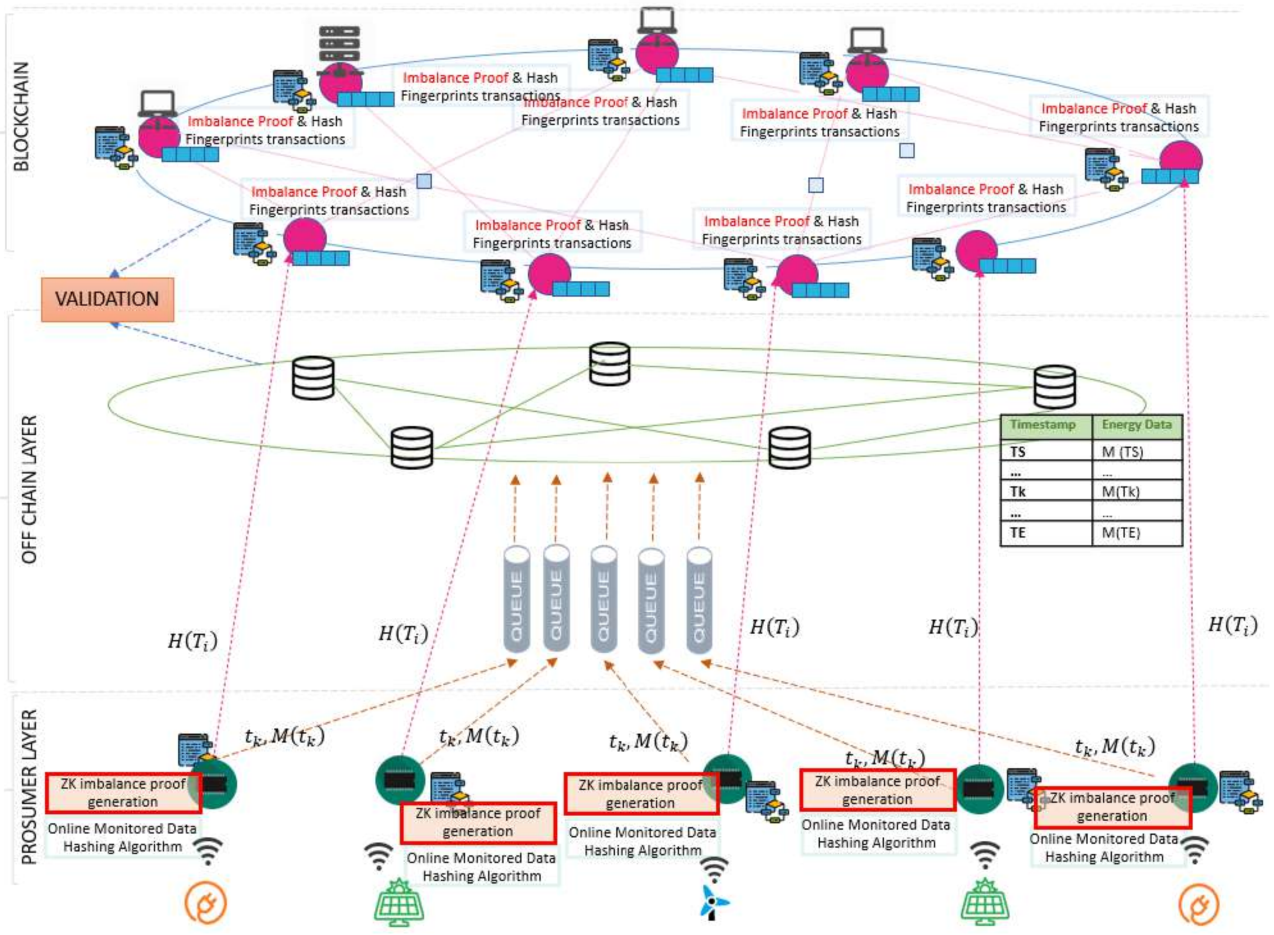


Figure 2: eDREAM 2nd tier solution for energy data and Zero-Knowledge Proofs integration

Zero Knowledge proof solution will be used to provide privacy over the monitored values, while at the same time allowing the Smart Contracts to validate on chain the prosumer's activity. The proposed ZK solution, replaces the aggregated energy monitored value, with a ZK proof which is generated at prosumer level. The proof is registered and validated on chain, being further used by the prosumer associated Smart Contract to evaluate and financially settle the prosumer's activity. The mechanisms for computing the digital fingerprints remain the same, the fingerprints being registered in the prosumer smart contract, sealing them in an immutable log and offering security feature and the possibility of latter validation.

2.3 2nd Tier DLT and Zero-Knowledge Proofs

For a better understanding of the mechanisms involved and the potential of the eDREAM ZK-Proof based privacy solution we describe them in the context of decentralized flexibility management and control use case. In this use case the Aggregator is responsible to inject the flexibility request (i.e. flexibility order) in the Smart Contract associated to a prosumer and regulating its behaviour. Whenever new monitored energy data were registered, the associated energy transactions are signed by the edge devices. The mined transactions were responsible to trigger the prosumer associated smart contract execution, which evaluates the deviations between the actual monitored energy profile and the flexibility request injected by the Aggregator. In case of successful delivery, the prosumer would be rewarded (i.e. incentives are being allocated), while in case of significant deviations the prosumer would be penalized proportionally with the imbalance created.

In this case once a transaction containing the monitored energy data values is published and mined, every network participant has access to that information. Even if the identity of the prosumer is pseudo-anonymized due to the lack of mapping information between the prosumer's address in the chain and the prosumer's identity in real life, it could potentially become a security breach, since if a malicious entity would be able to establish this connection, it could also extract behavioural patterns from the public energy profiles registered in chain.

Thus, we have implemented a privacy preserving solution that integrates zero knowledge proofs to hide the energy monitored data and requested flexibility profiles, while registering on chain only the deviations and a ZK proof validation is conducted to check that indeed the deviation is correctly computed.

Table 1: Generator function for ZK Proofs in eDREAM

Generator Function	Generated Keys	*F* Program	Validation Rules Implementation

zokrates	$PK_{flexibility}$
	
	$VK_{flexibility}$
		$F_{flexibility}\left(\begin{matrix} H_{hour}, \\ S_{monitoredValue}\end{matrix}\right)$	field result =
			if flexibilityRequest[H_{hour}] > $S_{monitoredValue}$
			then flexibilityRequest[H_{hour}] - $S_{monitoredValue}$
			else $S_{monitoredValue}$ - flexibilityRequest[H_{hour}]
			fi

As depicted in Table 1, as a generator function we have used the zokrates tool [5] that given custom functions can generate the ZK keys to be used by each individual prosumer. As validation rules, we propose the validation of the monitored value against a pre-contracted Flexibility Request. Instead of having the validation function provide a true/false output, we have implemented a modified version where the output of the function is the deviation generated by the monitored value from the flexibility requested.

Table 2: Prover and Verifier Functions for ZK Proofs

Prover	**Prosumer**
Prover Function	$P(PK_{flexibility}, H_{hour}, S_{monitoredValue})$
Prover Output	$Proof_{deviation}$

Verifier	Smart Contract
Verifier Function	V ($VK_{flexibility}, H_{hour}, Proof_{deviation}$)

As depicted in Table 2, the proving key $PK_{flexibility}$ will be used by the prosumer to generate the proof, while the verifier key $VK_{flexibility}$ will be used by a Smart Contract to verify the proofs provided by the prosumer, at the same time providing information about the deviation generated. Thus, the only public information registered on chain is the hour H_{hour} associated to the proof containing the output of the F Program, that is the deviation measured by the prosumer.

2.3.1 Flexibility Request Communication

In this section, we present how the zero-knowledge proof solution is used to assure the privacy of the data exchange between the Aggregator and the Prosumer for flexibility delivery and successful enrolment of the prosumer in the program.

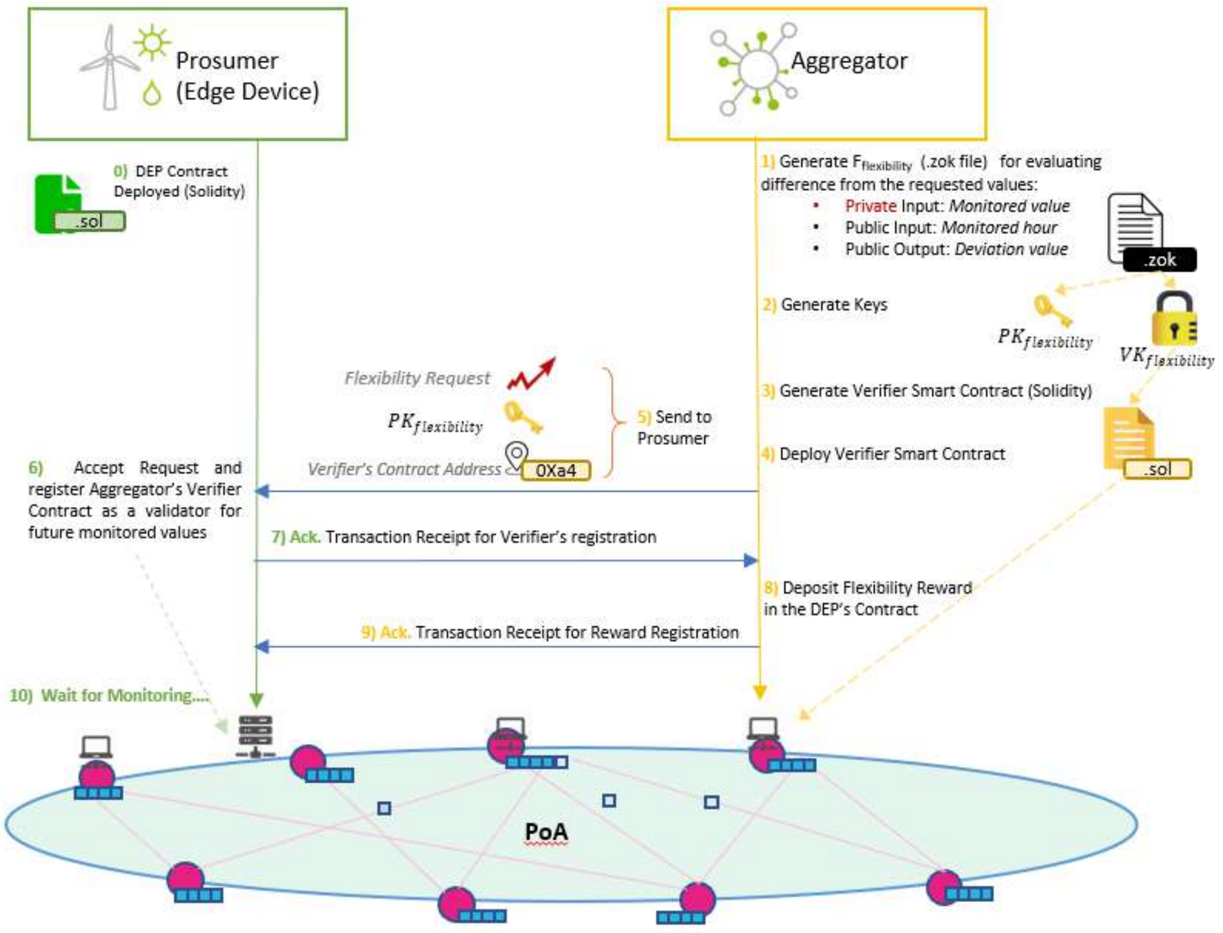
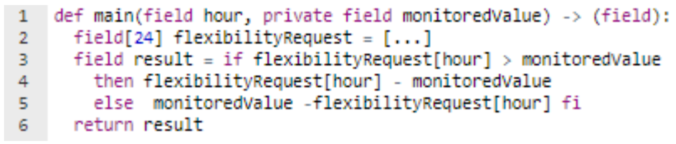

Prosumer (Edge Device)
Aggregator
0) DEP Contract Deployed (Solidity)
.sol
1) Generate $F_{flexibility}$ (.zok file) for evaluating difference from the requested values:
• Private Input: Monitored value
• Public Input: Monitored hour
• Public Output: Deviation value
.zok
2) Generate Keys
$PK_{flexibility}$
$VK_{flexibility}$
Flexibility Request
$PK_{flexibility}$
Verifier's Contract Address
0Xa4
5) Send to Prosumer
3) Generate Verifier Smart Contract (Solidity)
4) Deploy Verifier Smart Contract
.sol
6) Accept Request and register Aggregator's Verifier Contract as a validator for future monitored values
7) Ack. Transaction Receipt for Verifier's registration
8) Deposit Flexibility Reward in the DEP's Contract
9) Ack. Transaction Receipt for Reward Registration
10) Wait for Monitoring....
PoA

Figure 3: Zero knowledge proof for assuring the privacy of data exchange between aggregator and prosumers over blockchain

The steps involved in this process are depicted in Figure 3, and detailed below:

- Step 1: Once the Aggregator determines the Flexibility Request profile that should be ordered from each individual Prosumer, it generates the ZK program ($F_{flexibility}$) enforcing the verification rules, as depicted in Figure 4. At any time, given the hour and the private monitored value, the $F_{flexibility}$ will compute the deviation value as an absolute difference between the requested and the monitored value.

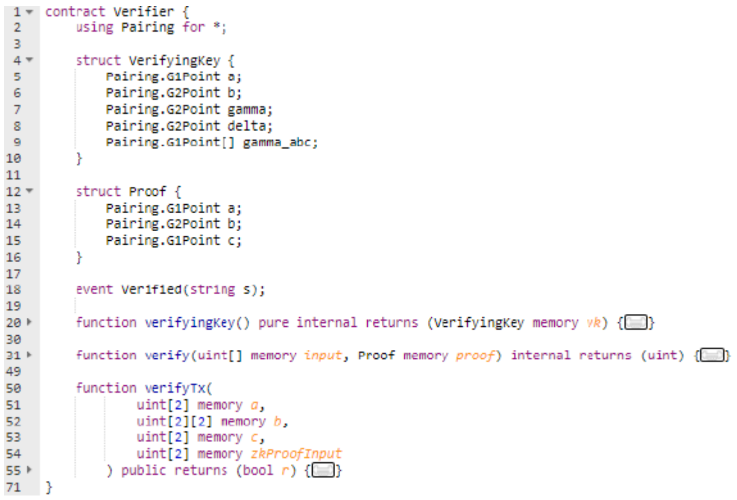


```
def main(field hour, private field monitoredValue) -> (field):
  field[24] flexibilityRequest = [...]
  field result = if flexibilityRequest[hour] > monitoredValue
    then flexibilityRequest[hour] - monitoredValue
    else  monitoredValue -flexibilityRequest[hour] fi
  return result
```

Figure 4: $F_{flexibility}$ ZK Program for aggregator to prosumer communication

- <u>Step 2:</u> Once the setup of the $F_{flexibility}$ program is complete, it will generate two keys: a prover key ($PK_{flexibility}$) and a verifier key ($VK_{flexibility}$). The prover key will be used by the prosumer to generate a proof, $Proof_{deviation}$ used to validate that according to the energy monitored value (now a private information) registered by the sensors, the imbalance value publicly registered is indeed correct. As depicted in Figure 4, for running the $F_{flexibility}$ program, the only public information specified are the hour and the output of the function (the deviation result) while the monitored value is kept private, thus leveraging on the ZK-Proof mechanisms to validate the proof of the private monitored value against the public ones.

- <u>Step 3:</u> The verifier key is used for generating a Verifier Smart Contract (see Figure: 5). This Verifier contract will be used to verify that indeed the energy deviation value made public and registered on chain is correct. This is possible by providing to the Verifier Contract the $Proof_{deviation}$ generated by the corresponding $PK_{flexibility}$ based on the energy monitored value, which is now kept private, and the time. The parameters required by the Verifier Contract are (see Figure: 5): (i) three elliptic curve points (a, b, c) of the $Proof_{deviation}$ that make up the zkSNARKs proof and (ii) $Proof_{deviation}$ inputs representing the public parameters specified by the $F_{flexibility}$ (i.e. zkProofInput [0] -> $F_{flexibility}$ input parameters: Hour, zkProofInput [1] -> $F_{flexibility}$ output parameters: Figure 4Deviation Value).



```
contract Verifier {
    using Pairing for *;

    struct VerifyingKey {
        Pairing.G1Point a;
        Pairing.G2Point b;
        Pairing.G2Point gamma;
        Pairing.G2Point delta;
        Pairing.G1Point[] gamma_abc;
    }

    struct Proof {
        Pairing.G1Point a;
        Pairing.G2Point b;
        Pairing.G1Point c;
    }

    event Verified(string s);

    function verifyingKey() pure internal returns (VerifyingKey memory vk) {▭}

    function verify(uint[] memory input, Proof memory proof) internal returns (uint) {▭}

    function verifyTx(
            uint[2] memory a,
            uint[2][2] memory b,
            uint[2] memory c,
            uint[2] memory zkProofInput
        ) public returns (bool r) {▭}
}
```

Figure: 5 Verifier smart contract using the Verifier Key generated

- <u>Steps 4 & 5:</u> The Verifier Smart Contract will be deployed on chain by the Aggregator. The Aggregator can now contact the prosumer to ask for his participation in the flexibility program. For this the Aggregator will specify: The Flexibility Request Profile, the Prover Key, $PK_{flexibility}$, to be used by the Prosumer, and the Verifier's Smart Contract address.

- <u>Step 6:</u> If the Prosumer accepts the Aggregator proposal, it will make this known by registering the Verifier Smart Contract provided by the Aggregator as a Validator in the Prosumer associated smart contract, thus giving permission to the Verifier Smart Contract to validate all the following activity to be registered on chain.

- Step 7: The Prosumer will send its acknowledgement to the Aggregator by specifying the Transaction Receipt proving the Verifier's Smart Contract registration as a Validator.
- Steps 8 & 9 & 10: In return, the Aggregator will deposit the flexibility reward for the entire period in the prosumer's DEP contract and notify the prosumer of its action. From now the prosumer is setup to register the proof of his activity on chain.

2.3.2 Real Time Deviation Registration

Once the setup and the Prosumer's registration with the Flexibility Request is completed, the Prosumer energy consumption profile will be monitored and the deviations from the flexibility request profile are assessed.

The deviation detection process is built upon the 2nd tier energy ledger presented in detailed D5.1 [1] overview in Section 2.2. In summary it: (i) stores off-chain, in a NoSQL database, the real-time data collected from energy metering devices while (ii) all the collected values are hash-linked back at edge level and periodically stored on-chain for validation enforcement and (iii) validation services are built upon the off-chain stored data resulting in a tamper-evident solution. The real-time energy data collected from the smart meters are processed with a compression function, supporting the validation in near real-time that indeed a prosumer has followed the precontracted profile (the profile precontracted from the Aggregator). The compression function considers the same granularity as the one used by the Aggregator flexibility request (one energy transaction value per hour).

Using zero knowledge proofs we will hide the registration of the compressed energy transaction value by computing the deviation off-chain and registering on the chain only the $Proof_{deviation}$ demonstrating that the computed energy deviation is indeed correct.

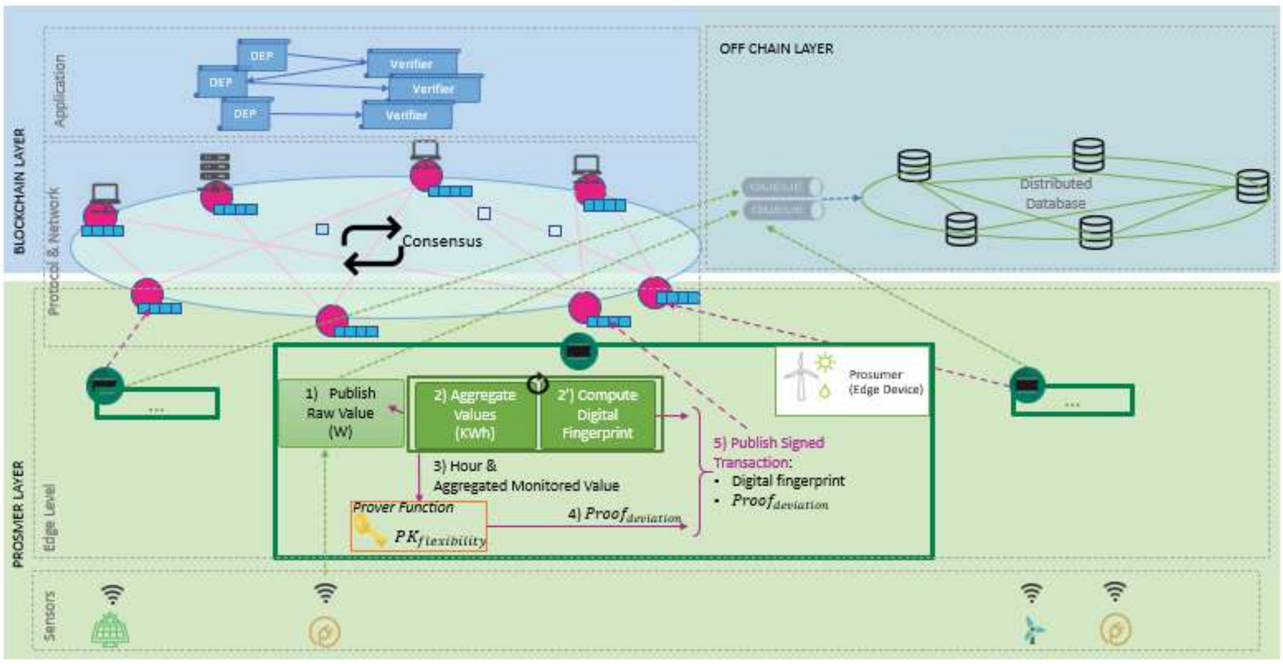
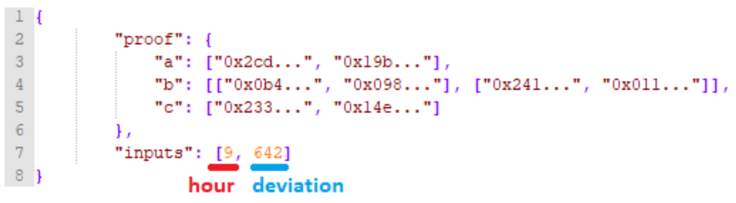


Figure 6: Energy deviations detection process augmented with zero knowledge proof for data privacy

The process of registering the energy deviation values from flexibility requests is depicted in Figure 6 and features the following steps:

- Step 1: The monitored energy value received from the smart energy meters is firstly directly forwarded to the Message Queues, that ensure a scalable registration of the monitored values in the distributed database cluster.
- Step 2: The monitored value is forwarded to the online hashing algorithm which periodically (at every hour) computes the average of monitored values together with the digital fingerprint of the values (obtained by hashed-linking back the real-time monitored values).
- Step 3: The average monitoring value will be provided together with the corresponding monitoring hour and the proving key $PK_{flexibility}$ to the Prover Function, obtaining the $Proof_{deviation}$, and the deviation will be registered (see Figure 7).



```
{
        "proof": {
            "a": ["0x2cd...", "0x19b..."],
            "b": [["0x0b4...", "0x098..."], ["0x241...", "0x011..."]],
            "c": ["0x233...", "0x14e..."]
        },
        "inputs": [9, 642]
}
```

Figure 7: SK Proof snippet for energy deviation registration

- Step 4: The edge node will sign a transaction intended for the corresponding prosumer associated contract, containing the digital fingerprint obtained and the $Proof_{deviation}$ together with the registered deviation value.

2.3.3 Smart Contract Interaction

Once reaching the chain, the published energy transaction will be sealed in a new block, triggering a state update as a result of the execution of the smart contract regulating the prosumer behaviour. After the energy transaction registration on chain, that also contains the $Proof_{deviation}$, the interaction with the Smart Contracts associated with the prosumer is described as depicted in Figure 8.

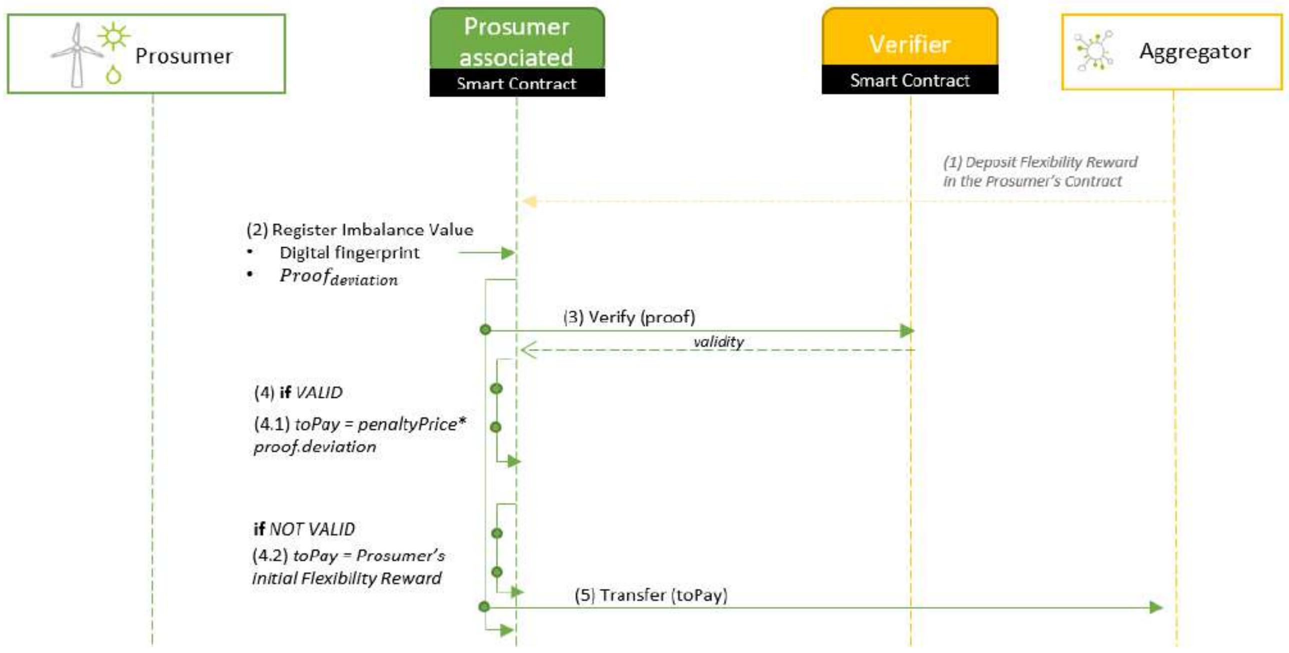


Figure 8: Privacy preserving interaction with Smart Contracts associated with the prosumer

The initials steps of the communication required the Prosumer contract to be aware of the Verifier Contract responsible to evaluate the real time prosumer activity. Furthermore, following the Prosumer's agreement to participate in the Flexibility Program, the Aggregator was responsible to fund the Prosumer contract with the reward owed (i.e. incentives) for the entire flexibility request. The Prosumer smart contract will act as an escrow for the reward received from the Aggregator. If the prosumer fails to register his/her progress, the Aggregator will be able to retrieve the funds. Otherwise, if the prosumer succeeds to register his/her progress, the funds will decrease only if deviations from the initial agreement are registered, and by the end of the Flexibility Program, the prosumer will have custody over the remaining funds.

Once a new transaction is published, the Prosumer smart contract will enforce the following rules upon the registered values:

- The DEP contract will redirect to the Verifier smart contract the proof registered by the transaction.
- If the proof turns to be valid then the Prosumer smart contract will compute the penalty proportionally to the deviation registered through the proof. If, however no deviation is registered, than the penalty will be set to value 0.
- If the proof is invalid (i.e. someone tampered with the proof, prior to signing the transaction which is possible only on the edge node, thus involving a malicious Prosumer trying to cheat the Flexibility Program), then the penalty computed by the Prosumer smart contract will consists of all funds received initially from the Aggregator. Such a drastic measure is required discouraging the potential malicious activity of prosumers while the Aggregator may use the recovered funds to enter in agreement with another Prosumer for providing the needed flexibility the remaining period.

Finally, the penalty computed is transferred to the Aggregator's address.

2.4 Implementation and Evaluation

The eDREAM platform prototype was enriched with the above-described zero knowledge proof solution for preserving the privacy of prosumers data. The prototype architecture and mapping of technologies are depicted in Figure 9.

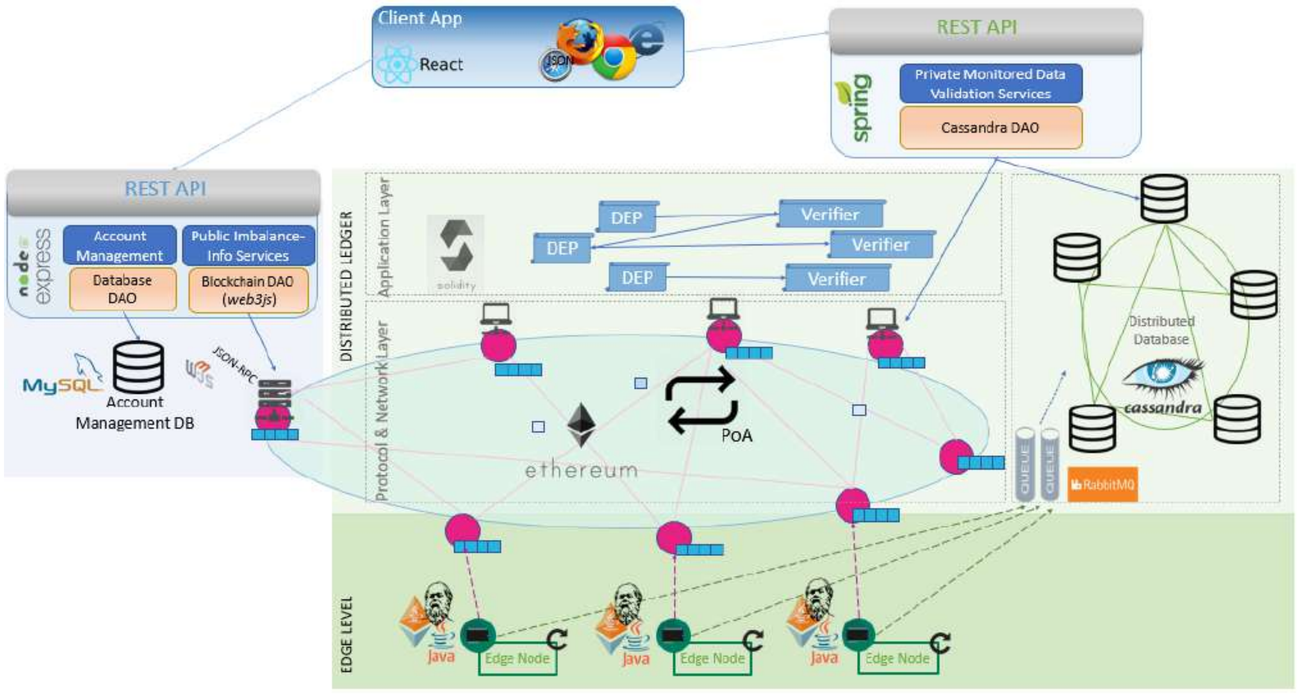


Figure 9: Prototype architecture and technologies mapping

The **Edge Layer** features prosumers associated energy monitoring applications that connect both to the RabbitMQ queues, that are responsible to feed the real-time data to the Cassandra DB for storage, and to the specified Ethereum Node (local or remote) for periodically storing the values on-chain and issuing ZK proofs over the deviations form flexibility request values registered, using the Zokrates tool.

Protocol & Network Layer is built on a custom Ethereum [6] blockchain network where the smart contracts managing the business enforcement of the flexibility services are deployed. The contracts are implemented using Solidity [7]. For the blockchain setup we used the Proof of Authority as consensus algorithm. To be able to compare our results with the public networks ones, respecting the block times recommendations we have considered the default mining time of 15 seconds and a gas limit comparable with the public networks of 8 000 000 gas per block.

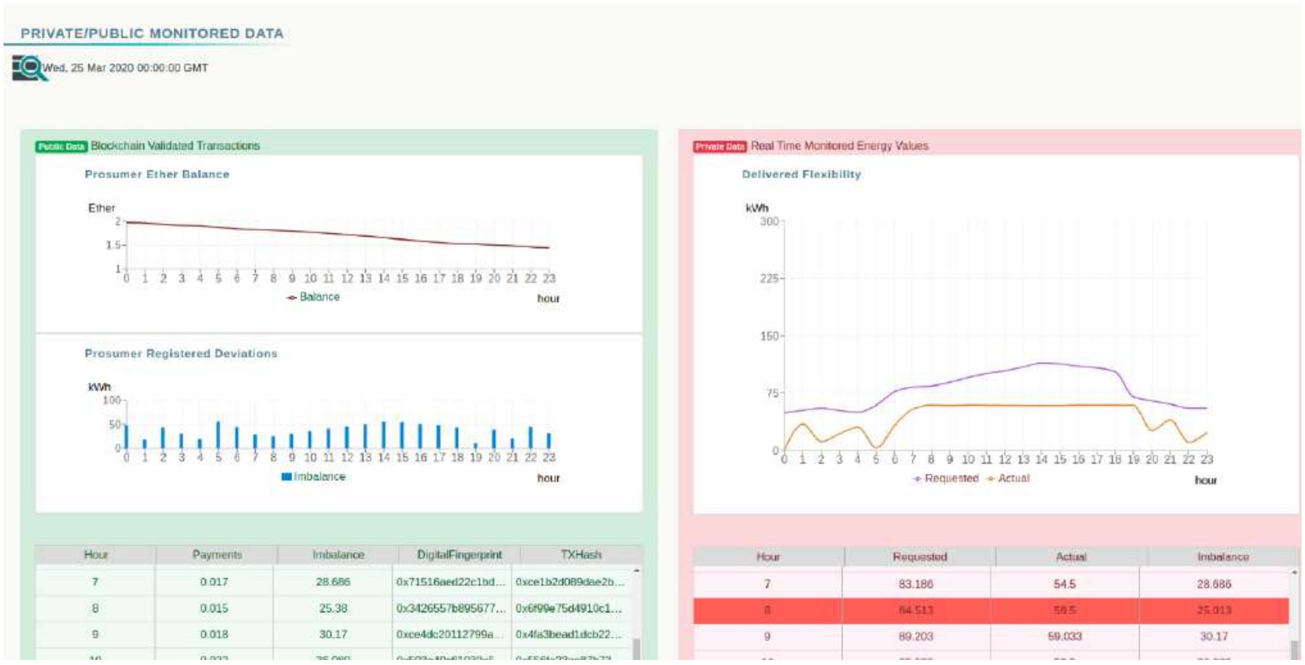
Application Layer features:

- *Server Application exposing REST API (Blockchain API)* acts as a proxy between the client applications and the blockchain platform. This architecture has been chosen to provide fast prototyping features, allowing the simulation of many prosumers, as opposed to fully decentralized client applications that would require deploying several client applications proportional to the number of prosumers registered. The main components of the server-side are the MySQL Database and the REST API. The MySQL Database represents the account management storage, having the goal of saving data about the registered prosumer accounts. The REST API is implemented using NodeJS [8], and web3js [9] exposing an API over the database and over the blockchain platform. The Database DAO manages access to the MySQL Accounts database and the Blockchain DAO manages the smart contracts method calls.
- *Server Application exposing REST API (Cassandra API)* acts as a proxy between the client applications and the Cassandra database. The application is used to expose the validation services required to annotate the monitored data (stored in the Cassandra DB), identifying the inconsistency found with respect to the already stored on-chain digital fingerprints by running the algorithms described in D5.1 [1].
- *React Client App* – the web application that allows the user to interact with the prototype and visualize the progress, and real-time flexibility and balance tracking. Moreover, the client application also connects to the validation services exposed as REST services from the Scaling Tier level.

The **Scaling Tier** contains a distributed Database and Message Queues (Cassandra DB [10] and RabbitMQ[11]) that provide the scaling-tier storage solution for the monitored energy values received from the household installed devices.

For evaluation we have considered a setup in which a Flexibility Request is issued by an Aggregator for a Prosumer over a 24-hour period. It is computed based on the historical data of the selected prosumer, considering the Prosumer flexibility availability and calculated baseline (see D3.5 [13] and D3.2 [14] for more details on how they are calculated).

In Figure 10 a snippet from the Prosumer client application is depicted. On the left side the page shows public data registered in Blockchain and, as a result, visible to all the network participants. There are two charts depicted. The first chart, “Prosumer Ether Balance” is showing how the balance of the logged-in Prosumer has changed during the day. The Prosumer started the day having the contract pre-filled with ether, as a result of his agreement to participate in the Flexibility Program. However, during the day, the balance is slowly decreasing, due to the deviations registered between the monitored energy values and the flexibility request (both are private information). In the second chart, the “Prosumer Registered Deviations”, the bars show the deviations as quantity of energy per hour registered in the Prosumer associated smart contract. The table below the two charts, depicts the information registered hourly in the Prosumer smart contract, and the energy transaction hash, offering the possibility to verify that indeed the transaction was successfully mined and validated by the network.

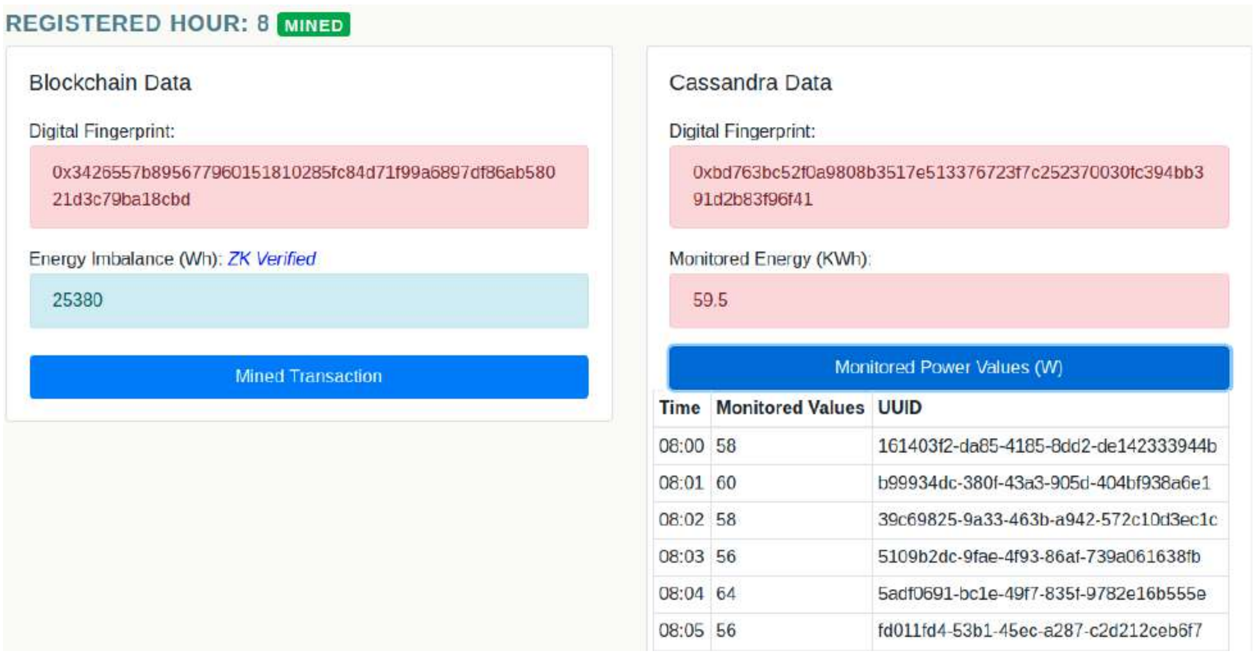


Hour	Payments	Imbalance	DigitalFingerprint	TXHash
7	0.017	28.686	0x71516aed22c1bd...	0xce1b2d089dae2b...
8	0.015	25.38	0x3426557b895677...	0x6f99e75d4910c1...
9	0.018	30.17	0xce4dc20112799a...	0x4fa3bead1dcb22...

Hour	Requested	Actual	Imbalance
7	83.186	54.5	28.686
8	84.513	59.5	25.013
9	89.203	59.033	30.17

Figure 10: Private and public energy data registered in the blockchain platform

On the right side of the page (see Figure 10), the Prosumer private information is presented. This information to be securely stored in the platform, is accessible only to the logged in owner Prosumers. The chart shows the details of the flexibility requested profile received from the Aggregator and the Prosumer actual energy consumption profile monitored in real time. In the table on the right, it can be observed that indeed the deviations obtained by subtracting the flexibility request and monitored values equal to the imbalance registered on the chain. However, the 8th hour row displays the average energy consumption for that hour and the colour red symbolizes that the some of the energy monitored values registered at smaller timestamp in Cassandra during that hour may have been tampered since their actual registration.


REGISTERED HOUR: 8 MINED

Blockchain Data

Digital Fingerprint:
0x3426557b895677960151810285fc84d71f99a6897df86ab580
21d3c79ba18cbd

Energy Imbalance (Wh): *ZK Verified*
25380

Cassandra Data

Digital Fingerprint:
0xbd763bc52f0a9808b3517e513376723f7c252370030fc394bb3
91d2b83f96f41

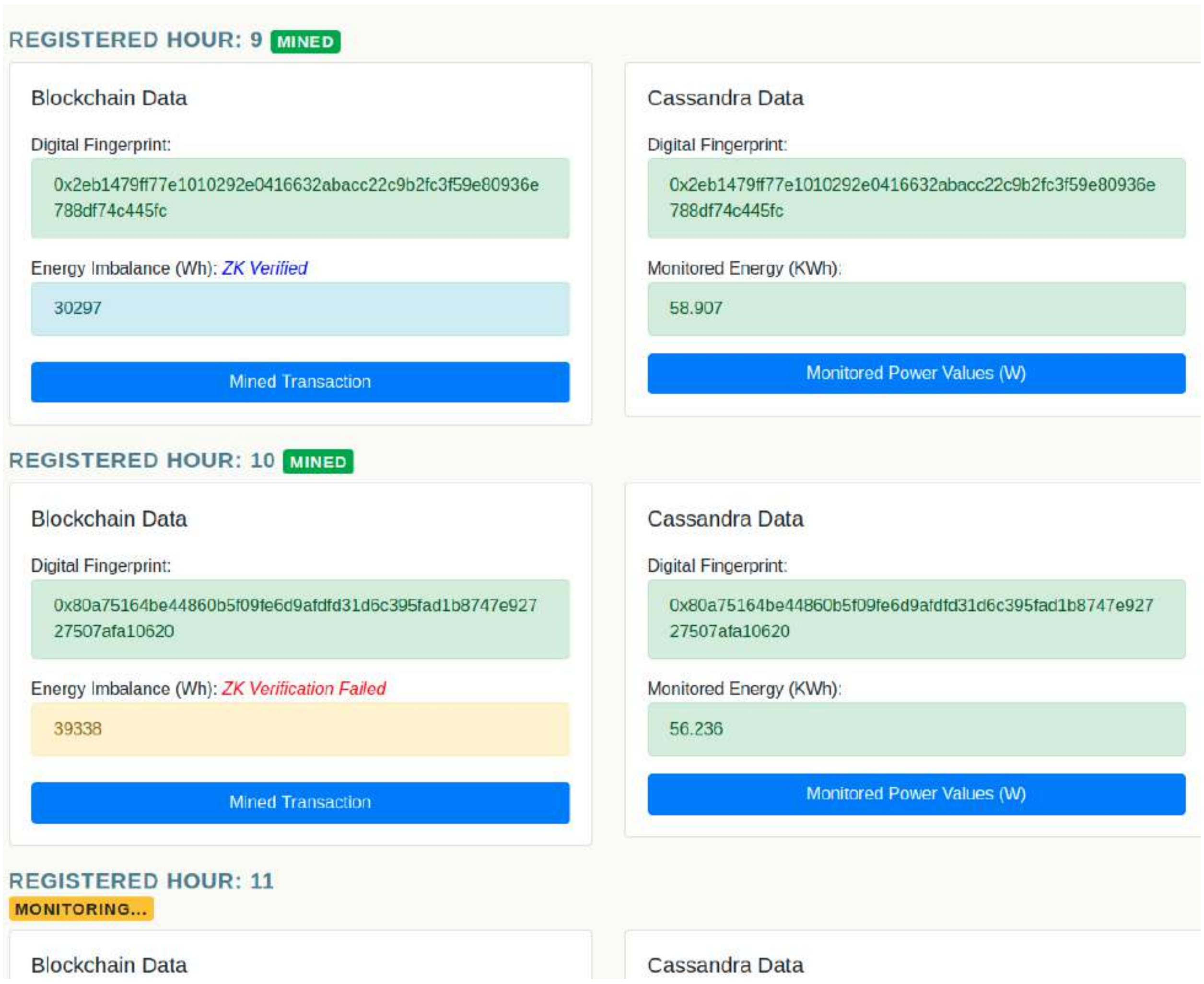
Monitored Energy (KWh):
59.5

Time	Monitored Values	UUID
08:00	58	161403f2-da85-4185-8dd2-de142333944b
08:01	60	b99934dc-380f-43a3-905d-404bf938a6e1
08:02	58	39c69825-9a33-463b-a942-572c10d3ec1c
08:03	56	5109b2dc-9fae-4f93-86af-739a061638fb
08:04	64	5adf0691-bc1e-49f7-835f-9782e16b555e
08:05	56	fd011fd4-53b1-45ec-a287-c2d212ceb6f7

Figure 11: Validation of energy monitored values privately stored

For a more accurate view of the situation, the validation services are invoked, and the results can be seen in Figure 11. All the sampled energy data values during the 8th hour are checked using the digital fingerprinting algorithms of the eDREAM 2nd tier energy data ledger. It is again marked, signalling that at least one of the monitored value from the real time stored values (i.e. values shown in the table below) is incorrect and had been tampered. As a result, the Prosumer is made aware of the fact that at least one of the monitored values from the table has been altered. However, the deviation value registered on chain which is done on every hour has successfully been validated using the ZK proof. Since the provided ZK Proof was computed at the edge node based on the deviation between the monitored value and flexibility requested on hourly basis, we can assume that once the ZK Proof was correctly validated on chain, the client and the aggregator have been correctly settled during the delivery of the flexibility profile (the settlement being done by the smart contract in a tamper proof manner)..

In Figure 12 however, one can see that at hour 10 an inconsistency is detected regarding the deviation value registered on chain. This means that, when the zk proof was registered in the Prosumer smart contract, the Validator Contract have signalled an inconsistency between the hour, the declared deviation and the 3 elliptic curve points, meaning that someone tampered with the proof at edge node level, prior to the energy transaction being signed and deployed on chain.


REGISTERED HOUR: 9 MINED

Blockchain Data

Digital Fingerprint:
0x2eb1479ff77e1010292e0416632abacc22c9b2fc3f59e80936e788df74c445fc

Energy Imbalance (Wh): *ZK Verified*
30297

Cassandra Data

Digital Fingerprint:
0x2eb1479ff77e1010292e0416632abacc22c9b2fc3f59e80936e788df74c445fc

Monitored Energy (KWh):
58.907

REGISTERED HOUR: 10 MINED

Blockchain Data

Digital Fingerprint:
0x80a75164be44860b5f09fe6d9afdfd31d6c395fad1b8747e92727507afa10620

Energy Imbalance (Wh): *ZK Verification Failed*
39338

Cassandra Data

Digital Fingerprint:
0x80a75164be44860b5f09fe6d9afdfd31d6c395fad1b8747e92727507afa10620

Monitored Energy (KWh):
56.236

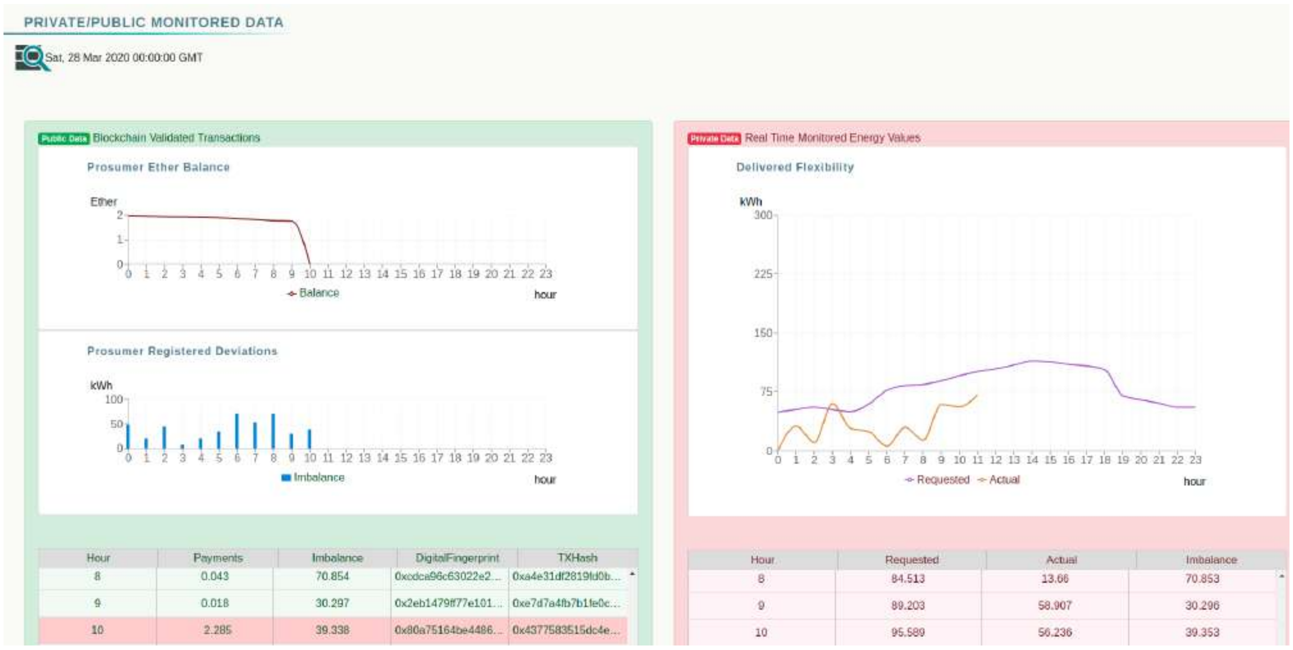
REGISTERED HOUR: 11
MONITORING...

Blockchain Data

Cassandra Data

Figure 12: Inconsistency detected in the energy deviation registered on chain using the ZK Proof

A detailed view of the payments and the activity on chain, is depicted in Figure 13. On the left side, the blockchain public information is depicted. The hour 10 is marked as being invalid according to the ZK Verifier key, and as a result in the “Prosumer Ether Balance” chart, it can be seen that the Prosumer lost all its funds, received from the Aggregator once with the its enrolment in the Flexibility Program. The funds have been returned to the Aggregator as a result of the Prosumer misbehaviour and attempt to cheat the network.


PRIVATE/PUBLIC MONITORED DATA
Sat, 28 Mar 2020 00:00:00 GMT
Public Data
Blockchain Validated Transactions
Prosumer Ether Balance
Ether
Balance
hour
Prosumer Registered Deviations
kWh
Imbalance
hour
Private Data
Real Time Monitored Energy Values
Delivered Flexibility
kWh
Requested
Actual
hour

Hour	Payments	Imbalance	DigitalFingerprint	TXHash
8	0.043	70.854	0xcdca96c63022e2...	0xa4e31df2819fd0b...
9	0.018	30.297	0x2eb1479ff77e101...	0xe7d7a4fb7b1fe0c...
10	2.285	39.338	0x80a75164be4486...	0x4377583515dc4e...

Hour	Requested	Actual	Imbalance
8	84.513	13.66	70.853
9	89.203	58.907	30.296
10	95.589	56.236	39.353

Figure 13: ZK Proof inconsistency detected in stored energy deviation is penalized

3 Bitcoin Integration for Improved Trustworthiness

In section 2, we already discussed how it is possible to ensure the energy data privacy in eDREAM. More generally, although private blockchains offer significant advantages in terms of privacy, latency, and energy consumption, in contexts where reliability, transparency, and security are critical requirements it could be useful to leverage on the advantages provided by public blockchains. Public blockchains, not depending or needing to trust any central authority, are designed to provide permanent decentralized proofs which can't be erased or modified by third parties.

Since we're taking into account the security of a network, it could be worth to try to quantify the amount of security provided by a public blockchain, in comparison with another one.

In cryptography, computational security is a way to quantify the security of a crypto algorithm. Computational security views a system as *secure* if it cannot be broken within a reasonable amount of time, and with a reasonable amount of resources such as memory, hardware, budget, or energy [14]. For a blockchain, this could be reduced to calculating the cost of carrying out a 51% attack.

Proof of Work (PoW) blockchains, in fact, are typically secured by nodes set up to recognize, in the event of a *fork* in transaction history, the blockchain with the most blocks as the correct version of history. Miners controlling more than 50% of the network hashing power can mine silently a forked copy of the blockchain and, every time they send funds to one address to the main chain, sending funds to an address they control to the forked chain. Since other nodes only know about the main chain, the transactions continue to be seen as valid and accepted. The attacker can later release the silently mined blocks that will be accepted by the other nodes since they will form the longest chain (due to the higher hashing power), returning in possess of the funding already spent. This is known as a 'double spend' attack.

The biggest blockchains have sufficient mining capacity supporting them, making it extremely expensive to carry on a 51% attack, and it is also possible to try calculating the cost of a hypothetical 51% attack against them in an empirical way.

As an example, the open source project Crypto51 [15] tries to calculate this cost putting the current network hash rates and the coin prices for the most popular blockchains together with the cost needed to *rent* the needed hash power on the popular NiceHash [16] platform.

At the time we're writing, the most secure public blockchain according to this metrics is the Bitcoin public blockchain, against which the cost for 1h attack is quantified at $633,556 and the network hash rate is quantified at 127,422 PH/s. For comparison, the cost of 1 hour of attack against Ethereum public network results in $87,145 and the network hash rate is 175 TH/s.

Of course, blockchains cannot be compared considering only one specific metric since different blockchains have different characteristics. For example, Ethereum has introduced the possibility to use smart contract developed with a Touring-complete language and for this reason it allows a flexibility not possible on other blockchains but, in this section, we will describe how it is possible to implement a second level solution for secure data storage using Bitcoin's blockchain extending our Ethereum-based solution thanks to Hierarchical-Deterministic wallets and the OP_RETURN script code, both provided by the Bitcoin blockchain.

3.1 Hierarchical Deterministic Addresses

HD (Hierarchical-Deterministic) wallets are defined as wallets which *can be shared partially or entirely with different systems* in Bitcoin Improvement Proposal (BIP) 32 [17].

Deterministic wallets, permit to calculate the public keys without revealing the private keys, allowing to implement use cases in which software platforms generate fresh addresses (public key hashes) for each transaction, without sharing the corresponding private keys, which can be used to spend the received funds.

Being basically a *single* chain of keypairs, deterministic wallets cannot be partially shared. Hierarchical Deterministic wallets allow to selectively share addresses by supporting *multiple* keypair chains, all derived from the same root key.

This feature enables the implementation of complex use cases in which a third party is able to manage an address used to receive payments on behalf of a user, without the need for granting access to the keys used by the user to spend money.

Extended Keys

An extended key is derived using a parent key and a *chain code*, a 32 bytes long extension, so as not to depend entirely on the parent key.

Assuming the public key cryptography used in Bitcoin [18], we define point(p) as a conversion function returning the coordinate pair resulting from elliptic curve point multiplication of the base point with the integer p so that an extended private key is represented as (k, c), where k is the parent private key and c the chain code, and an extended public key is represented as (K, c), where K = point(k) and c is the chain code.

Each extended key has 2^{32} child keys, each with its own index. The first 2^{31} child keys, with indexes from 0 to 2^{31} - 1, are defined as *normal* child keys while the keys with indexes from 2^{31} to 2^{32} – 1 are defined as *hardened* keys [17].

Given a parent extended key, it is possible to derivate the child key depending on the key type (public or private) and its index (defining if the key is a hardened key or not). Table 3 summarizes the possible combinations and outputs for the Child Key Derivation (CKD) functions.

Table 3: Child key derivation functions

Parent key	Child Key	Description
Private	Private	Computes a child extended private key from the parent extended private key
Public	Public	Computes a child extended public key from the parent extended public key. It is only defined for non-hardened child keys.
Private	Public	Computes the extended public key corresponding to an extended private key. For non-hardened keys, child public keys of a given parent key can be derived without knowing any private key.
Public	Private	This is not possible

Combining together several CKD functions, it is possible to build a *tree* like the one represented in Figure: 14.

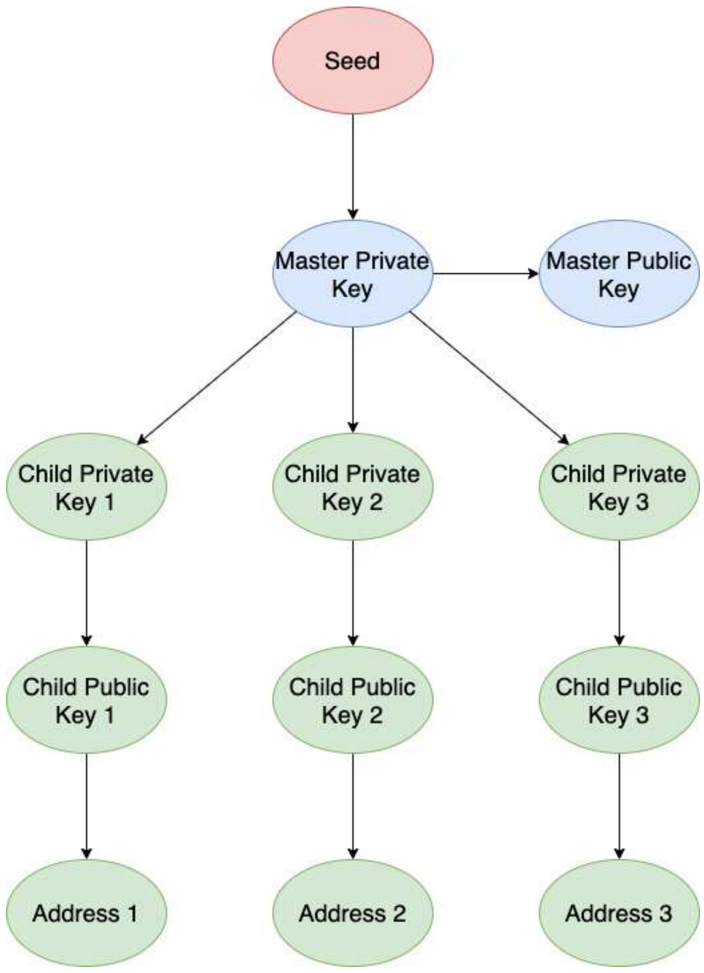


Figure: 14 HD wallets

From the master node *m* it is possible to derivate several CKD(m, i) for different values of *i*, obtaining a set of derived nodes. The standard notation for this is written as m/a/b/c, equivalent to CKD(CKD(CKD(m, a), b), c) [17].

Thanks to this design, an extended private key allows reconstruction of all descendant private and public keys, and knowing an extended public key allows reconstruction of all descendant (non-hardened) public keys.

Wallet support

A wallet supporting HD keys is organized as several accounts and each account is composed of two key chains: a public one and a private one. This structure enables the implementation of different use cases, some of which are proposed in BIP 32 specification itself, including: full wallet sharing, “read only” access to external auditors, per-office balances inside a single organisation, supporting for recurrent B2B transactions, supporting for unsecure third-party money receivers [17].

3.2 OP_RETURN

Unlike Ethereum-like blockchains, Bitcoin blocks are limited in size. From the early phases of Bitcoin development, the block size limit was 1 megabyte. In 2017, with the adoption of the Segregated Witness, or

SegWit, protocol (BIP 141 [19]), the block *size* limit was replaced by a block *weight* limit of 4 million weight "*units*" calculated differently for each kind of transaction included and resulting in a theoretical maximum block size of 4 megabytes.

Nevertheless, since its inception, users tried to store any kind of data on the Bitcoin blockchain [20]. At the beginning, this was done using more or less sophisticated "hacks" and exploiting some attributes of blockchain transactions in an unplanned way. Some of the ways used to store custom data are editing the *nonce* field, using the *coinbase* string, generating "fake" addresses, and using transaction scripts, each with its own characteristics and disadvantages. The nonce header can take only 4 bytes of data for example, to use the coinbase string the user has to mine its own block to encode the string, and using the transaction outputs to store data has a big collective disadvantage for the whole blockchain of bloating the Unspent Transaction Outputs (UTXO) database creating fake entries. Due to security concerns, valid *standard transactions* can only be composed of a limited set of functions [21] and, since miners only relay standard transactions, custom scripts have never caught on as a viable solution.

For this reason, since its version 0.9.0, Bitcoin reference implementation supports the storage of data using the OP_RETURN code [22]. OP_RETURN is a transaction script function (or *opcode*) meant to mark a transaction as invalid [21]. This function accepts as payload up to 40 bytes.

Since OP_RETURN outputs are provably unspendable [23], they are discarded from storage in the UTXO set although stored in the block chain, reducing the cost to the network compared with other data storage techniques which burden the UTXO database. The 40 bytes limit, while too restricted for data storage, has the capacity to encode a hash value representing any digital document.

3.3 Notarization on public networks

One of the main reasons for using public blockchains as a data storage solution is notarization. The characteristics of a decentralized, distributed ledger in fact enables data timestamping, data integrity verification, and proof of data ownership.

Generic document notarization services built on top of Bitcoin already exist [24][25]. Such services offer the capability to securely store online *proof of existences* for different kind of documents. Once the document is uploaded, its unique *hash* is determined and included as payload in a blockchain transaction. Once the transaction is included in a mined block, the block time serves as a temporal anchor, proving that the document existed at that time. Since it is not necessary to share the document itself, there is no need to store the original data or to share it with others, making it possible to publicly prove the knowledge of certain information without publicly revealing the data.

These services usually feature an easy graphical user interface, guiding the user through a process that can be broken down into four main steps during which the document is selected, the "anchor" transaction is funded sending an amount of bitcoin to a specific address, the transaction with the document hash is permanently included in a block, the proof can be verified by checking the transaction using any *block explorer* (a tool showing information about blocks, transactions, and addresses of a specific blockchain).While the user interface hides the complexity of the process, it is possible to determine what happens behind the scenes in each step (Figure 15).

Once the document is selected, its cryptographic hash is calculated and uploaded. Hashing the document before the upload grants that the actual document is not shared as it never leaves the user's client. For each user request, identified by the document hash, a unique bitcoin address is derived from the service *parent* *key*. The user is then prompted to fund the blockchain transaction. Usually, the requested amount is determined taking into account the miners fee needed to propagate the transaction and the notary service provider's share. By using a different address for each document, the service is able to keep track of the advancement of any specific request. Once a payment is received, the related transaction output is then "consumed" to start the actual notarization. This happens by creating a new transaction taking as input the previous transaction UTXO and producing as output *i)* the transfer of the funds to the service provider main account, *ii)* the OP_RETURN function including as payload the document hash. The transaction is then signed using a private key derived from the parent key and broadcasted. Finally, once mined, the transaction is publicly visible in the blockchain along with its outputs. The user is notified with the transaction address.

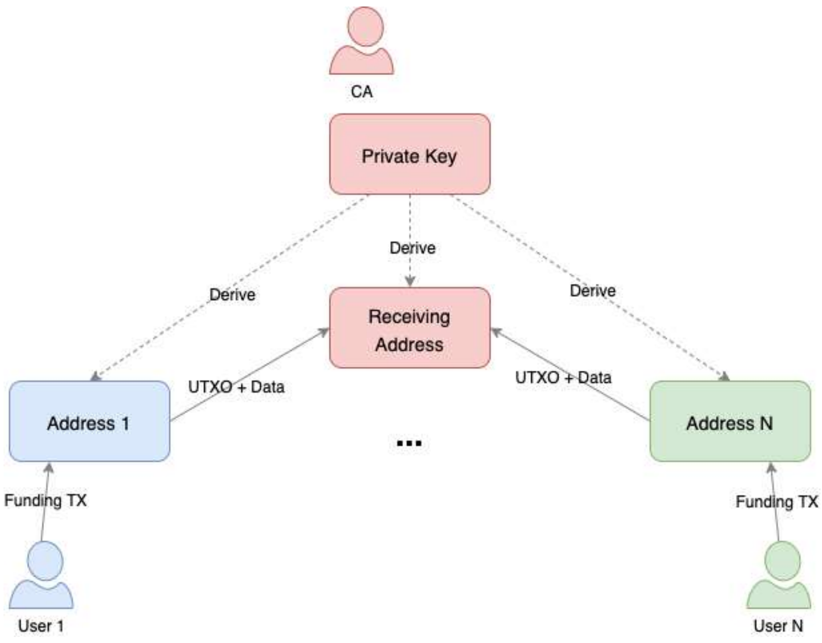


Figure 15: derived addresses and transactions for notarization

When the process is complete, the validation can be performed: the document hash is calculated and uploaded. The server keeps track of the requests already processed and so is able to provide the blockchain transaction address containing the matching payload, proving that the document was already known when the block was mined. The user, in turn, can demonstrate to be the original owner by proving that he is the sender of the funding transaction.

Keeping track of requests already processed is convenient but not necessary since the transaction address is the only data strictly necessary to validate the process. This makes the process secure even in case of data loss by the service provider, and not storing the document data makes the service secure from possible data breaches.

As a final note, although here we refer for convenience to the hash of the document, the transaction can contain as payload any string up to 40 bytes so many providers use strings composed by a specific prefix followed by the hash of the document and encoded in different formats.

3.4 Energy Data Notarization on public networks

The general process illustrated in section 3.3 can be simplified and adapted to the specific case of the off-chain storage of energy data. The first assumption is that the *certifying entity*, differently from the documents' notarization, has the need to process recurring data from the same sources rather than single transactions from many different users.

In this case, it might be worth considering the usage of a dedicated node for this purpose. The node would not need the computational capacity to *mine* new blocks by itself, but it should just have the capacity to sign and broadcast transactions over the network. Being a dedicated node instead of a third-party service provider, there is no need for a service fee either.

Finally, although for an individual user address reuse is considered a poor choice for privacy [26], in this case our node represents a well-known entity (being a certifying entity) sharing aggregated information so reusing the same set of addresses (Figure 16) results in reasonably safe operation from this point of view. Under these assumptions, the extension for public ledger without Turing-complete smart contract has been designed and further described in Figure 16.

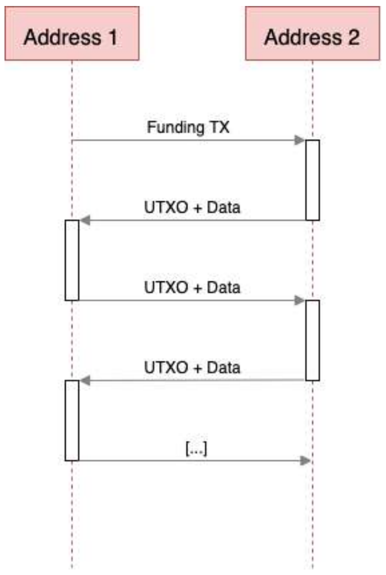


Figure 16: Data notarization using the same set of addresses

The system is first initialized: two addresses are derived from the same parent key and the first address is funded with an initial transaction.

1. The monitored energy value received from the smart energy meters is forwarded to the message queues, ensuring a scalable registration of the monitored values in the distributed database cluster.
2. The monitored values are periodically (hourly) aggregated computing their average over the time window and chaining them together with the corresponding monitoring hour. The hash for the resulting string is generated.
3. A new transaction is created, including:
    a. Input: UTXOs from previous transactions
    b. Output 1: transfer total amount from the sending address to the receiving one
    c. Output 2: OP_RETURN including the previously generated hash
    d. Miner Fee: auto-determined
4. The transaction is signed and broadcasted. Once the transaction is included in a block, it becomes verifiable.

5. Sending and receiving addresses are swapped, the process is repeated periodically.

4 Conclusions

In this deliverable, we have presented the evolution of the blockchain-based 2nd layer storage solution for energy data. The document covers the new features designed and developed on top of the first version of the eDREAM blockchain platform for secure and distributed energy data storage, already described in deliverable D5.1 [1], which forms the foundation of the blockchain-based platform for the distributed control and management of energy micro-grids, on top of which the smart-contracts based marketplaces (defined in deliverables D5.2 [27] and D5.5 [28]) and the validation and settlement process (defined in deliverable D5.3 [29]) are built.

The main features added, concern the application of Zero-knowledge proofs to improve the privacy of the eDREAM data storage solution and the extension of the storage solution for using it with Bitcoin-like public blockchains, without Turing-complete smart contracts support. The new functionalities increase the overall flexibility of the blockchain platform, completing the work started with the first version of the platform and allowing its application in accordance with the privacy needs of users and the exploration of new use cases where greater security or greater transparency is required.

References

[1] eDREAM D5.1 – eDREAM deliverable D5.1 "Blockchain platform for secure and distributed management of DR programs V1", May 2019.

[2] Miers, Ian, et al. "Zerocoin: Anonymous distributed e-cash from bitcoin." 2013 IEEE Symposium on Security and Privacy. IEEE, 2013.

[3] Sasson, Eli Ben, et al. "Zerocash: Decentralized anonymous payments from bitcoin." 2014 IEEE Symposium on Security and Privacy. IEEE, 2014.

[4] Ben-Sasson, E., Chiesa, A., Tromer, E., & Virza, M. (2014). Succinct non-interactive zero knowledge for a von Neumann architecture. In 23rd {USENIX} Security Symposium ({USENIX} Security 14) (pp. 781-796).

[5] Eberhardt, Jacob, and Stefan Tai. "Zokrates-scalable privacy-preserving off-chain computations." 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). IEEE, 2018.

[6] Wood, Gavin. "Ethereum: A secure decentralised generalised transaction ledger." Ethereum project yellow paper 151.2014 (2014): 1-32. Available online at http://gavwood.com/paper.pdf, Last accessed on April 2020

[7] Solidity. Available online at https://solidity.readthedocs.io/en/latest/ (accessed on April 2020).

[8] NodeJS, Available online at https://nodejs.org/en/ (accessed on April 2020).

[9] Ethereum JavaScript API, Available online at https://web3js.readthedocs.io (accessed on April 2020).

[10] Lakshman, Avinash, and Prashant Malik. "Cassandra: a decentralized structured storage system." ACM SIGOPS Operating Systems Review 44.2 (2010): 35-40.

[11] RabbitMQ. Available online: https://www.rabbitmq.com/ (accessed on April 2020).

[12] Jean-Philippe Aumasson, "Serious Cryptography, A Practical Introduction to Modern Encryption"**,** November 2017, 312 pp. ISBN-13: 978-1-59327-826-7

[13] eDREAM D3.5 – eDREAM deliverable D3.5 "Electricity production / consumption forecasting techniques and tool V2", February 2020.

[14] eDREAM D3.2 – eDREAM deliverable D3.2 "Recommendations for baseline load calculations in DR programs V1", April 2019.

[15] Crypto51. Available online at: https://www.crypto51.app/about.html (accessed on April 2020).

[16] NiceHash. Available online at: https://www.nicehash.com/ (accessed on April 2020).

[17] BIP 0032 on GitHub. Available online at: https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki (accessed on April 2020).

[18] SEC 2: Recommended Elliptic Curve Domain Parameters, Version 2.0, January 2010. Available from: http://www.secg.org/sec2-v2.pdf

[19] BIP 0141. Available online at: https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki (accessed on April 2020).

[20] "Hidden surprises in the Bitcoin blockchain and how they are stored: Nelson Mandela, Wikileaks, photos, and Python software". Available from: http://www.righto.com/2014/02/ascii-bernanke-wikileaks-photographs.html.

[21] Script on bitcoin.it wiki. Available online at: https://en.bitcoin.it/wiki/Script#Words (accessed on April 2020).

[22] Bitcoin Core version 0.9.0 release notes. Available online at: https://bitcoin.org/en/release/v0.9.0#opreturn-and-data-in-the-block-chain (accessed on April 2020).

[23] BIP 0074. Available online at: https://en.bitcoin.it/wiki/BIP_0074 (accessed on May 2020).

[24] Proof of Existence. Available online at: https://proofofexistence.com (accessed on April 2020).

[25] Notarify. Available online at: https://www.notarify.it/ (accessed on April 2020).

[26] Address reuse on bitcoin wiki. Available online at: https://en.bitcoin.it/wiki/Address_reuse (accessed on April 2020).

[27] eDREAM D5.2 – eDREAM deliverable D5.2 "Self-enforcing smart contract for DR tracking and control V1", May 2019.

[28]eDREAM D5.5 – eDREAM deliverable D5.5 "Self-enforcing smart contract for DR tracking and control V2", May 2020.

[29] eDREAM D5.3 – eDREAM deliverable D5.3 "Consensus based techniques for DR validation and financial settlement", April 2020.