2019 IEEE International Conference on Blockchain (Blockchain)

A Decentralized Service-Platform Towards Cross-Domain Entitlement Handling

Alexander Rech
Graz University of Technology
Institute of Technical Informatics
Graz, Austria
rech@tugraz.at

Christian Steger
Graz University of Technology
Institute of Technical Informatics
Graz, Austria
steger@tugraz.at

Markus Pistauer
CISC Semiconductor GmbH
Klagenfurt, Austria
pistauer@cisc.at

***Abstract*—As our world is becoming increasingly connected, secure and seamless cross-domain access to heterogeneous services is becoming more and more important. However, cooperation between client-server based systems in the sales domain is often difficult to achieve due to a lack of standardization as well as limited trust between service providers. Furthermore, the demand for anonymization techniques is becoming more important due to tighter regulations and increased privacy awareness of users. This is especially relevant when data needs to be shared by several parties. In this paper we describe a software service platform designed to increase collaboration between different cross-domain services. The general idea is to provide a common trusted layer for existing client-server systems that can be used for anonymizing and sharing services in the form of digital entitlements across independent service providers, their systems, and users, while ensuring secure authentication and authorization. Furthermore, we leverage the tamper-resistant properties of the blockchain and smart contracts in order to protect sensitive transactions against fraudulent manipulation and to introduce service-based agreements to the overall system.**

***Keywords*-Connected Services; Blockchain; Access Control; Security; Privacy;**

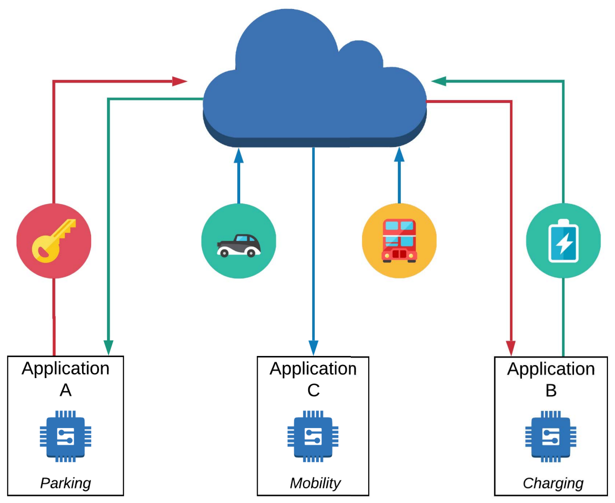


Figure 1. Digital entitlements can be exchanged between independent applications: inter-app and single-app cross-domain service exchanges are feasible.

I. Introduction

Today's service redemption systems are often designed as client server applications that are tailored to specific application scenarios. Due to rigid system boundaries, time and cost consuming integration procedures, and the lack of trust between different service providers there is often no intention of providing the possibility to share services across different domains and their systems. Additionally, public awareness of data privacy has risen significantly over the past few years and the value of data ownership has become a matter of intense public awareness and of ever greater importance to individual citizens. We want to increase the scope of businesses regarding their services and users and propose a platform for managing cross-domain services. The overall platform can be used for two scenarios: an inter-app approach, where services in form of digital entitlements are exchanged between independent platforms, and a single-app approach, where heterogeneous services are integrated into one application. Fig. 1 illustrates our idea. On the one hand, a parking application focused on issuing parking permits and a charging app that by contrast is responsible for authorizing the user to charge his vehicle may cooperate with each other. In this sense, when a costumer redeems a specific service from one service provider, additional offers from another independent service provider could be unlocked as a reward, redeemable at the corresponding redemption unit (parking gate, charging station). On the other hand, a route planer app, for instance, may suggest several possibilities to reach the user's destination by integrating and combining different mobility services (e-bikes, car-sharing, public transport, etc.). These approaches would not only benefit the customer and result in a higher user satisfaction, they would also yield benefits for participating vendors in terms of a larger advertising scope and additional sales. From a technical standpoint, the platform that is described in this paper provides secure authentication, authorization, and accounting for applications of different kinds. In order to incentivize a cooperation between different service providers, trustability shall be further enhanced by leveraging blockchain and smart contract methodologies enabling transaction security, traceability, and data protection against fraudulent changes.

978-1-7281-4693-5/19/$31.00 ©2019 IEEE
DOI 10.1109/Blockchain.2019.00069

A blockchain is a peer-to-peer network architecture; a distributed database fully shared across multiple entities that does not have a central authority to manage it. All database entries are synchronized across all network participants, reducing the risk of single point of failures. Data is collected into blocks which in turn are linked, starting from a predefined genesis block which defines the properties and the underlying consensus mechanism of the blockchain. Consensus on the validity of blockchain transactions is achieved with a dedicated consensus algorithm, executed by a set of network authorities that is designed in such a way that acting against the rules does not pay off. Depending on whether the majority votes in favor of accepting a new block containing multiple transactions and datasets or not, it is added to the blockchain or rejected. Blockchains can be either public, where transactions are verified by an independent group of network participants, or private, where only a set of authorized individuals can process new transactions. A mixture of both would be federated blockchains. They do not allow any person to participate in the process of reading, writing, or verifying without an explicit invitation. This concept behind federated blockchains is particularly interesting in business applications in which the participants, such as different companies, may identify, but not fully trust each other. In order to specify additional rules on how different companies, aka service providers, interact with each other, smart contracts can be used. A smart contract is a computer program running on top of a blockchain, consisting of inputs, deterministic outputs, and trigger-conditions. They are able to process and store arbitrary datastructures. When a contract is deployed, it is assigned a random address that is used in future transactions as a reference to invoke the contract's functions. It stores its results as transactions in the distributed ledger. They cannot be deleted or altered by fraudulent entities due to the tamper-resistant design of the blockchain (cryptographic primitives such as hash functions, asymmetric cryptographic signatures, etc) [1], [2].

By leveraging these blockchain and smart contract properties we are presenting a platform for enabling tamper-resistant service exchange between vendors. Due to this collaboration, services from one service provider can be made accessible to another independent service provider, who would be able to obtain and forward them to his customers on demand or in an automated way, as soon as certain prerequisites are fulfilled. Transactions including issuance, obtainment, and sharing of cross-domain services are uploaded to the blockchain via smart contracts. We believe our approach to be very general and applicable to a wide range of use cases, paving the way for a new level of interconnectivity between different cross-domain services.

II. Related Work

The following papers build their approaches on the Ethereum blockchain. Ethereum is a virtual computer, aiming to be a decentralized world computer by offering smart contract capability. The network is composed of miners, full nodes, and light nodes. Light nodes rely on full nodes for security and can validate states by downloading and verifying block headers. Full nodes comprise the whole blockchain database. A subset of these validate all blocks and execute all contracts. Users submitting transactions to the Ethereum blockchain hold a private/public keypair. The public key is used to generate a blockchain address that serves as a public user identifier as well as an identifier for deployed smart contracts. Ethereum comes with different consensus protocols [3]. The most common are Proof of Work (Etash), where miners put work into solving a puzzle, and Proof of Stake (Casper) where stakeholders bet on whether a certain block is added or not. In contrast, in Proof of Authority (Clique) a set of approved authorities decides whether a transaction is valid or not.

Independent of the actual application area or use case, many blockchain approaches focus on combining non-blockchain access-permission systems with the distributed ledger technology for increased security. One approach couples blockchain and off-blockchain storage to construct a personal data management platform in order to control access permissions to private data collected by a service. The collected data is encrypted and sent to an off-blockchain key-value store. Only a hash is retained, acting as pointer to the data on the blockchain. The data can be queried by the user or the service using the key associated with it. Subsequently, the blockchain verifies if the digital signature belongs to either the user or the service. Every time a users subscribes to a service a new transaction specifies the access permissions and another contains the hash of the data [4]. Another approach focuses on the management of electronic medical records of patients distributed across several data providers. The authors' blockchain implementation gives patients an immutable log and easy access to their medical information across providers and treatment sites. In order to guarantee that the data is not altered, hashed data pointers are stored on the blockchain with an additional query string which is intended to be executed on the corresponding server of the data providers [5]. Data assurance and resilience are also crucial security requirements in cloud-based IoT applications. The authors of this paper [6] describe a procedure for secure drone communication, focusing on the integrity of the collected data. Again, the blockchain is used together with a traditional cloud-based system. Instead of registering the drone to the blockchain, they anchor the hashed data records collected from drones to the blockchain network and generate a blockchain receipt for each record stored in the cloud. Another IoT-blockchain approach focuses on effective and trusted data distribution across several devices. All blockchain-related operations are forwarded to a gateway, which is responsible for managing access control and endpoint authentication. Devices may act as information providers and also as consumers, identified by a globally unique identifier. Every device that generates an information item may forward its hash to the blockchain. The idea of this solution is that IoT service providers store access control policies that protect sensitive data in access control providers (ACPs) and in return ACPs generate secret keys (using a hash with the device's locally available identifier as input). The path to the access control policies (e.g. URL) is known by the devices [7]. Many of today's blockchain-based prototypes, like the projects mentioned above, have one thing in common: the hybrid approach between a traditional cloud environment and blockchain layer, where hashes are pushed to the blockchain for verifying the actual data stored inside a traditional database. By doing so, the advantages of both methodologies can be combined, such as speed on the one hand, and enhanced data protection on the other hand.

Other approaches mentioned in the following paragraph discuss design choices and limitations especially for permissioned blockchain types, such as hard coded consensus algorithms, mainly utilized for industrial applications [8]. Hyperledger Fabric, for instance, is an open-source project. It is a general-purpose permissioned blockchain system which also can be seen as a distributed operating system for permissioned blockchains [9]. The authors of another paper propose an architecture based on satellite chains that are able to run different consensus protocols in parallel, making governance models more easily adaptable. Nodes join a specific satellite chain if they want to interact with other particular nodes. Each satellite chain maintains its own private ledger and prevents any node that is not part of the satellite chain from receiving or accessing transactions [10].

More and more cities are adopting smart city concepts to enhance the living quality of their citizens and optimize the resources of cities regarding healthcare, transportation, energy, and education, for instance. The paper [11] describes how service-based-middleware for cloud and fog computing may enhance smart city technologies. In this sense, applications running in specific spots in cities can utilize locally available edge-computers, end-user devices, or other nearby edge devices to access different city services and to facilitate cooperation between diverse systems. Another approach attempts to decrease the heterogeneity of pre-existing systems, by concatenating their services and user pools on application level in a privacy preserving way [12]. In fact, data privacy is becoming more and more important due both to an increasing public privacy awareness and to increasingly stringent data protection rules. For example, in 2016, the European Union passed the General Data Protection Regulations (GDPR) [13]. Research is also being done on privacy preserving data management techniques. New concepts are being elaborated that use clustering algorithms as a pre-process to further improve the diversity of anonymized data [14].

III. Design

Our proposed solution, which is partly based on the design of [12], is a software platform for secure authentication, authorization and accounting of heterogeneous services in the form of digital entitlements for client-server redemption systems. It is a central place for service management, with additional decentralized control instances responsible for the integrity and availability of transaction-based data.

A. Requirements

Privacy preserving identification and authentication. When data, especially user-related data needs to be passed between different entities, privacy aspects become more important. Therefore, only tokenized user data should be exchanged and the link between real and anonymized data should be revocable. Additionally, each device needs to be authenticated when communicating to the federated platform.

Service authorisation management. A device has always to be authorized to use or redeem a specific service, which should be represented as a generic entitlement object. These entitlements should be assignable to devices, be revocable at any time, and be redeemable by dedicated control units.

Service co-modality. The interexchange of services between independent service providers shall be improved by providing a federated service layer where services and agreements between heterogeneous service providers can be defined, obtained, and subsequently forwarded to users and their devices.

Data integrity. All components interacting with each other should rely on PKI mechanisms. In this sense, certificates as well as all tokenized data should be signed by a trusted authority and be verifiable against fraudulent changes. Additionally, the integrity of agreements and transactions shall be further protected by leveraging blockchain and smart contracts capabilities.

Decentralized consortium consensus. A decentralized consensus algorithm shall be used to decrease the chance of misuse of the network by a malicious party. By relying on a consortium consensus approach additional advantages in terms of speed, network scope, and authorization handling should be achieved compared to completely public approaches.

Data availability. The risk of data loss should be drastically minimized, due to the distributed character of the blockchain. In this sense, each node shall maintain a local copy of the ledger. Furthermore, the framework should ensure that service providers control their transactions and have transparency over all agreements they are involved in.

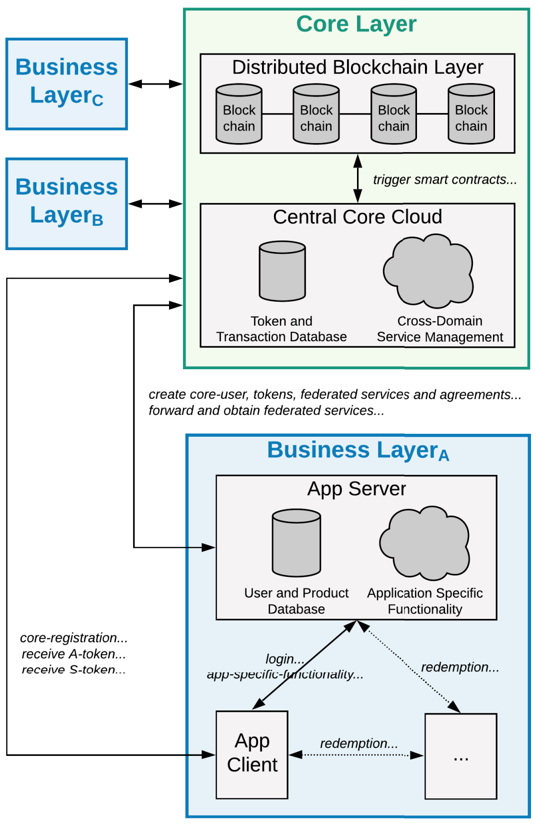
Trusted traceability. Obtained and redeemed services as well as transaction-related data should be traceable by uploading a fingerprint to the distributed network.

B. *Architectural Overview*

Our platform is divided into two main parts, a business layer and a core layer. While the business layer is further composed of an application client and an application server unit, the hybrid core layer consists of a central cloud and a decentralized blockchain part. An overview of the entire system is given in Fig. 2. The business layer provides users with application-specific features, depending on the application area of the system. The application server of this layer interacts with the client devices and keeps them synchronized. It holds real user data (e.g. email, name, etc.), product and service related datasets (e.g. type, name, price, etc.), and offers application specific functionality (login, data processing, etc.), which may vary depending on the underlying business (parking provider, charging provider, etc.). The redemption unit can be part of the application server (online redemption) or outsourced to a dedicated device (parking gate). One or more business layers may communicate with the core cloud over a RESTful interface. The main tasks of the core cloud are abstraction (tokenization) and distribution of heterogeneous services across different business layers. Two types of tokens responsible for authentication and authorization are distinguished:

- **Authentication-token (A-token):** Authentication between application clients and the core layer is managed via A-tokens. They are issued during the registration process of the application client on the core layer. A-tokens are tied to a specific device, core-server-signed and consist of the devices public key, a validity period, and token properties.
- **Service-token (S-token):** The digital ownership of an item or service is represented by S-tokens, which can be compared to digital tickets or entitlements. They can be unambiguously associated with a service provider via an application-Id (a-Id) and a concrete service from the application layer via a service-Id (s-Id). The creation of an S-token can be triggered in two different ways. On the one hand, when an application client obtains a service on the application layer this triggers the token generation request on the core layer via a REST call. On the other hand, a service provider may trigger the creation of S-tokens and forward them to user devices on the application layer. After the issuing procedure of an S-token, they are bound to an A-token. If the A-token expires or is invalidated all corresponding S-tokens are invalidated too.

The core server works as a certification authority (CA) and signs all tokens. The servers certificate is in turn signed by an external root CA and is stored on all participating systems' CA stores (it is globally trusted). Application clients utilize public key pinning for the root certificates in order to prevent malicious server certificates to be issued by tampered CAs. Furthermore, the core server features a federated market-



Core Layer
Distributed Blockchain Layer
Block chain
Block chain
Block chain
Block chain
trigger smart contracts...
Central Core Cloud
Token and Transaction Database
Cross-Domain Service Management
Business Layer$_C$
Business Layer$_B$
create core-user, tokens, federated services and agreements...
forward and obtain federated services...
Business Layer$_A$
App Server
User and Product Database
Application Specific Functionality
core-registration...
receive A-token...
receive S-token...
login...
app-specific-functionality...
redemption...
App Client
redemption...
...

Figure 2. Platform overview: the core layer is attached on top of business layers and responsible for issuing, sharing, and redeeming services.

place layer where services (e.g. authorization to enter a parking lot, redeem a food voucher, trigger the start of charging session) can be put, obtained by other service providers, and forwarded as anonymized S-tokens to their users. Supplementary to this, the decentralized blockchain-unit that is part of the core layer and addressable via a smart-contract based interface. It acts as a trusted, tamper-resistant storage facility for cross-domain transactions (S-token exchange). Additionally, agreements between service providers of different business layers can be defined for automating the creation and forwarding procedure of S-tokens as soon as the pre-defined conditions are met. Due to the distributed design of the blockchain, all transaction-based service data is available to multiple entities and protected against fraudulent changes. Consensus is reached by means of a Proof of Authority approach, involving participating service providers (with their corresponding application servers) as sealers.

C. *Interaction Between App and Core*

We shall now describe the interaction between the application and the core layer. In this we will emphasize the

.

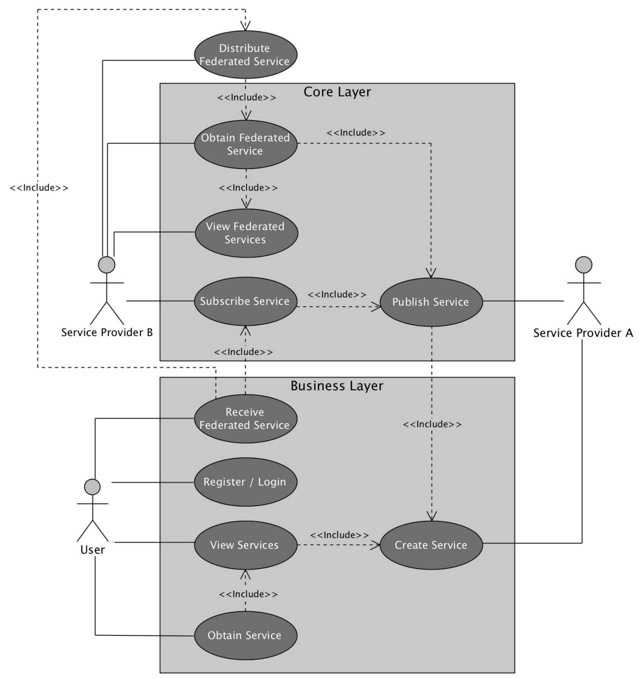
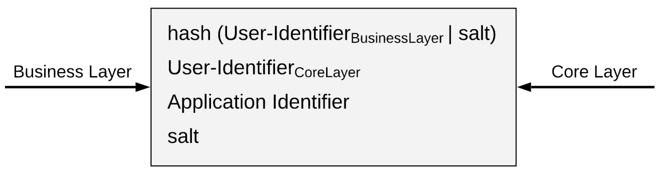


Figure 3. Interaction between the business layer and the core layer from the perspective of service providers and their users.

interplay with the blockchain layer. An overview of the most important features of the system and its actors is given by Fig. 3.

Establish a connection between business and core layer. The first step for a service provider to establish a connection from his business layer to the core foresees a registration on the core layer. This involves the submission of merchant related data and the app-identifier of the client application (e.g. Android application id) via a dedicated REST interface. Subsequently, a service provider specific application Id (a-Id) is issued as well as a dedicated API-key for further interactions from the service provider side with the core layer over the RESTful interface. Furthermore, a blockchain wallet (private-public keypair) is generated since each service provider or rather application server is involved in the consensus procedure of service-based transactions. The private key is used to sign blockchain transactions, while the public key is utilized for identification purpose. After the registration a service provider is finally authorized to publish, obtain, and forward federated services. In order to publish a federated service, descriptive fields (name, price, etc.) and a unique service-Id (s-Id) are submitted from the application layer. Subsequently, the service becomes visible for other service providers (uniquely identifiable via a-Id and s-Id).



hash (User-Identifier$_{BusinessLayer}$ | salt)
User-Identifier$_{CoreLayer}$
Application Identifier
salt
Business Layer
Core Layer

Figure 4. Privacy preserving mapping of a user object between business layer and core layer – confidential data remains on the business level.

The integration of the application client into the overall framework can be done next. First, a successful login of the application client to the application server is necessary. The specific mechanism can vary depending on the underlying application. After this step, the registration between the application client and the core layer is triggered by the application server by sending a registration request. Subsequently, the core server issues a one-time registration-ticket bound to the end-user device, which is forwarded to the device through the application server. The ticket can be redeemed directly by the application client at the core cloud. Next, the creation of an anonymized core user entity is triggered, that is identifiable via a dedicated user-Id (u-Id). Finally, the device receives a user-bound A-token required for core server authentication. This distributed login mechanism assures that the core layer never receives user-related data (e.g. name, email), increasing the level of anonymity and privacy. The real user data remains on the application layer and is not distributed.

Creation of service-tokens. S-tokens can be made available to the user in two different ways: a user obtains services and entitlements on the business layer or a service is obtained by a service provider and forwarded to a user. In the first case, the obtaining of a product on the business layer side triggers the creation of an S-token on the core layer by submitting a corresponding s-Id. Consequently, the S-token is linked with the core-user. Last but not least, a new transaction entry is generated inside the core layer. The second case involves the service provider obtaining a federated service. The moment a service is obtained the creation of an S-token is triggered and the S-token can be assigned to the user via his user id (u-Id).

In order that S-tokens can be assigned to the intended user on the business layer without revealing confidential data of the user on the core layer, a privacy-preserving mapping is used. This is depicted by Fig. 4. A dedicated table is stored on the central core cloud, containing the user identifier of the application layer (e.g. email) hashed together with a salt, as well as the anonymized core user. Furthermore, the application identifier is added in order to address the right app. If a service provider intends to automate the issuing process and make it condition-dependent, agreements can be elaborated by signing a federated service. This triggers a validation flag that makes the agreement valid. An agreement consists of the federated service, additional datasets identifying the interested service provider as well as a condition that determines under which circumstances the federated service specified inside the agreement should be issued. A condition tells how many S-tokens have to be obtained or redeemed by a user before the service (reward) is issued, or which specific S-token needs to be redeemed in order to trigger a reward, for instance. If the agreement's trigger condition is met, the federated service specified inside the agreement is derived to an S-token and forwarded to the client who triggered the current transaction.

Interaction with the blockchain layer. Each transaction, which in our context includes either the creation or the redemption of S-tokens, is stored in a transaction history table on the core server in an anonymized way. Anonymized means that each transaction history entry only consists of a timestamp and different ids that identify the service provider (a-Id), the actual service (s-Id), and core user entry (u-Id), without referring to the real data that lies on business layer. All transactions that involve the obtainment of a federated service by a service provider, are additionally forwarded as key-value pairs to the blockchain through a dedicated smart contract. While the value consists of the hashed transaction entry, the key is composed of the a-Id and a timestamp. For each transaction a receipt as well as the key is generated and forwarded to the service providers of the application layer. The key provides the data management system with the ability to access the transactions for additional validation and tracking of past transactions, while ensuring the integrity of all transactions. A set of transactions is used to compose a new block, which will be confirmed by the network's sealers. If the block was accepted, it will be integrated into the existing blockchain, making it part of the tamper-resistant distributed ledger. Fig. 5 summarizes which data is stored on the blockchain via smart contracts. In short, all data that should be accessible to multiple independent parties (businesses) is saved on the blockchain via smart contracts. On the application client side, every time the transaction history is updated, the core layer queries all corresponding agreements and checks if one of the conditions is met by equalizing them with the transaction history table. If the last service that was added to the history table has resulted in a progress of the agreement's condition, this transaction is also forwarded to the blockchain and linked to the agreement. By this means the service providers can keep track of the completion state of all agreements they are involved in. The same is true for application clients if the interface of the business layer is adapted accordingly. One of the properties of the blockchain is that data once added can no longer be removed or changed. This is especially advantageous when data that should not be altered is stored, but can pose a problem for agreement handling, since often agreements should expire after a specific time or need to be canceled. We took care of this scenario to the extent that

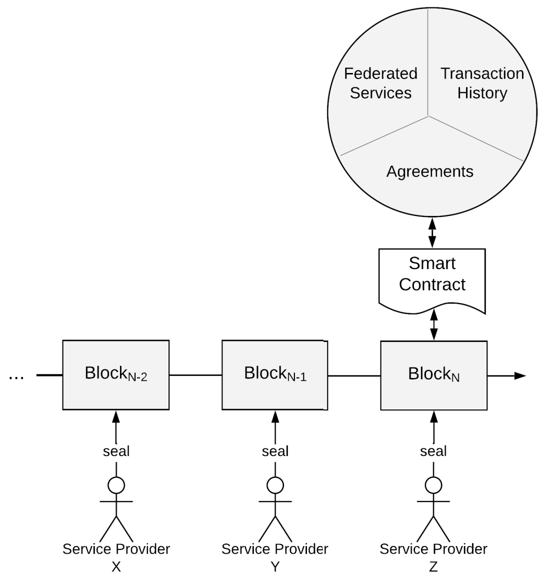


Figure 5. Smart contracts are used to store all cross-domain related service data onto the blockchain in a tamper-resistant manner.

a one-time cancellation request can be used which sets the agreement's validation flag to false. Subsequently, this makes the corresponding smart contract unusable, by blocking all further method calls.

IV. Implementation

There are two classes of entities accessing the core servers functionality: the application servers and the application clients through a RESTful interface, which is protected by basic HTTP authentication. Every call has to be authenticated using a secret API key, which is passed as a username in the basic HTTP authorization header. All communication between clients and the core server is done over HTTPS. The only exception is the device registration process, where a client certificate is not yet available. In this case, the registration-ticket is used for identifying the user. After registration, the application client uses an X.509 certificate for all subsequent TLS connections. The overall platform utilizes PKI combined with ECDSA as signature algorithm and SHA256 for hashing data.

We decided to rely on the already widespread Ethereum blockchain in the context of the blockchain part. We chose Ethereum's PoA Clique consensus algorithm, since it is suitable for achieving cooperation between several service providers in terms of speed, network scope and authorization handling. Each participating service provider becomes a sealer and is involved in the validation procedure. A local Ethereum blockchain was installed on Linux-based machines for the proof of concept we developed. To set up the network including creation of nodes, accounts and the genesis block, the Go Ethereum protocol was utilized. It is available as standalone terminal client under the name Geth. The interaction between the JAVA-based application code and the Ethereum blockchain was established using the web3js API. It is a lightweight, but highly modular Java library for working with smart contracts and interacting with nodes on the Ethereum network. Furthermore, for debugging the local blockchain network an IPC approach was chosen by accessing the local geth.ipc file with the Geth client. A bootnode was used for helping nodes to discover each other, since each node could have a dynamic IP. The bootnode itself runs on a static IP and can be used by other nodes as reference to find participating nodes.

Smart contracts were programmed in Solidity [15], a high-level programming language for the Ethereum Virtual Machine. It is statically typed, supports inheritance and user-defined types. A contract in the sense of Solidity is a collection of code (functions) and data (states) that is stored at specific addresses on the blockchain. They have to be installed in a one-time operation on all nodes by sending a transaction to the network. Once installed, modifications are no longer possible. We tested the contracts using the Remix browser application (www.remix.ethereum.org). In order to embed it into the server application the contract was compiled using the solc-compiler. During the compilation process a binary file and an application binary interface in JSON format are created. These outputs are required to generate JAVA wrapper classes with the web3j command line utility. Finally, these wrapper files can directly be integrated into an application.

V. Security Evaluation

Our platform acts as a cross-domain entitlement marketplace and integrates blockchain technology to further enhance the level of security and trust between different service providers. The business layer is composed of a client-server infrastructure, while the core is attached on top of it to provide the means for sharing different services. A security evaluation of the elaborated proof of concept is given in the following.

User and service related data from the business layer is tokenized for increased anonymity before being submitted to the core layer. Furthermore, these tokens are responsible for security goals such as authentication and authorization. On the one hand, authentication from the application client to the core layer is provided through A-tokens, which are extended certificates, tied to a particular device and issued in the course of a Kerberos-based authentication mechanism, ensuring that no real user data is forwarded to the core server. On the other hand, services are converted to S-tokens that identify the actual entitlement and the service provider responsible for the redemption. In our security model, the core layer communicates with the business layer over an encrypted channel (HTTPS). The tokens themselves (A-token, S-token), however, are not encrypted. This reduces additional computational overhead and is not security critical, since tokens are only composed of anonymized data. Furthermore, if an adversary had access to the core layer's transaction database he would not learn anything about the raw data, as it is obfuscated. Regarding transactions, only anonymized entries consisting of different ids and a timestamp are stored. Those that should be accessible by multiple parties are uploaded in a hashed format (SHA-256) to the blockchain. By doing so, participating service providers have the possibility to keep track of federated services and the completion state of agreements. If a device is lost or stolen, the possibility of spending existing tokens with that device cannot be prevented. This risk can be reduced by revoking a devices A-token in a timely manner, which subsequently invalidates all S-tokens automatically.

The overall architecture uses PKI with ECDSA for identifying communication participants and verifying the data exchanged. All tokens are core-server signed and can be verified with the server's public key. In order to prevent the issuance of interacting with a malicious core server application, clients may use public key pinning. A trusted root certificate is pre-installed on all devices. Additionally, all blockchain transactions are cryptographically signed and ensure that an attacker cannot corrupt the network. Note that while data integrity is not ensured in each node, since a single node can tamper with its local copy, we can still reduce the risk by means of a sufficient distribution of the data.

The decision making power is distributed from a central authority across multiple network participants, which strongly impedes fraudulent behavior. If hackers tamper with the block data, the network's consensus model will ensure that other nodes reject the bad node. Unlike other consensus algorithms where the most important resource is computing power (Proof of Work) or monetary resources (Proof of Stake) in the case of our Proof of Authority (Ethereum's Clique) approach, a set of pre-approved authorities is responsible for the integrity of the network. This means an adversary would need to gain control over other sealer's machines or persuade them to act maliciously, which makes an attack vector more improbable. We assume the blockchain to be tamper-resistant. This requires a sufficiently large network of trustworthy sealers. Any new service provider wishing to join the network, must first be approved by the other sealers, which gives us full control over which nodes can seal blocks. To make sure a malicious signer cannot do too much harm to the network any signer can sign at most one of a consecutive number of blocks. The same voting is applied when an authority node is removed from the network.

VI. Conclusion

Cross-domain service exchanges are often difficult to achieve due to rigid system boundaries and lack of standardization as well as a scant level of trust and privacy. In this paper we described a framework for issuing and sharing services from different application areas in the form of digital entitlements across heterogeneous applications. Our platform is subdivided into a business layer composed of a client-server system, where application specific features lie, and a core part acting as a common trusted layer for abstracting user and service data as well as for distributing different entitlements across multiple entities. The predominant trust mechanism of the platform is a public-key infrastructure (PKI) that maps an identity to a cryptographic public key using signed certificates. Authentication of devices as well as authorization of services are handled with anonymized tokens. Due to the generic nature of the tokens multiple heterogeneous entitlements can be represented, shared and seamlessly integrated in different systems and applications. Since the blockchain emerged as a tamper-resistant tool with great traceability and data integrity protection mechanisms, we extended the core layer with a decentralized blockchain part. A federated Ethereum-based blockchain with a Proof of Authority based consensus approach that involves participating services providers, is used to store and manage federated services and agreements. The communication between the centralized cloud and the decentralized blockchain part is enabled via a smart contracts based interface. Furthermore, from all transactions, including the issuing, forwarding, and redemption of cross-domain-services, data pointers are retained on the distributed ledger for verifying the data against fraudulent changes. Furthermore, by applying our framework we incentivize the use of cross-domain rewards. In this sense, a customer may receive additional offers and services coming from independent service providers as soon as pre-defined conditions between different service providers are met. Regarding future work we will look into end-to-end security strategies in order to provide means for exchanging confidential data (e.g. sensitive sensory data) across heterogeneous systems.

Acknowledgment

This work was written in tight cooperation with CISC Semiconductor GmbH and Graz University of Technology (Institute of Technical Informatics). Parts of this research have been elaborated within the research program STIP, funded by the Austrian Federal Ministry for Digital and Economic Affairs (BMDW) under the program COIN-Programmlinie "Netzwerke" (grant number 867914).

References

[1] G. Broad and H. Cambridge, "Blockchain A Beginners Guide," pp. 1–57, 2017. [Online]. Available: https://blockchainhub.net/blockchain-technology/

[2] V. Buterin, "On Public and Private Blockchains," 2015. [Online]. Available: https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/

[3] G. WOOD, "Ethereum: a Secure Decentralised Generalised Transaction Ledger," *Ethereum Project Yellow Paper*, vol. 32, no. 10, pp. 1365–1367, 2018.

[4] G. Zyskind, O. Nathan, and A. S. Pentland, "Decentralizing privacy: Using blockchain to protect personal data," *Proceedings - 2015 IEEE Security and Privacy Workshops, SPW 2015*, pp. 180–184, 2015.

[5] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "MedRec: Using blockchain for medical data access and permission management," *Proceedings - 2016 2nd International Conference on Open and Big Data, OBD 2016*, pp. 25–30, 2016.

[6] X. Liang, J. Zhao, S. Shetty, and D. Li, "Towards data assurance and resilience in IoT using blockchain," *Proceedings - IEEE Military Communications Conference MILCOM*, vol. 2017-Octob, pp. 261–266, 2017.

[7] G. C. Polyzos and N. Fotiou, "Blockchain-assisted information distribution for the internet of things," *Proceedings - 2017 IEEE International Conference on Information Reuse and Integration, IRI 2017*, pp. 75–78, 2017.

[8] M. Vukolic, "Rethinking Permissioned Blockchains," *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts - BCC '17*, pp. 3–7, 2017.

[9] "Hyperledger Fabric," 2018. [Online]. Available: https://github.com/hyperledger/fabric

[10] W. Li, A. Sforzin, S. Fedorov, and G. O. Karame, "Towards Scalable and Private Industrial Blockchains," in *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts - BCC '17*. Abu Dhabi: ACM, 2017, pp. 9–14.

[11] N. Mohamed, J. Al-Jaroodi, S. Lazarova-Molnar, I. Jawhar, and S. Mahmoud, "A service-oriented middleware for cloud of things and fog computing supporting smart city applications," *2017 IEEE SmartWorld Ubiquitous Intelligence and Computing, SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI 2017 - Conference Proceedings*, pp. 1–7, 2018.

[12] A. Rech, M. Pistauer, and C. Steger, "Increasing Interoperability Between Heterogeneous Smart City Applications," in *Lecture Notes in Computer Science*. Tokyo: Springer International Publishing, 2018, pp. 64–74.

[13] European Parliament and European Council, "REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 27 April 2016," p. 88, 2016.

[14] P. Canbay and H. Sever, "The Effect of Clustering on Data Privacy," in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. Miami: IEEE, 2015, pp. 277–282.

[15] "Solidity," 2016. [Online]. Available: https://solidity.readthedocs.io