A COMPARATIVE EVALUATION OF BLOCKCHAIN SYSTEMS FOR APPLICATION SHARING USING CONTAINERS

ABSTRACT

The Cloud computing paradigm is built on the concept of virtualization, allowing multiple virtual machines to cohabit on one physical device to enable the scaling up and down of applications through elastic on-demand provisioning. More recently containers e.g. Docker, have been shown to enable a more lightweight mechanism than hypervisors and proved to be a viable alternative for virtualization, based on shared operating systems. The advent of such lightweight environments has brought a multitude of application uses in research, science and industry, enabling pre-configured operating environments to be shared, reused and instantiated on demand. The sharing of containers has currently been exposed using centralized repositories (e.g. Dockerhub), which allows containers to be shared and to form the building blocks for further development. In this paper, we take a look at the next evolution of this lifecycle and consider whether it is viable to securely share container-based applications within a decentralized group of individuals and to provide an audit trail recording exactly who has shared what, and with whom. For this purpose we consider the use of Blockchain technologies, and consequently perform a comparative analysis of Blockchain technologies for this use case. The paper provides mostly a review and taxonomy of different ledger systems, which we believe may be of interest to the SafeData Workshop participants.

Adam Brinckman, Donald Luc, Jarek Nabrzyski, Gary L. Neidig, Joel Neidig,
Tyler A. Pucketty, Swapna Krishnakumar Radha, Ian J. Taylor

A Comparative Evaluation of Blockchain Systems for Application Sharing Using Containers

Adam Brinckman*, Donald Luc*, Jarek Nabrzyski*, Gary L. Neidig†, Joel Neidig†
Tyler A. Puckett†, Swapna Krishnakumar Radha*, Ian J. Taylor*
*University of Notre Dame, Center for Research Computing, Notre Dame, IN, 46556
{abrinckm, dluc, naber, sradha, ian.j.taylor }@nd.edu
†Itamco, 6100 Michigan Rd, Plymouth, IN 46563
{glneidig, jdneidig, tapuckett}@itamco.com

***Abstract*—The Cloud computing paradigm is built on the concept of virtualization, allowing multiple virtual machines to cohabit on one physical device to enable the scaling up and down of applications through elastic on-demand provisioning. More recently containers e.g. Docker, have been shown to enable a more lightweight mechanism than hypervisors and proved to be a viable alternative for virtualization, based on shared operating systems. The advent of such lightweight environments has brought a multitude of application uses in research, science and industry, enabling pre-configured operating environments to be shared, reused and instantiated on demand. The sharing of containers has currently been exposed using centralized repositories (e.g. Dockerhub), which allows containers to be shared and to form the building blocks for further development. In this paper, we take a look at the next evolution of this lifecycle and consider whether it is viable to securely share container-based applications within a decentralized group of individuals and to provide an audit trail recording exactly who has shared what, and with whom. For this purpose we consider the use of Blockchain technologies, and consequently perform a comparative analysis of Blockchain technologies for this use case. The paper provides mostly a review and taxonomy of different ledger systems, which we believe may be of interest to the SafeData Workshop participants.**

I. Introduction

The modern use of computation for scientific discovery has fundamentally changed the way scientists interact with data. Researchers now routinely interact with large amounts of data, various computational infrastructures, and sophisticated analysis tools to evaluate their hypotheses and results. There are a number of projects in progress (e.g. [1], [2], [3]) that are beginning to allow scientists a means of publishing digital research objects and analytical pipelines to enable interactivity and streamlined reproducibility of such experiments. These platforms are starting to see an emergent trend where researchers can publish research and expose the analysis of diverse data sets and tools for the visualization of results.

Alongside such developments have been the underpinnings of virtualization. For example, Docker containers are capable of providing portable and isolated applications by packaging them in containers based on LXC technology [4]. Furthermore, it has been previously shown that containers result in equal or better performance than VMs in almost all cases [5]. The use of containers consequently as a lightweight sharing mechanism for applications is now not only a vision, but has been a reality for a few years.

By building on such activity, in this paper, we look at the possibility of empowering users to be able to share applications securely with individuals, or within groups of individuals, to foster a private means for collaboration of application development or data analytics. We aim to provide an environment that has the following key attributes:

- An environment that allows users to share applications using containers, which does not have a central point of failure;
- An environment that is secure and able to withstand malicious attacks;
- Users can trust that sharing rules will be executed exactly as they intend without the need for a trusted third party;
- Sharing within groups should be viewable by all parties in that group and all transactions should be immutable i.e. they cannot be altered or deleted – this will help to foster open collaboration within private groups;
- The application container should be transmitted securely to a participant, or the group of participants with whom it was shared.

Using these basic criteria, we consider a use case in the design of ASIC chips. Our team has been working on designing and building the CRAFT repository [6], with the aim of it being a collaborative gateway for circuit designers to be able to work on chip designs together. During the design process, the multiple parties, very often independent companies that are involved, need to share implementations and IPs (intellectual property), but in a controlled and secure way. Very often the designs are first shared privately within a team, until a final design is accomplished and released for sharing with other partners. Under these circumstances, a set of secure groups representing different teams should be formed to enable secure and controlled collaboration amongst team participants for development of subsequent designs of a chip during the whole process. For this purpose, we investigate the use of the Blockchain technology to foster such environments

and to perform an evaluation to find a suitable Blockchain implementation for implementing such a platform.

We discuss how this system could be used to design a platform for securely sharing containers by addressing the volume of data that would be involved in sharing containers across a Blockchain. To this end, we propose a hybrid system that takes a hash of the data for recording onto the Blockchain and stores the encrypted containers in a secondary system by using the hash as the lookup. We believe this approach can be used to satisfy the above constraints and provide a scalable and decentralized mechanism for such a platform. We then provide the status of our on-going project, called Secure Messaging on a Blockchain Architecture (SIMBA) funded by DARPA, in which we are developing these goals in order to realize this vision.

The rest of the paper is organized as follows. Section II discusses the underlying virtualization technologies that provide the core enabler for the platform. Section III then provides a set of evaluation criteria and performs a survey of Blockchain systems against these criteria. Section VI provides an overview of a design, in which the chosen Blockchain system is used to implement an application container sharing capability, in a way that can scale efficiently as the number of transactions increases. In Section VII, we provide a current status of the current implementation of this work.

II. Virtualization And Containers

Virtualization is the foundation of Cloud computing that allows users to run their own application and environment on multiple virtual machines (VM), which are layered on the physical hardware. Virtualization enables scaling up and down of applications by elastic on-demand provisioning of VMs in response to their variable load to achieve increased utilization efficiency at a lower operational cost, while guaranteeing the desired level of Quality of Service (QoS, such as response time) to the end-users. Typically, VMs are created using VM images, which are stored in vendor proprietary repositories. Such an approach creates vendor lock-in and hampers portability or simultaneous usage of multiple federated Clouds. VMs hinder this process further because they are time consuming to create, monolithic in nature (with large deployment and migration overheads). Another problem comes with the fact that they are based on emulating virtual hardware, meaning they are heavyweight in terms of system requirements.

Containers, on the other hand, are based on shared operating systems. They are much more lightweight and more efficient than hypervisors. Instead of virtualizing hardware, containers rest on top of a single Linux instance. This means one can provide incremental snapshots, which do not require the duplication that VMs need, to provide a small, neat capsule containing the target application and perhaps some minimal dependencies that are required for the particular application instance at hand. The implications are clear - using a tuned-up container system, you can often expect to see four-to-six times as many server instances per capacity compared to using Xen or KVM VMs. Containers use the Linux LXC user space tools, where applications run with their own file system, storage, CPU, RAM, and so on. While the hypervisor abstracts an entire device, containers abstract just the operating system kernel.

Container environments are timely with a number of approaches to the underlying infrastructure merging. For example, Google (which uses cgroups for its containers) and Parallels (which uses 'bean-counters' in OpenVZ) have merged their codebases so there are no practical differences between them. The recent advances in Docker [7] have also made them popular and created a surge in activity in support, stability and growth.

More recently, there are also more lightweight images that aim at vastly reducing the size of the containers. For example, we have been investigating RancherOS [8], which has built a minimalist Linux distribution customized for running Docker containers. It provides the minimal dependencies necessary to run Docker directly on top of the Linux Kernel, and have all user-space Linux services distributed as Docker containers. Traditional Linux distributions are primarily designed for administrators to operate whereas RancherOS is designed to leverage the Docker API and host sophisticated management agents. Systems like RancherOS therefore could provide a good initial platform for the sharing of container-based applications across secure groups.

III. Blockchain

From the initial release of bitcoin in 2009 [9], Blockchain has historically supported the first digital currency providing a digital ledger that records every bitcoin transaction that has ever occurred. Since then, the use of the technology has been proposed and used in multiple domains, enabling a permanent, transparent ledger system for decentralized crowd-funding, to enable a distributed and decentralized social network called Synereo [10], compiling data on sales, storing rights data, and tracking digital use and payments to content creators, such as musicians.

The secure transmission of files or messages, on the other hand, has been typically implemented as a server-based approach that protects sensitive data when being transported between two or more participants. For corporations, this provides protection beyond corporate borders and often offers compliance with industry regulations such as HIPAA, GLBA and SOX. Because of the centralized nature of secure messaging architectures, messages can originate from any graphical user interface (GUI) without the need to manage cryptographic keys beforehand. Secure messages provide non-repudiation as the recipients are personally identified and transactions are logged by the secure email platform. In the Blockchain context, there are some applications related to messaging but they do not offer the same level as security as a traditional secure messaging platform. For example, the CryptoGraffiti.info service offers the ability to encode arbitrary text as Bitcoin addresses, which can then be imported to the wallet in order to save the desired message into the Blockchain forever, thus allows users to write and read arbitrary messages

on the Blockchain via a web interface. CoinSpark [2] also allows text messages and files to be attached to any transaction.

Blockchains are capable of transferring any digital data between participants in a secure and auditable fashion with an authorization infrastructure that enables multiple parties to coexist, conducting any kind of transaction and providing the authorization hierarchy capable of defining who has access to what. Within the context of this paper, there are two specific areas that we focus on: the secure sharing of containers and smart contracts for managing the data management, and writing and reading to and from the Blockchain.

To this end, we identify the following features that would constitute a suitable Blockchain system:

1) Secure Sharing
    a) Flexible Payloads: Support transmission of files
    b) Multiple Channels: Support for multiple channels/groups on the Blockchain
    c) On-Chain Encryption: ability to store encrypted data on the Blockchain
    d) API to underlying services - We would like to create a clear separation between the front-end Web application and the back-end interface to the Blockchain, so an API, e.g. a REST-based API, is required.
2) Smart Contracts
    a) We need an interface to the reading and writing of data so we can plug in custom extensions of how we process the data.
    b) Smart contracts typically provide a pluggable architecture for implementing customization of a Blockchain transaction and would facilitate such logic.

The rest of this section is organized into two sections that cover non-hyperledger and hyperledger-based approaches, which is then followed by a taxonomy of these systems relating to the categories defined above.

IV. Non-hyperledger-based Projects

A. BlockCypher

BlockCypher [11] API interacts with Blockchains by mainly using RESTful JSON API which is accessed over HTTP or HTTPS from the api.blockcypher.com domain. It has an official and unofficial version. The unofficial version was made by the community and BlockCypher can not guarantee it is up to date. Essentially, BlockCypher fuses Blockchain technology with enterprise scale. It implements the entire Bitcoin protocol but it is designed to be agnostic to any particular Blockchain and therefore supports a multitude of chains. BlockCypher boasts that it is the only enterprise system that posts their reliability numbers and is considered to be best in class for enterprise development.

B. Openchain

OpenChain [12] is developed by Coinprism, it uses Partitioned Consensus. uses a client-server architecture instead of a peer-to-peer architecture. There is no miner, transactions are directly validated by the asset administrator which means transactions are instant and free. Uses assigned aliases to users instead of using base-58 addresses. OpenChain has multiple levels of control such as a fully open ledger that can be joined anonymously, closed loop ledger where participants must be approved by the administrator, or any mixture of the two above.

C. Bitcore

Bitcore [13] runs on a peer-to-peer network powered by the Bitcore Library. It provides a powerful Blockchain API and Insight Blockchain explorer, a backend only Blockchain REST and websocket API service. Uses micropayment channels for rapidly adjusting bitcoin transactions. It implements a mnemonic code for generating deterministic keys.

D. ChromaWay

ChromaWay [14] is a colored coins open source platform that uses the high performance ChromaNode, an open source Bitcoin data API which enables SPV and supports notifications via WebSocket set protocol. ChromaWay is a scalable architecture that allows several servers to cooperate to serve the data. A new Smart Contract protocol is currently being worked on, which should allow for the digitizing and representing workflows in a secure, private, and efficient way. ChromaWay is partnering with LHV Bank by using Cuber, which established the first time a financial institution issued assets on a Blockchain. It offers cloud solutions for their platform and APIs.

E. Ethereum

Ethereum [14] is an adaptable and flexible open source Blockchain platform that is programmable by the developer. This allow for more complexity than your average Blockchain platform. Ethereum Virtual Machine, or EVM, is at the heart of Ethereum. EVM is where all the code executed focuses on achieving zero downtime to make the Blockchain data forever unchangeable. The decentralized Ethereum platform runs smart contracts that are written using Solidity, which is a contract-oriented, high-level language whose syntax is similar to that of JavaScript and it is designed to target the Ethereum Virtual Machine (EVM). Solidity is statically typed, supports inheritance, libraries and complex user-defined types among other features. Such applications run on a custom built Blockchain, which sits on top of a stable back-end infrastructure. Using the platform, developers can create markets, store registries of debts or promises, even execute a will, all without a middle man or counterparty risk.

Ethereum is a very flexible platform, but Solidity is not designed for integration with other non contractual based tools e.g. hashes and NoSQL may be needed to support a scalable container transmittable platform. Although there are emerging third party APIs to Ethereum from BlockCypther [15], Ethereum does not provide an API itself.

F. Monax (Eris)

Monax [16] has their own SDK (Eris) which was built from the ground up and allows for easier and faster application building. Eris is a modular platform of numerous connected services and tools that simplify building, testing, and running ecosystem applications. Eris also has its own chains called Eris chains. Eris chains are the gateway to the unlocking of permissible, smart contract optimized Blockchains. Eris chains also expose a range of options for developers to create, administer, and operate Blockchains of various varieties. They also boast about updating their software and add-ons which make it easier to add a new feature or fix a bug, and all updates are highly tested and vetted by their legal engineers.

G. MultiChain

Multichain [17] allows managed permissions on who can send, receive, create, and more. It also has simple deployments such as a 2-step new block creator. MultiChain allows for multiple key-value, time series or identity databases on a Blockchain which is great for timestamping, data sharing, and encrypted archiving.

V. Hyperledger Blockchain Projects

In December 2015, the Hyperledger project [18] started as a collaborative effort led by the Linux Foundation and consisting of 30 founding members. It differentiates itself from the other aforementioned implementations by being more versatile and business application oriented, and it moves away from the currency roots to expose Blockchain technology to more application domains. It was created to advance Blockchain technology focusing on key features to create a cross-industry open standard for distributed ledgers to help transform business transactions globally. Their motto is "A code speaks" and therefore has a strong focus on collaborative projects that develop technologies for deploying distributed ledgers quickly and easily.

There are several efforts underway at present:

- Sawtooth Lake
- Iroha
- Blockchain Explorer
- Cello
- Fabric

These are described in this section.

A. Sawtooth Lake (in Incubation)

Sawtooth Lake [19] is a highly modular platform for building, deploying and running distributed ledgers. Participants in the ledger can submit transactions to add, remove or modify records on the system, according to a set of rules that are guaranteed to be enforced by the ledger. It has a simple architecture with four main layers:

1) Application Layer
2) Ledger Layer
3) Journal Layer
4) Communication Layer

The system exposes a software framework for constructing decentralized ledgers with extensible transaction types. The data model and transaction language are implemented using a transaction family and they provide three default transaction families for building, testing and deploying a marketplace for digital assets:

- EndPointRegistry: A transaction family to register ledger services
- IntegerKey: A transaction family to test deployed ledgers
- MarketPlace: A transaction family to buy, sell, and trade digital assets

The configuration transaction family stores on-chain configuration settings, along with a configuration family transaction processor written in Python.

Sawtooth Lake abstracts the core concepts of consensus and isolates consensus from transaction semantics. Sawtooth Lake currently provides a single consensus protocol called PoET, for "Proof of Elapsed Time", which is a lottery protocol that builds on trusted execution environments (TEEs) provided by Intel's SGX to address the needs of large populations of participants.

The communication layer implements a gossip protocol for communicating among participants in a validator network. The layer provides basic facilities for sending messages directly to other validators and broadcasting messages to the entire validator network. Messages are typed, signed by the originator, and encoded in CBOR for transmission. Other Sawtooth Lake components can register type-specific handlers for messages. The Communication layer provides two related abstractions that enable customization for different deployment environments:

- Topology : A protocol for establishing connections between peers in the gossip network can be customized. The SLDLP currently implements three protocols, one for random connections (that is most resilient to malicious manipulation), one for scale-free topologies (that supports efficient broadcasts), and one based on a distributed hash table (that supports routing to 'interest groups').
- Routing: A default method for forwarding messages through the network is a simple broadcast to all peers. However, this may be overridden as is the case with a distributed hash table implementation.

Sawtooth is a project still in incubation and although it has several useful features to implement SIMBA, it lacks some of the flexibility and extensibility offered by the other implementation e.g. smart contracts is still under development and there is no support for multiple channels at present.

B. Iorha (in Incubation)

Iroha [20] is a distributed ledger project that was inspired by Fabric and aims to provide a development environment where C++ and mobile application developers can contribute to the Hyperledger Project. The project seeks to complement Fabric, Sawtooth Lake, and other potential projects, by creating reusable components in C++ that can be called from languages such as Go. In this way, Iroha attempts to realize a robust

library of reusable components that can be selected and used freely by those running distributed ledgers on Hyperledger technology.

The design and architecture of Iroha is greatly inspired by Fabric e.g. membership, Blockchain, and chaincode services. APIs are made to be similar to Fabric and provide an environment for C++ developers to contribute to Hyperledger projects, to provide infrastructure for mobile and web application support, and to provide a framework to experiment with new APIs and consensus algorithms that could potentially be incorporated into Fabric in the future. Iroha specifically aims to bring in more developers while providing libraries for mobile user interface development.

The full specification can be found in [20]

C. Blockchain Explorer (in Incubation)

Hyperledger Blockchain-explorer [21] is a project in Incubation that was proposed by Christopher Ferris (IBM), Dan Middleton (Intel) and Pardha Vishnumolakala (DTCC) to create a user-friendly web application for a Hyperledger to view/query blocks, transactions and associated data, network information (name, status, list of nodes), chain codes/transaction families (view/invoke/deploy/query) and any other relevant information stored in the ledger.

Instead of launching competing open-source services, the Hyperledger Explorer is an effort to merge the Blockchain explorers being developed by DTCC, IBM and Intel. Similar to block explorers already being offered for other public Blockchains, the tool makes it easier to learn about Hyperledger from the inside, while still protecting the privacy valued by many of the non-profit organization's members.

The explorer, when completed, is expected to give Hyperledger developers and non-technical users access to block information, transaction data, network information (such as a list of nodes) and chain codes or transaction families.

Since the the purpose of this project is to support the other hyperledger projects, it is not included in the taxonomy below.

D. Cello (in Incubation)

Cello [22] focuses on providing Blockchain as a Service (BaaS) to help reduce the effort required for manipulating (e.g., create and destroy) chains manually. With Cello, operators can create and manage multiple Blockchains in a pool through a dashboard, at the same time users (typically the chaincode developers) can obtain Blockchains instantly with a single request, as illustrated in the figure below. Cello will support existing Hyperledger Blockchain implementations including Fabric, SawToothLake and Iroha.

Currently, in order to boot a chain, a developer needs to install scripts and if multiple tenants are required to obtain separate chains at the same time, they have to modify the scripts carefully and create these chains manually. This procedure is time consuming, and even worse, can often lead to possible misconfigurations.

The focus of the Cello project is to bring the Cloud service model into the Blockchain ecosystem, to provide a multi-tenant chain service efficiently and automatically. Taking Hyperledger Fabric, as an example, currently, the solution is to create a Hyperledger Fabric chain, which includes:

Manual installation of each peer node on different servers. This requires much effort and is error prone. Setup scripts (e.g., Docker-Compose) to start a fabric network. This requires a specific server configuration, which makes it hard to share resources and to dynamically create multiple chains.

Cello solves these problems in a different way, by maintaining a pool of chains automatically. Users get chains with various configurations instantly, while operators can dynamically scale the physical resources through a dashboard. Since the the purpose of this project is to support the other hyperledger projects, it is not included in the taxonomy below.

E. Fabric

Hyperledger Fabric [23] is the first project to graduate Incubation to Active status on March 3, 2017. Fabric, written in Go [24] is a DAH and IBM collaborative effort implementation of Blockchain technology. Fabric can be a single network or can comprise networks of networks, each managing their own assets, agreements and transactions. The so-called Ordering Service (OS) is the founder of a network and is initiated using a config file with the rules (usually called Policies) that govern the network, which can, for example, be used to govern how members can be added or removed, and configuration details like block size. The rules will also include policies for changing the rules using a consensus amongst some or all of the members of the network. Fabric also requires some endorsement policies to act.

The OS represents a network's certificate authority, which is responsible for verifying and authenticating transactions arriving from any member of the network. One of Fabric's key innovations is the ability to run multiple channels, a private Blockchain overlay on the network, to allow for data isolation and confidentiality. A channel-specific ledger is shared across the peers in the channel, and transacting parties must be properly authenticated to a channel in order to interact with it. The OS processes orders on a first-come-first-serve basis and returns blocks to their corresponding channels. The control of the OS can be shared and co-administered by the participating members in the network or it can be hosted and maintained by a trusted third-party.

Fabric implements a sophisticated event mechanism where validating peers and chaincodes can emit events on the network that applications may listen for and take actions on. There is a set of pre-defined events, and chaincodes that can generate custom events. Events are consumed by one or more event adapters, which asynchronously receive the information. Adapters may further deliver events using other vehicles such as Web hooks or Kafka.

Fabric's main focus area is on smart contracts, which are implemented using a concept called "chaincode". Chaincode is essentially a piece of code that is written in one of the supported languages, such as Go or Java and it is installed and instantiated through an SDK or CLI onto a network of

Hyperledger Fabric peer nodes, enabling interaction with that network's shared ledger. There are three aspects to chaincode development:

1) Chaincode Interfaces
2) APIs
3) Chaincode Responses

For communication, Fabric uses gRPC [25] for the peer-to-peer communication, which allows bi-directional stream-based messaging. It uses Protocol Buffers to serialize data structures for data transfer between peers, which are a language-neutral, platform-neutral and extensible mechanism for serializing structured data. Data structures, messages, and services are described using proto3 [26] language notation. Messages consist of 4 types: Discovery, Transaction, Synchronization, and Consensus.

1) Fabric Composer: Fabric Composer [27] and the on-line tool [28] allows a user to define a business network, creating and tracking assets and performing transactions. Business networks represent an economic network made up of participants, assets, and the transactions that are performed between them. Assets are tangible or intangible goods, services, or property, and are stored in registries. Transactions are operations which may transform, create, and transfer assets or participants. Participants can interact with multiple business networks and maintain assets as they move between them using set identities. Applications can consume the data from business networks, providing end users with simple and controlled access points to the business network. Business networks can also be integrated with existing systems to create assets, transactions, and participants based on existing data.

Once assets, transactions, and access rules have been defined, Fabric Composer can automatically create a REST API to those resources, which can, in turn, be used to make transactions on the Blockchain. The API is generated using node.js and is documented using swagger [29]. This makes it an extremely useful REST interface to Fabric from front-end applications.

2) Fabric SDK: Application programming interfaces or APIs can be used to develop applications that can interact with the Blockchain network in lieu of end users. Hyperledger fabric SDK provides a useful API. This API can be used by Node.js applications to communicate with core components of a hyperledger fabric based Blockchain network. The SDK provides APIs that support two main pluggable components namely, pluggable key value store and pluggable member service. Pluggable key value store can be used to access keys associated with a member. The "chain.setKeyValStore()" method overrides the file-based key value store implementation available by default. Access to the chain key value store must be given appropriate protection as it holds sensitive private keys. The pluggable member service component can be used to register and enroll new members into Blockchain network. The ''chain.setMemberServices()" method overrides the default implementation of MemberServices. Member services can be used for the purpose of creating a hyperledger fabric based permissioned Blockchain network. A permissioned Blockchain network provides confidentiality, anonymity for members and unlinkability of transactions.

System	Flexible Payloads	Smart Contracts	Multiple Channels	On-Chain Encryption	REST API
BlockCypher	✗	✗	✓ (blockchains)	✓	✓
Openchain	✓	✓	✗	✓	✓ Partial
Bitcore	✗	✗	✗	✓	✓
ChromaWay	✓	Not Released	✗	✓	✓
Ethereum	✓	✓	✗	✓	✗
Monax	✓	✓	✗	✓	✓ Partial
MultiChain	✗	✗	✗	✓	✗
Sawtooth Lake	✓	Not Released	✗	✓	✓ Partial
Iroha	✓	✓	✗	✓	✗
Fabric	✓	✓	✓	✓	✓

Fig. 1. The Resulting Taxonomy

F. Taxonomy of Blockchain Systems for Application Sharing

The above Blockchain systems were reviewed according to the following criteria, as discussed earlier in this section and the resulting taxonomy is presented in Figure 1.

As shown in figure 1, Fabric checks all of the criteria and shows the most promise as a suitable candidate for implementation of the system. It provides a number of flexible ways of accessing a Fabric network (via the fabric composer or SDK) it provides secure channels that can be configured with multiple roles in order to scope internally. Fabric also provides users with the capability to enable permissioned Blockchain networks. It has a built-in feature called the "membership identity service" which manages user IDs and also authenticates users into a Blockchain network. When a user joins an existing or newly created hyperledger fabric Blockchain network, he or she is provided with enrollment certificates or ECerts. Fabric employs the use of a Certificate Authority (CA) service for enabling users to authenticate across the various components in the system. For example, the ECerts are provided by Fabric's CA module, which enables a user to authenticate and obtain authorization into the Blockchain network. It also provides participants with transaction certificates or TCerts. TCerts provide anonymity and unlinkability when transactions are performed on a hyperledger Fabric Blockchain.

VI. Container-Based Application Sharing Using Blockchain

As discussed in the previous section, data or messages that are put onto the Blockchain can be encrypted raw data. However, we would also like the system to support the secure messaging of files and also containers. The sharing of containers would allow pre-configured applications and/or data e.g. using Linux containers, to be sent to participants using the

Blockchain. This level of sharing opens up several possibilities if the sharing can be secured e.g. collaboration on sensitive data or applications, customized rendering or visualization tools wrapped along with the data so that the recipient can view without needing to install complex applications, and so on. For the use case of chip designs, the sharing of containers would facilitate the collaborative nature of the chip design process, in order to be able to work on new designs in a secure way.

Many of the implementations include a GUI, and there are a few different options to run GUI applications inside a Docker container, like using SSH with X11 forwarding, or VNC, but the simplest method is to share the X11 socket with the container and use it directly. This allows direct connectivity to the applications running on a container.

However, one key issue with the transmission of larger files, such as containers, is how such data impacts the size of the Blockchain. The Blockchain is not designed for large datasets because it will get very large very quickly and the computation of the addition of new data to the Blockchain will get more and more time consuming. To combat this issue, we are envisioning using SHA2 hashes to store a footprint of the data and to store the data itself externally using a NoSQL database. In this section, we describe the architecture for this approach. We first look at containers and NoSQL and then feed these technologies into the current design of how SIMBA will address the passing of large documents on the Blockchain.

A. Hash Functions

A hash function is a mapping of a message from any size input message to a fixed length hash key, which is typically far smaller than the original message. A cryptographically strong hash key must have the following properties:

- It must be non-reversible: One must not be able to construct the original message from the hash key.
- It must be sensitive to input changes: It must change significantly with any small change in the input message, even if this change occurs in a single bit. This is also known as the avalanche effect, i.e. a small change in the input creates a large change in the output.
- It should be collision resistant: It should be impractical to find two messages with the same hash key.

Formally, a hash function H takes a message m of arbitrary length as input and produces a bit string h having a fixed bit-length as output i.e. $h = H(m)$. A hash is similar to the extra bits added to a packet in order to perform error correction. Hash functions are one-way functions, meaning that it is unfeasible to find the input m that corresponds to a known output h. This is obvious when you consider the size of the input and output of a hash function. The output is always the same size (e.g., 128-bit) but the input can be of any size (e.g., 16 MByte).

This hash value therefore can be used to determine the integrity of data. Computing the hash of a data and comparing the result with a previously calculated hash result can be used to identify whether data has been tampered with. Figure 2 shows how this hash function process on documents can be accomplished in SIMBA. Using this approach, we will store a SHA hash of the data on the Blockchain, rather than the data itself, which will make every message a fixed length of 256 or 512 bits. The SHA-256 hash algorithm can be used for this purpose [6] but we are looking at both SHA2 and SHA3.

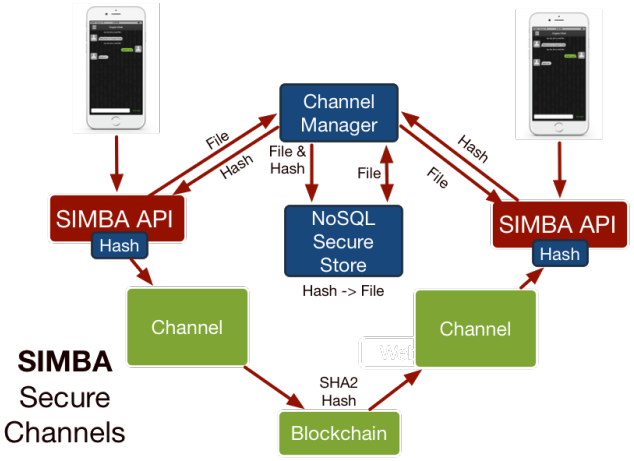

Fig. 2. The SIMBA Blockchain Hash Strategy

VII. SIMBA Current Implementation Status

The Secure Messaging on a Blockchain Architecture (SIMBA) project is working on architecting and designing a mobile peer-to-peer, end-to-end, secure messaging application that uses Blockchain technology. By building on ITAMCO's Crypto-Chat technology, connecting it to a Web application and a Blockchain API, SIMBA plans to provide a platform for delivering secure messaging and other messaging features including: the ability to interface with smart contracts, send files and send Docker containers securely across the underlying Blockchain infrastructure. The SIMBA mobile phone application and Web dashboard act in a symbiotic manner, mirroring and synchronizing content as the user moves seamlessly between them, resulting in a flexible and powerful decentralized secure messaging infrastructure. The coupling of the mobile phone application and a Web dashboard is an approach that is becoming more common with mainstream messaging applications, such as WhatsApp [30] and PushBullet [31], developing seamless mobile to Web connectivity, along with a host of various other types of collaborative technologies, such as issue tracking software e.g. JIRA [32], customer relationship management e.g. Zoho [33], time management e.g. Toggl [34] and scrum based tools e.g. Trello [35]. Figure 3 shows an overview of the system.

We are building SIMBA from the Fabric implementation due to the rationale described in section III. At the lower layer of the stack, we plan to override the P2P Protocol in Fabric that Google RPC (gRPC [25], which is implemented over HTTP/2 standards and use QUIC instead [36]. It provides many capabilities including bidirectional streaming, flow control, and multiplexing requests over a single connection. Furthermore, it

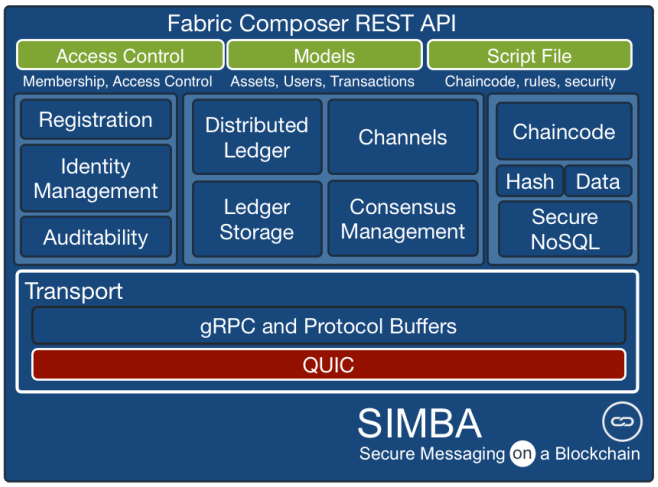

Fig. 3. The SIMBA Architecture

works with existing Internet infrastructure, including firewalls, proxies and security. Fabric itself will be used as it is and provides the services shown in the middle of Figure 3. On the upper layer we will interface with the gRPC backbone using Fabric composer, as described in Section V-E1. This provides us a model based-API to interface with Fabric.

For implementation, we have set up an instance of Fabric on a VM at Notre Dame. The host is a RHEL 7 machine with 4 cores, 8 GB RAM, and 100 GB disk. It has a private IP address and can be accessed using the Notre Dame VPS. We have also configured Fabric to interface with this instance and defined models for the secure transmission of secure messages. We have a demonstration of the core messaging capability implemented so far. We also have implemented the basics of secure Fabric channels for group chat.

In the next phase, we plan to interface using smart contracts to process the data, generate hashes and encrypt the data for insertion into MongoDB, using the hash value as the key lookup for each data file (container). We are also working on configuring the channels so that multiple roles can co-exist so subsets of the group's participants can be sent messages, rather than all of them.

VIII. Conclusion

In this paper, we have investigated the use of Blockchain to design a platform for securely sharing containers and addressing the data scaling issues of such a system if the containers were stored on the Blockchain. To this end, we have described a hybrid system that can generate a hash of the data for the Blockchain transaction and store the hash/encrypted container pair in a NoSQL database as a secondary lookup. We believe this approach can both scale and provide the distributed ledger for recording the distributed transactions that represent the sharing of the containers. We also briefly described a use case in ASIC chip design, in which such a system could allow the multiple parties to share their IPs, designs, and control tests privately within their own teams until a broader release can be made.

Acknowledgments

This works has been made possible through the Department of Defense Advanced Research Projects Agency (DARPA) under a SBIR Program, solicitation number DoD SBIR 2016.2, topic SB162-004. The grant number is SB162-004, entitled SIMBA: Secure Messaging on the Blockchain Architecture.

References

[1] B. A. Nosek, G. Alter, G. C. Banks, D. Borsboom, S. D. Bowman, S. J. Breckler, S. Buck, C. D. Chambers, G. Chin, G. Christensen, M. Contestabile, A. Dafoe, E. Eich, J. Freese, R. Glennerster, D. Goroff, D. P. Green, B. Hesse, M. Humphreys, J. Ishiyama, D. Karlan, A. Kraut, A. Lupia, P. Mabry, T. Madon, N. Malhotra, E. Mayo-Wilson, M. McNutt, E. Miguel, E. L. Paluck, U. Simonsohn, C. Soderberg, B. A. Spellman, J. Turitto, G. VandenBos, S. Vazire, E. J. Wagenmakers, R. Wilson, and T. Yarkoni, "Promoting an open research culture," *Science*, vol. 348, no. 6242, pp. 1422–1425, 2015. [Online]. Available: http://science.sciencemag.org/content/348/6242/1422

[2] M. McLennan and R. Kennell, "Hubzero: a platform for dissemination and collaboration in computational science and engineering," *Computing in Science & Engineering*, vol. 12, no. 2, 2010.

[3] B. Ludäscher, K. Chard, N. Gaffney, M. B. Jones, J. Nabrzyski, V. Stodden, and M. Turk, "Capturing the" whole tale" of computational research: Reproducibility in computing environments," *arXiv preprint arXiv:1610.09958*, 2016.

[4] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[5] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," in *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On*. IEEE, 2015, pp. 171–172.

[6] "The Craft Repository," https://craftproject.org/.

[7] "Docker," www.docker.com.

[8] "RancherOS," http://rancher.com/rancher-os/.

[9] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," http://www.bitcoin.org/bitcoin.pdf, 2009.

[10] "Synereo," http://www.synereo.com.

[11] "BlockCypher," https://www.blockcypher.com/dev/bitcoin/#introduction.

[12] "OpenChain," https://www.openchain.org/.

[13] "BitCore," https://bitcore.io/.

[14] "ChromaWay," https://chromaway.com/platform/.

[15] "Blockchain Developer API for Ethereum — BlockCypher," https://www.blockcypher.com/dev/ethereum/.

[16] "Monax," https://monax.io/library/.

[17] "MultiChain," http://www.multichain.com/.

[18] "Hyperledger," https://www.hyperledger.org.

[19] "Sawtooth Lake," http://intelledger.github.io/.

[20] "Iroha Whitepaper," https://github.com/hyperledger/iroha/blob/master/docs/iroha_whitep
.

[21] "Hyperledger Explorer," https://www.hyperledger.org/projects/explorer.

[22] "Cello," GitHub - hyperledger/cello: Read-only mirror of gerrit.

[23] "Fabric," http://fabric-sdk-node.readthedocs.io/en/master/.

[24] "The Go Language," https://golang.org/ .

[25] "GRPC," http://www.grpc.io/ .

[26] "Proto Buffers V3," (https://developers.google.com/protocol-buffers/docs/proto3.

[27] "Fabric Composer," https://github.com/fabric-composer/fabric-composer.

[28] "Fabric Composer Online Tool," https://fabric-composer.github.io/.

[29] "Swagger," https://swagger.io.

[30] "WhatsApp," https://www.whatsapp.com/ .

[31] "PushBullet," https://www.pushbullet.com/.

[32] "Jira," https://www.atlassian.com/software/jira .

[33] "Zoho," www.zoho.com.

[34] "Toggl," www.toggl.com.

[35] "Trello," https://trello.com/.

[36] "QUIC," https://chromium.googlesource.com/external/github.com/google/proto-quic/ .