BONIK: A Blockchain Empowered Chatbot for Financial Transactions

Md. Saiful Islam Bhuiyan*, Abdur Razzak*, Md Sadek Ferdous*, Mohammad Jabed M. Chowdhury†, Mohammad A. Hoque‡, Sasu Tarkoma‡
*Department of Computer Science and Engineering, Shahjalal University of Science and Technology, Sylhet, Bangladesh.
†Department of Computer Science & Information Technology, La Trobe University, Melbourne, Australia
‡Department of Computer Science, University of Helsinki, Helsinki, Finland
Email: saif_lesnar@outlook.com, razzakrana17@gmail.com, sadek-cse@sust.edu, m.chowdhury@latrobe.edu.au, {mohammad.a.hoque, sasu.tarkoma}@helsinki.fi

***Abstract*—Chatbot is a popular platform to enable users to interact with a software or website to gather information or execute actions in an automated fashion. In recent years, chatbots are being used for executing financial transactions, however, there are a number of security issues that must be carefully handled for their wide-scale adoption. Similarly, blockchain technology, with a number of security advantages, has emerged as one of the foundational technologies with the potential to disrupt a number of application domains, particularly in the financial sector. In this paper, we forward the idea of integrating chatbot with blockchain technology in the view to improve the security issues in financial chatbots. More specifically, we present *BONIK*, a blockchain empowered chatbot for financial transactions, discuss its architecture and the design choices that we have adopted for its different aspects. Furthermore, we explore the developed Proof-of-Concept (PoC), evaluate its performance, analyse how different security and privacy issues are mitigated using BONIK.**

***Index Terms*—Blockchain, Chatbot, Financial Chatbot, Financial Transaction, Private Blockchain, Hyperledger Fabric**

I. Introduction

Chatbot is an advanced application of Artificial Intelligence (AI) providing a platform for users to interact with a software or web services in an automated fashion. In recent times, chatbot has emerged as a technology with a wide-scale adoption in the industry, particularly as an alternative for customer care services that require trivial interactions [1]. A recent study finds that 74% of users prefer to engage with chatbots for answers to simple queries [2]. The increased adoption of chatbot has resulted its market size to grow to an estimated $2.6B$ USD in 2019 with a forecast to increase to $9.4B$ USD by 2025 [3]. Following this trend, the service industry is exploring the possibility for financial chatbots to facilitate financial transactions in a seamless fashion [4]. For example, WeChat, a prominent Chinese message app with chatbot facility, introduced a fund transfer facility in their messaging platform [5]. However, transactions are much more sensitive than services answering to mere trivial queries. Therefore, such chatbots must guarantee a number of security and privacy properties, such as confidentiality, integrity, authenticity, availability, control and transparency of financial transactions [6]–[8]. Also, relying on a single entity to transfer funds introduces a single point of failure. These issues must be addressed before a wide-scale adoption of such chatbots.

In recent years, Blockchain technology (blockchain, in short) has emerged as one of the fundamental technologies with the potential to disrupt a number of application domains [9]. Blockchain offers a number of advantages such as immutability of data and code, distributed consensus mechanism, data provenance and transparency [10], which have the potential to effectively tackle many security issues mentioned above for financial chatbots. Even though a few existing works (e.g. in [6]–[8]) explored different security and privacy issues in chatbots including WeChat, an effective solution is still at large. In this paper, we present *BONIK*, a blockchain empowered chatbot for financial transactions that effectively addresses many security issues involving a financial chatbot. Using BONIK, one can execute financial transactions in a secure and privacy friendly way by just interacting with a chatbot. In this paper, we present its architecture, protocol flow, usages and different other aspects.

Contributions: The main contributions of the paper are presented below:

- We formulate a number of functional, security and privacy requirements, underpinned by a rigorous threat model, for a financial chatbot.
- We provide a detailed architecture of BONIK and discuss how we have developed a Proof-of-Concept prototype along with its detailed protocol flow.
- We evaluate its performance, analyse how the developed prototype has satisfied the formulated requirements and explore its advantages and limitations.

Structure: Section II provides a brief background on blockchain and chatbot. Section III introduces the proposal a threat model and requirement analysis. Section IV presents the architecture of BONIK with implementation details. In Section V, the protocol flow of BONIK illustrates its use-case. Section VI evaluates the performance of BONIK under different criteria. In Section VII, we discuss how the design choices for BONIK have helped it to satisfy different requirements and explore its advantages, limitations and the possible future research scopes. Finally, we conclude in Section VIII.

II. Background

In this section we provide a brief background on blockchain technology (Section II-A) and chatbot (Section II-B).

A. Blockchain

Bitcoin is regarded as the first widely-used decentralised digital currency in the world which does not rely on a central entity, such as a central bank, for its creation and circulation [11]. Its main technological breakthrough is due to its underlying mechanism called *blockchain technology*, an example of a distributed ledger shared among a group of Peer-to-Peer (P2P) nodes [9]. The ledger itself maintains an ordered data structure consisting of a number of blocks chained together by cryptographic mechanisms. Each block contains a number of transactions where each transaction enables a user to transact a certain amount of bitcoin to another user/users. Each blocks refer to its previous block using a cryptographic hash which refers to its previous block and so on, thus forming a chain and hence, colloquially known as *blockchain*.

Evolving from the Bitcoin blockchain, a new type of blockchain system has emerged which facilitates the deployment and autonomous execution of computer programs, known as *smart-contracts*, on top of the respective ledger [12]. Being part of the ledger makes smart-contracts and their executions immutable and irreversible, a sought-after property having a wide-range of applications in different domains. In addition, a smart-contract supporting blockchain system has a number of other advantages such as distributed data control, data persistence, data provenance, accountability and transparency. Based on who can access a ledger in a blockchain system, there are generally two types of blockchain:

- **Public blockchain:** In a public blockchain, also known as permissionless blockchain, anyone can join and participate in the network for blockchain governance and transaction creation at any time. Examples of public blockchain systems are Bitcoin [13], Ethereum [14], Litecoin [15], Monero [16] and so on.
- **Private blockchain:** In a private blockchain, also known as permissioned blockchain, only authorised and trusted entities are allowed to participate supporting different levels of permissions and privacy [17]. Examples of private blockchain systems are Hyperledger Platforms [18], Quoram [19] and others.

B. Chatbot

Chatbot (or a *bot* in short) is an application program that can make auditory or textual conversations in real time with users [20]. This is a smart implementation of AI It providing a user-friendly conversational experience for users via multiple channels. It is the upcoming leading technology for vast potential for sales, customer service and marketing. In the next section, we will explore several aspects of a chatbot.

Use-cases. Chatbots are increasingly being used as personal assistants for users. For example, people can converse with a chatbot, ask questions and get things done such as call someone, pay bills, set up a meeting, provide replies to queries based on contextual information such as locations and carry on many other activities that a personal assistant is supposed to do. On March 24, 2017, a 4 years old child *Roman* even saved his mother's life using Siri, a chatbot from Apple [21], [22]. Other popular such chatbots are Google Assistant [23] and Amazons Alexa [24]. Chatbots are also being used to customer care centre so that a customer can query regarding their products and receive instant replies $24/7$.

Classification: Bots can be classified mainly in two types [25]:

- **Text-based:** A user interacts with a text-based chatbot with texts only. Users will query with texts and get answers with texts also. Such chatbots can be of two types. One is a bot that provides fixed options and users need to select an option to interact with. The other is a dynamic chatbot where the bot, on taking random queries from a user, provides a dynamic answer to the user.
- **Voice-activated:** This is the most sophisticated class of chatbots in which users interact with the bot using voice.

Mechanisms: Here, we provide a simple working mechanism of a chatbot. A chatbot consists of a number of components. The front-facing component for a text chatbot is the User Interface (UI) using which a user interacts and submits queries or selects options. A voice-activated chatbot utilises the microphone of the corresponding devices to receive instructions/inputs from the user. An option-based text chatbot is the easiest to develop as it just needs to be equipped to handle a limited number of pre-selected options. Dynamic textual and voice chatbots, on the other hand, need to utilise a number of additional components and advanced algorithms, such as voice translation and Speech To Text Reporter (STTR), to function properly. These chatbots also need to apply other Natural Language Processing mechanisms, such as Part-Of-Speech Tagging [26], Sentiment Analysis [27] to understand the query and finally a suitable output is produced.

Financial chatbots: A financial chatbot is a specific type of chatbot which is used within the financial domains with a wide-range of use-cases, such as allowing users to execute financial transactions, providing financial advises, preventing financial frauds, maintaining a personalised customer service and so on [4], [28]. In the scope of this paper, we restrict out attention only to executing financial transactions.

Security and Privacy issues: Because of their wide usages in different applications domains, chatbots often need to handle sensitive data. Therefore, the security and privacy issues are of great importance for chatbots. Here, we highlight a few of such issues, mostly applicable to financial chatbots, such as *secure authentication, data confidentiality and integrity, system availability, accountability* and *transparency* [6]–[8]. Only authenticated users should be allowed to interact with a chatbot so that they can submit queries for their respective bank account and transact with only their bank account. Data confidentiality and integrity will guarantee that the submitted transaction is accessible by an authorised entity and is secure against any corruption. System availability will ensure unin-

terrupted access while accountability and transparency of the system will help to increase the trustworthiness of the system. The principal data privacy issues mostly arise from the lack of control and consent over any submitted transaction.

III. Threat Modelling & Requirement Analysis

In this section, we present a threat model (Section III-A) and analyse a number of functional, security and privacy requirements (Section III-B) for a blockchain empowered financial chatbot.

A. Threat Modelling

Threat modelling is an integrated process of designing a secure system which is used to identify, communicate, and understand threats and mitigation mechanisms within the context of protecting (IT) assets, financial transactions and chatbot in the scope of this paper. To model threats, we have chosen a well established threat model called STRIDE [29] developed by Microsoft. The STRIDE model encapsulates different security threats which are briefly presented below.

- **T1-Spoofing Identity:** The act of spoofing refers to an adversary using the identity of an authorised user (e.g. a sender or a receiver of a financial transaction) to illegally access or participate in financial transactions.
- **T2-Tampering with Data:** An attacker can try to change a transacting amount in a financial transaction.
- **T3-Repudiation:** An attacker can repudiate certain invalid and illegal actions involving a financial transaction.
- **T4-Information Disclosure:** Private or sensitive data stored in the system is leaked to an attacker unintentionally.
- **T5-Denial of Service:** The system that is used to access the service can be the target of a denial of service attack.
- **T6-Elevation of Privilege:** An attacker might use other attack vectors such as malicious software with potential exploitable vulnerabilities in order to execute transactions without the knowledge of a valid user.

In addition to these, we have considered an additional threat which is crucial for any financial system.

- **T7-Replaying Transactions:** An attacker might capture an old transaction and submit it afterwards, thus launching a replay attack.

The privacy threats mostly emerge from the lack of any privacy control for any user. Based on this assumption, the identified threats are as follows.

- **T8-Explicit Consent:** Each transaction is being carried out with the explicit consent of a user.
- **T9-Lack of control and Transparency:** Users have little control on the way transaction is being carried out.

B. Requirement analysis

In this section, we present a set of functional, security and privacy requirements. The functional requirements capture the core functionalities of the system while security and privacy requirements ensure that they mitigate the identified threats.

Functional Requirements (FR): The requirements are presented below.

F1. Users should be able to execute financial transactions, e.g. balance query and transfer money, through the chatbot easily by interacting with it.
F2. The chatbot should be integrated with a private blockchain infrastructure simulating banking functionalities so that financial transactions can be carried out without any error.
F3. The system should ensure the transparency of the transactional data so that an authorised user can inspect different transactions when required, e.g. during a dispute.

Security Requirements (SR): Next, we present a set of security requirements to address the identified security threats.

S1. The system must ensure that only securely authenticated users can avail this service.
S2. The system must ensure that one user's chatting information is not shared with another user. *S1* and *S2* can combinedly can mitigate *T1* threat.
S3. Any conversational and transactional data must be transferred via networks in a secure manner so as to ensure the confidentiality, integrity and authenticity of a user's transaction data. This can mitigate *T2*, *T3* and *T4* threats.
S4. The system must guard against any Denial of Service attack so as to mitigate the *T5* threat.
S5. The system must take protective measures against any replay attack in order to mitigate the *T7* threat.

Privacy Requirements (PR): Privacy requirements are important to mostly mitigate the privacy threats. We present these requirements below.

P1. The system must ensure that each transaction activity must be carried out only with the user's consent. This mitigates *T6* and *T8* threats.
P2. The system should ensure that a user has full control over any of their transactions. This mitigates *T9* threat.

IV. Architecture & Implementation

In order to effectively tackle the identified security and privacy issues involving a financial chatbot, we propose to develop a chatbot rooted on a blockchain system. A blockchain system is decentralised in nature offering a secure transaction and time-stamping recording mechanism with a strong support for integrity and immutability. Moreover, a smart-contract empowered blockchain system offers the opportunity to deploy complex and immutable logic within a blockchain which can be invoked autonomously using transactions. Towards this aim, we present *BONIK*, a blockchain empowered financial chatbot, in this paper.

A user can interact with BONIK to securely submit transactions and carry out financial activities such as querying for current balance. The blockchain integration enables BONIK to validate each request against pre-defined access control rules codified in smart-contracts and if only validated, user requests are honoured. To achieve these goals, the proposed system must satisfy a number of functional, security and privacy

requirements. The security and privacy requirements must be formulated against a threat model. In the following, we present our threat model and the formulated requirements.

We illustrate the top-level architecture of BONIK in Figure 1. This architecture consists of three main components, namely Chatbot, dApp (Decentralised Application) and the Blockchain platform. Next, we discuss the functionalities of each of these components along with their implementation details and interactions between the components.

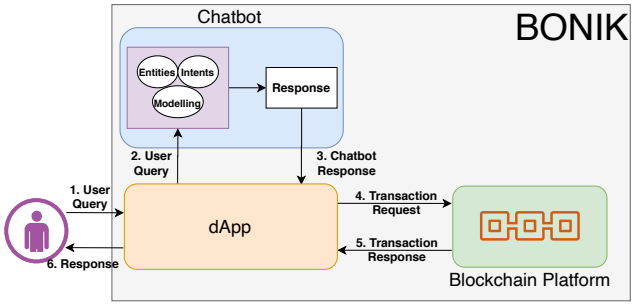


Fig. 1: Top-level architecture and flow in BONIK.

A. Blockchain platform

The chatbot in BONIK is integrated with blockchain platform to facilitate a number of security features and some crucial functionalities. For example, the smart-contract in the platform simulates the functionalities of a financial institution such as a bank. Every user of the system is assumed to hold an account with this bank and every financial activity in the system such as balance query and transfer money is carried out by this bank. In the system, there are two smart-contracts providing business logic to handle user requests. The first one is provided by the system which handles user registration and login while the other is provided by the bank which holds the business logic for financial transactions. This compartmentalisation of smart-contracts provides modularity in the sense if other banks are added in the network, they will just need to deploy and maintain their own smart-contracts in the platform without making much modifications in the system smart-contract.

For deploying the blockchain platform, we have studied different public and private blockchain systems. We have found that public blockchain systems are more secure, however, they are extremely slow, open to all and incur significant amount of cost to process and store data in a smart-contract supported public blockchain (e.g. Ethereum). Because of these reasons, we have chosen to work with a private blockchain system. Currently, Hyperledger Fabric is the most stable and popular private blockchain platform supporting smart-contract facility [30]. It also provides a unique concept of *channel* by which different blockchains can be maintained within the same network, thus creating a layer of privacy between different organisations, a must-have feature in any financial setting so that different activities remain private between different organisations. That this why we have selected to use Hyperledger Fabric as our preferred blockchain system for deployment in this research.

Fabric utilises a number of network entities such as peers, endorsers and orderers. A smart-contract is called a *chaincode* in Fabric terminology which can be invoked using transactions. A user utilises a peer for submitting a transaction which is forwarded to the endorser(s). Each endorser is responsible for validating a transaction by checking if an entity is allowed to perform a certain action in a ledger encoded within the transaction. The validated transaction is then forwarded to the orderer(s). The Orderer creates a block using the transaction and returns the block to the endorsers and peers which is then added to the blockchain and thus, updating the state of the ledger. Consequently, a response is returned to the user. All the entities (peers, orderers and the endorsers) are registered and authenticated via a Fabric specific special entity called *Membership Service Provider* (MSP). This ensures that only authorised entities are allowed in the blockchain network. Figure 2 summarises the flows of activities in Fabric.

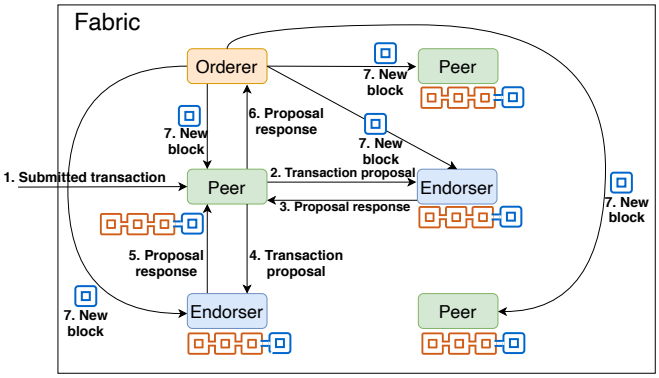


Fig. 2: Flow of activities in Fabric

The chaincode for BONIK has been written in Go where the blockchain platform consists of a varied number of endorsers, peers, orderers. The blockchain platform is deployed using Docker containers where each container assumes the role of one of these entities. In addition, there is an additional container which plays the role of the MSP including the CA (Certificate Authority). These entities are connected via a channel into which the chaincode is deployed. The consensus is based on Kafka which utilises the two orderers for block creation and dissemination as described above. As mentioned earlier, the functionalities of the Bank entity have been simulated within its respective bank chaincode for simplicity. During the starting phase, each user is initialised with 10,000 unit of currency for executing financial transactions.

B. dApp

A dApp (Decentralised Application) interfaces between a blockchain platform and other web (or mobile) applications so that those applications can interact with the blockchain platform. More specifically, a dApp is configured as a web server exposing APIs to web applications as well as is connected to a peer of the blockchain platform. Web applications use these

APIs to submit queries which are translated into blockchain transactions by the dApp. Then, the dApp uses the blockchain API to submit these transactions, via a peer, for invoking a smart-contract on the blockchain and then the usual flows, as illustrated in Figure 2, takes place. A chatbot is essentially a web application, however, has no mechanism to interact with a blockchain platform. To breeze this gap, we need a dApp which creates an interface between these two.

The dApp in BONIK has been developed using Node.js with Express [31], [32]. Node.js is a server-side JavaScript platform that is widely used for creating dApps in the blockchain domain. Express is a web application framework for Node.js which is used for developing web applications using Node.js. Fabric provides the required APIs to interact with any Node.js application. We have developed the dApp in such a way that it is integrated with the chatbot and Fabric APIs and can facilitate the flow described above.

C. Chatbot

Chatbot is the component with which a user interacts, via their browser, to submit different financial queries. BONIK utilises a text-based dynamic chatbot enabling the user to generate random queries and receive the corresponding responses. The chatbot is developed with the help of *Dialogflow* agent that uses strong Natural Language Understanding (NLU) modules in the backbone developed by Google [33]. It functions by taking inputs from a user as a query, processing the data after training itself using machine learning techniques and giving a response in return.

There are three basic elements in Dialogflow: *intents*, *entities* and *training*. An Intent refers to the mapping between the user's query and the agent's response. Each Intent looks like a cluster where seemingly different queries map to a single output, a preset output matched with a high probability. Indeed, a user may ask the system to initiate a transaction in many ways, however, the agent should detect that this is a user's attempt to create a transaction and responds with a single output.

Entities in Dialogflow extract the parameter values from a user's input with natural language. For example, with respect to BONIK, a query for a transaction (e.g. "*send account no 1123158964 1000 unit*") will have several corresponding entities: account number (1123158964) and amount (1000 unit). Training enables the Dialogflow agent to understand what user implies and approach them in a structured way. Machine Learning techniques are used in the backend for this purpose which enable the Dialogflow agent to cluster similar intents and handle entities. For this, Dialogflow uses their own language models. In addition, a developer can feed in their own training data suitable for a particular application. Based on these two, Dialogflow trains itself to handle user queries and generates responses. This model improves dynamically as users converse more and more with the agent which increases its performance and reliability. For our system, a Dialogflow agent named 'Transactional Chatbot' has been created. dApp interacts with this Dialogflow agent by calling the corresponding API with necessary information. The ML model of the agent has been trained with two datasets, namely user dataset and bot dataset, both have been developed by us for BONIK. The user dataset consists of the set of queries that a user can generate. The bot dataset, on the other hand, consists of sentences which are generated in response to any query from the user dataset.

TABLE I: Cryptographic Notations

Notations	Description
K_{U_f}	Public key of the sender.
$K_{U_f}^{-1}$	Private key of the sender.
K_d	Public key of the Dapp.
N_i	A fresh nonce.
$\{\}_K$	Encryption operation using a public key K.
$\{\}_{K^{-1}}$	Signature using a private key K^{-1}.
$H(M)$	SHA-256 hashing operation of message M.
$[]_{https}$	Communication over HTTPS channel K.

Next, a high-level flow involving different components of BONIK is discussed. Once the user is securely logged in (the process is described in the subsequent section), the user can submit different queries to the dApp via the Chatbot UI. These queries are passed to the Dialogflow service and are handled accordingly. When the response is returned from Dialogflow, the dApp processes and parses the response. If additional query is required, the response is returned to the user via the UI. If the response is sufficient to create a Fabric transaction, the dApp converts that response into a transaction which is then submitted to the blockchain platform via the connected peer and then the usual flows takes place. If the transaction is for balance query, upon receiving the response from the chaincode, the dApp displays the result on the UI. If the transaction is for transferring funds and the transfer is successful, an appropriate message is shown to the user. Alternatively, an appropriate error message is shown to the user via the UI if there is any error executing the transaction. The dApp flow in BONIK is illustrated in Figure 1.

V. Protocol Flow

In this section, we present the protocol flow between different components in BONIK. Before we illustrate the protocol flow, we introduce mathematical notations in Table I and data model in Table II.

TABLE II: Data Model

$req \triangleq \langle type, data \rangle$
$resp' \triangleq \langle resp, K_{U_f}, K_{U_f}^{-1} \rangle$
$TYPE \triangleq \langle registration, login, balQuery, transfer \rangle$
$DATA \triangleq \langle regisData, loginData, balData, transferData \rangle$
$regisData \triangleq \langle userName, h \rangle$
$loginData \triangleq \langle userName, h \rangle$
$balData \triangleq \langle userName, accountNum \rangle$
$transferData \triangleq \langle userName, fromAcc, toAcc, amount \rangle$
$string \triangleq \langle string_1, string_2, ..., string_n \rangle$

Data Model: We start with the request (denoted with req in Table II), which is submitted to the blockchain platform. req consists of $type$ and $data$. Here, $TYPE$ denotes the set

of different data types within a request and $type \in TYPE$ whereas, $DATA$ represent the set of corresponding data and $data \in DATA$. Both $TYPE$ and $DATA$ are defined as presented in Table II.

Next, *registration* in type signifies that the corresponding request will be a registration request consisting of the data set denoted with $regisData$ and so on. In $regisData$, $h = H(Password)$ denotes the hash of the provided password and $userName$ denotes the username (identifier) of the user. This implies that a registration request must contain a username and the hash of the password. $loginData$ also has the similar semantic in the sense that a login request must consist of the username and the hash of the provided password.

$balData$ on the other hand is used for balance query and consists of the $userName$ and $accountNum$, implying it must provide the username of the user and the account number to retrieve the balance of the user. Finally, $transferData$ is used for balance transfer requiring username of the user as well as the sender's account number ($fromAcc$), the receiver's account number ($toAcc$) and amount to transfer ($amount$).

Next, we model the functionality of Dialogflow in which a user query is submitted and a set of entities is returned. A user query, in essence, represents the interactions between the user and the chatbot for a meaningful request. For example, a balance transfer query will consist of all required interactions between the user and the chatbot. We use the notations $STRING$, $ENTITY$ to denote the sets of strings (representing a Dialogflow interaction) and entities (as generated by Diaglogflow algorithm). Next, we define a function to model the core functionality of Dialogflow: transforming a string of query into a set of entities.

Definition 1: Let $dFlowModel : string \rightarrow \mathcal{E}$ be the function that transforms a string into a set of entities.

Here, $string \subseteq STRING$ and $\mathcal{E} \subseteq ENTITY$. In other words, $string$ represents the set of all elements from the user and chatbot datasets required to build a meaningful balance query and balance transfer query and is modelled as presented in Table II, where $string_1, string_2, ..., string_n$ represent different elements from the user and chatbot datasets as submitted by the user and the chatbor while interacting for a particular request.

The dApp in BONIK is responsible for handling the returned set of entities ($\mathcal{E}$) which is parsed into corresponding requests, either a balance request or transfer request. We define the following function to model this parsing capability.

Definition 2: Let $parsing : \mathcal{E} \rightarrow req$ be the function that transforms a set of entities into a corresponding request.

Algorithms: We present the algorithms of the system chaincode and bank chaincode in Algorithm 1 and Algorithm 2 respectively.

Whenever the system chaincode (represented as SCC in Algorithm 1) receives a request (denoted with req in the algorithm), its invoke function is initiated. This function retrieves $data$ and $type$ from the request (line 5 and 6) and then invokes any of the other two functions, *regFunc* and *loginFunc*, depending the request type (line 7 to 10). For example, the *loginFunc* encodes the logic for the login functionality whereas the *regFunc* encodes the registration functionality. Once executed, a response is returned (denoted $resp$) back to the dApp (line 14).

Algorithm 1: SCC: // ▷ System Chaincode

```
1  Input: req → the request from the user
2  Output: resp → the chaincode generated response
3  Start
4      function invoke(req)
5          data := req.data;
6          type := req.type;
7          if req.type == login then
8              resp = loginFunc(data);
9          else if req.type == registration then
10             resp = regFunc(data);
11         else
12             resp =BankCC.invoke(req);
13         end
14         send resp back to user;
15     function regFunc(data)
16         uName := data.userName;
17         h := data.h;
18         putState(uName, h); ▷ store into blockchain
19         return TRUE;
20     function loginFunc(data)
21         uName := data.userName;
22         hPasswd = getState(uName); ▷ retrieve from
             blockchain
23         if data.h == hPasswd then
24             return TRUE;
25         else
26             return FALSE;
27         end
```

The bank chaincode (represented as *BankCC* in Algorithm 2) consists of three functions, namely *invoke, balQFunc* and *transFunc*. The the *balQFunc* encodes the algorithm for the balance query operation and finally, the *transFunc* encodes the logic for the balance transfer operation. When $invoke$ receives req, data and type values are retrieved from the request (line 5 and 6 in 2). Depending on the type values, the corresponding function is called with data (line 7 to 10). After executing their code, each of these two functions return a result which is stored in the response (denoted with $resp$, line 12 in the algorithm) and is then returned back to the system chaincode which consequently returns the response back to dApp.

Protocol flow: Now, we present the protocol flow illustrating user interactions with different components in BONIK. To interact with BONIK, a user must register herself following protocol presented in Table III and illustrated in Figure 3. Here, the user submits a username and password in the registration form. The password is hashed using SHA-256 hashing algorithm in the client side. This $userName$ and the hashed password make up $regisData$ where h denotes the

Algorithm 2: BCC: // ▷ Bank Chaincode

1 **Input:** $req \rightarrow$ the request from the user
2 **Output:** $resp \rightarrow$ the chaincode generated response
3 **Start**
4 **function invoke**(req)
5 $data := req.data$;
6 $type := req.type$;
7 **if** $req.type == balQuery$ **then**
8 $resp =$ balQFunc($data$);
9 **else**
10 $resp =$ transFunc($data$);
11 **end**
12 send $resp$ back to SCC;
13 **function balQFunc**($data$)
14 $uName := data.userName$;
15 $acct := data.accountNum$;
16 $balance = getState(acct)$;
17 **return** $balance$;
18 **function transFunc**($data$)
19 $uName := data.userName$;
20 $fromAcct := data.fromAcc$;
21 $toAcct := data.toAcc$;
22 $amount := data.amount$;
23 $fromBalance = getState(fromAcct)$;
24 $toBalance = getState(toAcct)$;
25 **if** $fromBalance > amount$ **then**
26 $fromBalance -= amount$;
27 $toBalance += amount$;
28 $putState(fromAcct, fromBalance)$;
29 $putState(toAcct, toBalance)$;;
30 **return** "TRANSACTION SUCCESSFUL";
31 **else**
32 **return** "TRANSACTION ABORTED";
33 **end**

hashed password. The req in the registration process consists of $registration$ type and $regisData$. As per the protocol, in the first message (denoted with $M1$ in Table III), a user (U_f) sends to the dApp a nonce (N_1), req encrypted with the public key of the dApp (K_d), over an HTTPS channel. dApp decrypts the request using its private key and forwards this request to SCC (M2 in Table III). This is handled in the *regFunction* where the username and the hashed password are extracted and are stored in the blockchain (line 16 to 18 in Algorithm 1). Then a $TRUE$ value is returned to the calling code (the $resp$ variable in line 10), signifying that the user registration response is successful. This response is returned to dApp. Next, dApp generates public and private keys for U_f, K_{U_f} and $K_{U_f}^{-1}$ respectively, using Fabric MSP functionality. This key pair and the response ($resp$) from SCC are combined to create $resp'$ (see Figure 3). Then, this response and its SHA-256 hash ($resp', H(resp')$) are returned to U_f over an HTTPS channel. Then, the user stores her public and private keys in her device for any future correspondence.

Every user must log in before accessing the service. The

TABLE III: Registration protocol

$M1$	$U_f \rightarrow D:$	$[N_1, \{req\}_{K_d}]_{https}$
$M2$	$D \rightarrow SCC:$	N_2, req
$M3$	$SCC \rightarrow D:$	$N_2, resp$
$M4$	$D \rightarrow U_f:$	$[N_1, resp', H(resp')]_{https}$

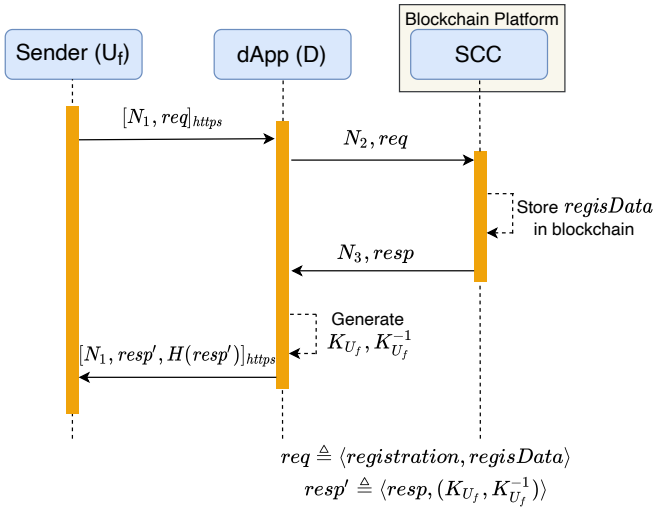


Fig. 3: Registration flow in BONIK.

login protocol is similar to the registration protocol where the user submits the username and password via their browser. These data are encoded into an appropriate req and submitted to dApp which invokes the $loginFunc$ in SCC to handle this request (line 20 to 26 in Algorithm 1). A successful validation will sign in the user to the system. For security, every request and response between the user and the dApp are signed with the sender's private key and are transmitted over HTTPS. Next, we present the protocol flow for the balance transfer from U_f, assuming U_f is already logged in. The protocol is illustrated in Figure 4. Once U_f logs in, a chat interface is loaded in her web browser to interact with the chatbot. U_f submits a query for balance transfer (denoted with $string$ protocol), using this interface, to dApp along with a nonce. It is to be noted, as per the mathematical model, $string$ encodes an interaction between the user and the Dialogflow chatbot consisting of a number of texts required for a meaningful query. We have not shown this interaction in the protocol flow for brevity.

Like before, $string$ is signed with $K_{U_f}^{-1}$ and transmitted over an HTTPS channel. After a successful signature verification, this string along with a secret key (denoted with key in Figure 4) is forwarded to Dialogflow over an HTTPS channel. Every request submitted to Dialogflow API must be registered and authorised beforehand. The secret key is used to validate the authorisation. Then, Dialogflow utilises its $dFlowModel$ function to convert this string to a set of entities ($\mathcal{E}$) which is returned to dApp. dApp utilises its $parsing$ function to convert it to a balance transfer request (consisting of $balQuery$ and $balData$). dApp then invokes SCC with this request which is internally forwarded to the invoke function of BCC. This balance transfer request consequently invokes the $transFunc$ (line 10 in Algorithm 2) where the balances

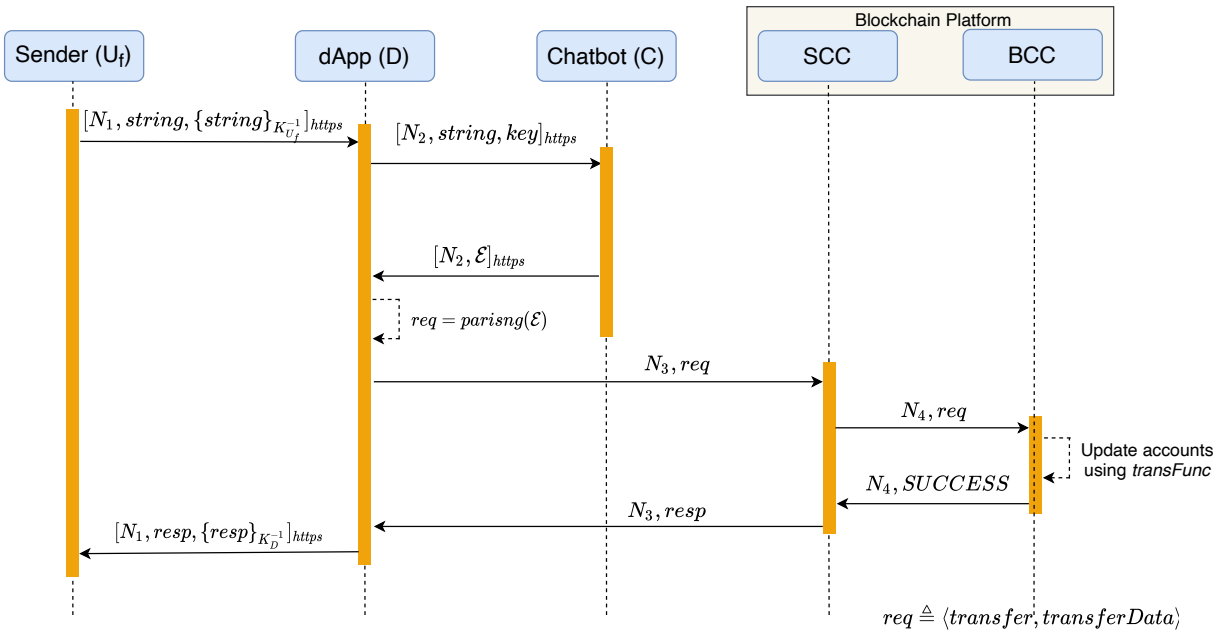


Fig. 4: Balance transfer flow in BONIK.

of the corresponding users' accounts are retrieved from the blockchain and after a validity check (if the user has sufficient balance), accounts are updated with the correct balance and stored in the blockchain (as outlined in line 19 to 29 in Algorithm 2). A successful balance transfer operation will return a "*TRANSACTION SUCCESSFUL*" response, otherwise a "TRANSACTION ABORTED" response will be returned. This response will be returned back to dApp and from there ultimately to the user over HTTPS. The balance query protocol for U_f will be similar and is excluded for brevity.

VI. Evaluation

To evaluate the performance of BONIK, we have utilised Hyperledger Caliper [34], a state-of-the-art blockchain benchmarking tool for Hyperledger blockchain platforms, including Hyperledger Fabric. With BONIK integrated with Caliper, we can measure the performance of its blockchain implementation with a set of predefined network configurations such as the number of entities within the network, the number of simulated users and requests accessing BONIK simultaneously.

The experiment has been carried out in a PC with a Ubuntu 18.04-64 OS and hardware configurations of Intel(R) Core i5-8265U @1.60GHz quad-core CPU, 8 GB DDR4 RAM, 256 GB SSD, 1 TB HDD and 2GB GeForce MX150 Graphics GPU. We have simulated between 10 to 50 users who have submitted different transactions for creating users (registrations), balance query and transfer at varied degrees of rate with three different network configurations consisting of 2 orderers 2 peers (denoted with 2O2P), 2 orderers 4 peers (2O4P) and 2 orderers 6 peers (2O6P).

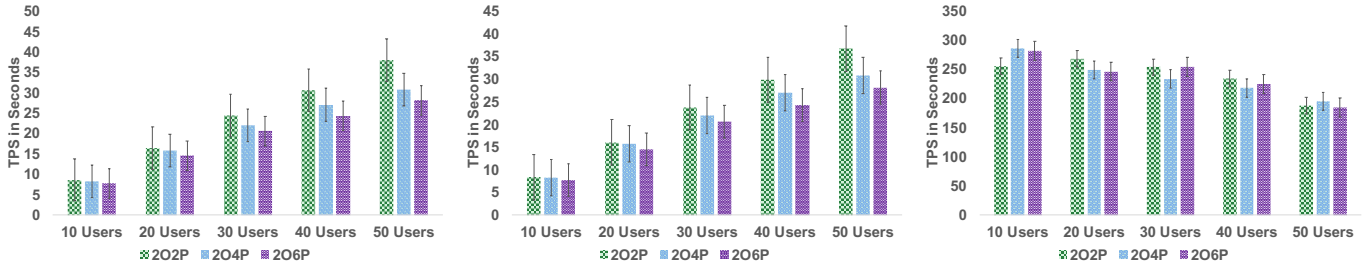
Caliper supports a wide-range of different configurations. Before our main experiments, we have tested these configurations to identify the ideal setup which is the following. The amount of time to wait before creating a batch, the *Batch Timeout* is set as 1s. The maximum message count for a single batch is set as 500 and the transaction rate is set 20 per second. With these configurations, each experiment has been carried out 5 times and the result is then averaged and presented next.

A. User Creation

In Figure 5a the average TPS vs different number of users against different configurations is plotted. Two trends are clear from this figure. The first trend is that TPS increases with the number of users in every configuration. For example, in 2 orderers 2 peers configuration (denoted with 2O2P), the TPS for 10 users is 8.6 which increases to 37.98 for 50 users under the same configuration, a $4x$ increase. This trend is seen in other configuration sets as well. This seems counter-intuitive, however, the underlying reason for this increase is because of the batching mechanism in Fabric in which Fabric waits for a certain number of transaction for putting them in a single block. With more users, more transactions are batched together within a single block, thus resulting in higher TPS. The second trend is that TPS decreases within the same user set with increased number of entities. For example, in 50 users set, TPS decreses from 37.98 to 28.14 for 2O2p and 2O6P respectively. As the number of entities increases in Fabric network, it takes more time for endorsing and creating blocks, resulting in decreased TPS.

B. Balance Transfer

The performance for balance transfer experiment is presented in Figure 5b. It exhibits similar treads, as in Figure 5a, TPS increases as the number of users increases while, within the same user set, TPS decreases as the number of network entities in Fabric increases. Furthermore, in both experiments, TPS remains almost similar. For exmaple, in 2O2P setting for 50 users, the TPS is 37.98 and 36.72 for user creation and



(a) User creation (b) Balance transfer (c) Balance query

Fig. 5: Transaction Per Second (TPS) using BONIK

balance transfer respectively.

C. Balance Query

The result for balance query is presented in Figure 5c. With a maximum TPS of 286.16 for 10 users in 2O2P setting, TPS for balance query is significantly higher than the previous two experiments. The main reason is that balance query is essentially a read operation from the chaincode which can be carried out locally from the Fabric, thus significantly reducing the latency and increasing the TPS. However, as the number of users increases, the TPS tends to to decrease for balance query: from 286.16 for 10 users to 194.9 for 50 users in the same 2O2P setting.

VII. Discussion

In this section, we examine how BONIK has satisfied its different requirements (Section VII-A), discuss its advantages and limitations (Section VII-B) and highlight possible future works (Section VII-C).

A. Analysing Requirements

Here, we explore if BONIK satisfies the formulated requirements of Section III-B.

Functional Requirements: BONIK enables a user to submit financial transactions for balance query and transfer to another account, thereby, satisfying *F1*. The blockchain component in BONIK is based on Hyplerdger Fabric, a private blockchain platform. The chaincode within the platform simulates the banking functionalities in an immutable and error-free manner and hence, BONIK satisfies *F2*. BONIK, underpinned by Fabric, inherits a core property of any blockchain platform, transparency, which enables every authorised entity to validate and verify any transaction. Thus, BONIK satisfies *F3*.

Security Requirements: The BONIK protocol requires every user to be registered and authenticated before submitting any financial transactions. This ensures *S1*. A secure session is maintained for each logged in user so as to create a layer of separation between different users. In this way, no user can access the chatting information of another user and thereby, satisfying *S2*. All data between the user and the dApp as well as between the dApp and Dialogflow are transmitted over secure HTTPS channels which ensures the confidentiality of the data. In addition, every request, except the registration request, from the user is digitally signed with the private key of the user. dApp only accepts such a request if the digital signature is successfully validated. In addition, the communication between dApp and Dialogflow are futher secured with a pre-generated secret key. All these steps combinedly fulfil *S3*. Fabric, being a distributed blockchain platform, offers an effective protection against any DoS attack. Even though it was not created in the current Proof of Concept (PoC), it will be trivial to build a network of dApps in order to guard against the DoS attack against a single dApp so that BONIK services would be accessible even if a single dApp is unavailable due to a DoS attack. This satisfies *S4*. We have extensively used nonces in every single step of our protocol to guard against any replay attack, thereby satisfying *S5*.

Privacy Requirements: Each activity related to any financial transaction, e.g. balance query or balance transfer, requires the user to explicitly digitally sign the transaction with the private key. If the transaction is not signed, the transaction will not be considered as a valid one and thus discarded. This implicitly represents the user consent and a control for the respective transaction, thereby satisfying *P1* and *P2*.

B. Advantages & Limitations

BONIK provides a number of advantages which are discussed next.

- BONIK is the first system to integrate a chatbot with a blockchain platform enabling any user to submit financial transactions using a chatbot in a secure and privacy-friendly fashion.
- BONIK would be beneficial to any financial institutions in order to supplement their existing services by which their users can avail financial services. For example, instead of calling to the customer care centre and being in the call centre queue for unspecified amounts of time, users could use BONIK to initiate financial transactions 24/7, any time of the day. BONIK's utility can be hugely increased by integrating with social network (e.g. Facebook) chatbot services, thereby allowing users to avail financial services from Facebook.
- Being underpinned by Hyperledger Fabric means that BONIK enjoys all the essential benefits of any private blockchain platform, such as decentralisation, im-

mutability of transaction data, resiliency, transparency, automatic code execution and so on. These features incredibly enhance the security of BONIK. Also, the private blockchain ensures that only authorised entity can participate in the blockchain network.

Unfortunately, the current implementation of BONIK has some limitations as presented below:

- The current PoC does not facilitate the transactions between multiple banks. However, this feature can be added by adding additional chanicode, for different banks and modifying the logic of dApp and the algorithms.
- The current PoC utilises a small dataset to train the chatbot with only limited query language.

C. Future Work

In future we would like to explore the following:

- We would like to explore how BONIK can be integrated with Facebook chatbot service so that users can facilitates its service from Facebook.
- We would like to add multiple bank feature in BONIK so that users can transact between different banks.

VIII. Conclusion

In this paper, we have presented BONIK, a blockchain empowered chatbot for financial transactions. At first, we have formulated a set of requirements based on a rigorous threat model for financial chatbots. The architecture of BONIK has been designed in such a way so as to satisfy the formulated requirements and to mitigate the identified threats. We have developed a PoC prototype and described its protocol flow to show its applicability. Furthermore, we have evaluated its performance, analysed its security and privacy issues, advantages and limitations. Using BONIK, one can execute financial transactions within a chatbot. Being rooted in a state-of-the-art private blockchain platform, Hyperledger Fabric, BONIK offers a number of security advantages over any existing financial chatbots. However, its true potential can be enhanced if it can be integrated with the chatbot platform in any social network, thereby laying out the foundation for a wide-scale adoption. Thus, BONIK can be regarded as a pioneering research with a far-reaching potential in this domain.

References

[1] R. McGrath, "How To Improve Customer Service With Chatbots," https://chatbotsmagazine.com/ill-never-buy-from-them-again-using-chatbots-to-avoid-bad-customer-service-e6a967360244, 2020-05-03. Accessed: 2018-08-10.

[2] D. Zaboj, "Key Chatbot Statistics You Should Follow in 2020," https://www.chatbot.com/blog/chatbot-statistics/, 2020-05-06. Accessed: 2018-08-10.

[3] "Key Chatbot Statistics You Should Follow in 2020," https://www.marketsandmarkets.com/Market-Reports/smart-advisor-market-72302363.html, accessed: 2020-06-06.

[4] T. Okuda and S. Shoda, "Ai-based chatbot service for financial industry," Fujitsu Scientific and Technical Journal, vol. 54, no. 2, pp. 4–8, 2018.

[5] "PAYMENT METHODSWeChat Pay Rolls Out Utility To Transfer Funds Between Smartphones," https://www.pymnts.com/news/payment-methods/2019/wechat-pay-rolls-out-utility-to-transfer-funds-between-smartphones/, 2019-09-23. Accessed: 2018-03-17.

[6] J. Bozic and F. Wotawa, "Security testing for chatbots," in IFIP International Conference on Testing Software and Systems. Springer, 2018, pp. 33–38.

[7] S.-T. Lai, F.-Y. Leu, and J.-W. Lin, "A banking chatbot security control procedure for protecting user data security and privacy," in International Conference on Broadband and Wireless Computing, Communication and Applications. Springer, 2018, pp. 561–571.

[8] F. Yan, M. Xu, T. Qiao, T. Wu, X. Yang, N. Zheng, and K.-K. R. Choo, "Identifying wechat red packets and fund transfers via analyzing encrypted network traffic," in TrustCom/BigDataSE 2018. IEEE, 2018, pp. 1426–1432.

[9] M. J. M. Chowdhury, M. S. Ferdous, K. Biswas, N. Chowdhury, A. Kayes, M. Alazab, and P. Watters, "A comparative analysis of distributed ledger technology platforms," IEEE Access, vol. 7, no. 1, pp. 167 930–167 943, 2019.

[10] M. S. Ferdous, M. J. M. Chowdhury, M. A. Hoque, and A. Colman, "Blockchain consensus algorithms: A survey," arXiv preprint arXiv:2001.07091, 2020.

[11] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Manubot, Tech. Rep., 2019.

[12] M. S. Ferdous, F. Chowdhury, and M. O. Alassafi, "In search of self-sovereign identity leveraging blockchain technology," IEEE Access, vol. 7, pp. 103 059–103 079, 2019.

[13] "Bitcoin," https://www.bitcoin.org/, accessed: 2020-07-10.

[14] "Ethereum," https://www.ethereum.org/, accessed: 2020-07-10.

[15] "Litecoin," https://litecoin.org/, accessed: 2020-07-10.

[16] "Monero," https://www.getmonero.org/, accessed: 2020-07-10.

[17] "Public Private Blockchain," https://www.draglet.com/blockchain-applications/private-or-public-blockchain, accessed: 2018-03-17.

[18] "Hyperledger," https://www.hyperledger.org/, accessed: 2020-08-01.

[19] "Quorum Blockchain," https://www.goquorum.com/, accessed: 2020-08-01.

[20] S. A. Abdul-Kader and J. Woods, "Survey on chatbot design techniques in speech conversation systems," International Journal of Advanced Computer Science and Applications, vol. 6, no. 7, 2015.

[21] "Apple Siri," https://www.apple.com/siri/, accessed: 2020-08-02.

[22] "Roman saves her mom:," https://www.cnet.com/news/child-saves-mother-iphone-siri-uk/, accessed: 2018-03-17.

[23] "Google Assistant," https://assistant.google.com/, accessed: 2020-08-02.

[24] "Amazon Alexa," https://alexa.amazon.com/, accessed: 2020-08-02.

[25] "The voice-activated experience and the text-based experience:," https://www.abe.ai/blog/how-secure-are-chatbots/, accessed: 2018-03-17.

[26] E. Brill, "Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging," Computational linguistics, vol. 21, no. 4, pp. 543–565, 1995.

[27] R. K. Bakshi, N. Kaur, R. Kaur, and G. Kaur, "Opinion mining and sentiment analysis," in 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom). IEEE, 2016, pp. 452–455.

[28] J. Tarbal, "Chatbots in Financial Services: Benefits, Use Cases and Key Features," https://www.artificial-solutions.com/blog/chatbots-financial-services-benefits-use-cases, 2020-01-27. Accessed: 2018-08-10.

[29] A. Shostack, Threat modeling: Designing for security. John Wiley & Sons, 2014.

[30] "Hyperledger Fabric," https://www.hyperledger.org/use/fabric, accessed: 2020-03-17.

[31] "Node.js," https://nodejs.org/en/, accessed: 2020-03-17.

[32] "Express js," https://expressjs.com/, accessed: 2020-03-17.

[33] "Google dialogflow," https://cloud.google.com/dialogflow, accessed: 2020-03-17.

[34] "Hyperledger caliper," https://www.hyperledger.org/use/caliper, accessed: 2020-03-17.