CrowdFAB: Intelligent Crowd-Forecasting Using Blockchains and its use in Security

Tara Salman, *Member, IEEE,* Ali Ghubaish, *Graduate Student Member, IEEE,* Roberto Di Pietro, *Senior Member, IEEE,* Mohammed Baza, *Member, IEEE,* Raj Jain, *Fellow, IEEE, and Kim-Kwang Raymond Choo, Senior Member, IEEE*

***Abstract*—Crowdsourcing applications, such as Uber for ride-sharing, enable distributed problem-solving. A subset of these applications is intelligent crowd-forecasting applications, e.g., Virustotal, for malware detection. In crowd-forecasting applications, multiple agents respond with predictions about potential future event outcome(s). These responses are then combined to assess the events collaboratively and act accordingly. Unlike conventional crowdsourcing applications that only communicate information, crowd-forecasting applications need to additionally process information to achieve a collaborative assessment. Hence, they require knowledge-based systems instead of simple storage-based ones for crowdsourcing applications. Most existing crowd-forecasting systems are centralized, leading to the inherent single point of failure and inefficient collaborative assessment. This paper presents CrowdFAB, Crowdsourced Forecasting Applications using Blockchains. We deploy a knowledge-based blockchain paradigm that transforms blockchains from simple storage to knowledge-based systems, thereby achieving crowd-forecasting requirements without centralization. In addition, we formulate a novel reputation scheme that assigns reputations to agents based on their performance. We then use this scheme when making assessments. We implement and analyze CrowdFAB in terms of overhead and security features. Further, we evaluate CrowdFAB for a collaborative malware detection use case, where multiple detectors are involved for crowd forecasting. Results demonstrate CrowdFAB's superior accuracy and other metrics performance compared to other works with the same settings.**

***Index Terms*—Crowdsourcing, Blockchains, Malware detection, Security assessment.**

I. Introduction

Modern problem-solving applications, including traffic analysis, ride-sharing, and job recruiting, are extremely complicated. They require users to collaborate with possibly thousands of others that could be at different levels of expertise. Crowdsourcing has emerged over the last decade as a distributed and efficient problem-solving model to support these applications [1]. Its traditional system consists of three main parties: requesters, agents, and a centralized server. A requester submits a task or a set of tasks that are critical for them. Agents (also referred to as workers in literature) compete to solve this task and submit solutions through the system. The centralized server regulates the communication, selects a "one-to-one matching solution," and grants the corresponding agent(s) a reward. For example, Uber requires riders (requesters) to request a ride, and several drivers (agents) can offer the ride while Uber chooses one driver to be offered to the rider [2]. Other examples include Upwork [3] for hiring tasks and Airbnb [4] for lodging reservations.

An emerging subset of crowdsourcing applications involves forecasting and crowd intelligence, where agents input their predictions as probabilistic responses to specific questions or events. We call these applications as crowd-forecasting applications, where requesters post questions and act upon several responses that might involve risk or uncertainty. An example is the Facebook forecasting application for predicting events such as disease outbreaks and election results [5]. Security assessments in distributed networks are also examples of crowd-forecasting applications. For example, the crowd can solve tracking and detecting malicious applications or network flows in mobile platforms. Thousands of distributed malware detectors (agents) individually evaluate if an application is malicious, and the system will reach a collaborative response about the application [6]. PolySwarm [7] and Virustotal [8] are examples of crowd-forecasting security industries that have emerged in recent years.

Unlike Uber and other crowdsourcing applications, crowd-forecasting applications do not offer a one-to-one matching between requesters and agents. Instead, a collective summary or decision, subsequently named knowledge summary, is reached from multiple answers [9]. Thus, crowd-forecasting models do not only communicate requests and responses but also process responses to get the knowledge summary. This indicates that crowd-forecasting models need knowledge-based systems rather than storage systems needed in crowdsourcing.

Despite their popularity, crowdsourcing and crowd-forecasting models have several weaknesses. Most of these models require a centralized party as a communication channel between requesters and agents [10]. This leads to many challenges, including trust issues, a single point of failure, and potential attack surfaces. Both requesters and agents need to trust the party in performing correct functionalities, which is problematic if the party misbehaves. The party can be a single point of failure, i.e., if the party fails or is compromised,

Tara Salman is with the Department of Computer Science at Texas Tech University, Lubbock, TX, 79409 USA (email: tsalman@ttu.edu)

Ali Ghubaish is with Washington University in Saint Louis, St. Louis, MO 63130 USA (email: aghubaish@wustl.edu)

Roberto Di Pietro is with the Division of Information and Computer Technology (ICT), College of Science and Engineering (CSE), Hamad Bin Khalifa University (HBKU), Doha, Qatar (email: rdipietro@hbku.edu.qa)

Mohamed Baza is with the department of Computer Science, College of Charleston, SC, 29424 USA (email: bazam@cofc.edu)

Raj Jain is with Washington University in Saint Louis, St. Louis, MO 63130 USA (email: jain@wustl.edu)

Kim-Kwang Raymond Choo is with the Department of Information Systems and Cyber Security, University of Texas at San Antonio, San Antonio, TX 78249-0631, USA (email: raymond.choo@fulbrightmail.org)

the whole system fails. The centralization also makes these systems vulnerable to distributed denial-of-service attacks, hijacking, and mischief attacks [11]. Finally, the knowledge summary in crowd-forecasting applications is vulnerable to manipulation by the central server [12].

Blockchain, as a secure, shared, and distributed cutting-edge technology, has the potential to resolve some of the above challenges. It facilitates communication between requesters and agents without needing for a central third party. This resolves the centralization challenge and makes the system less vulnerable to attacks. In addition, users in the system are inherently pseudo-anonymous, enhancing the system's privacy to some extent. As a result, many earlier works focused on building proper blockchain-based crowdsourcing systems. For example, the works in [11] and [13]–[17] have proposed blockchain-based crowdsourcing systems for applications such as crowdsensing, mobile crowd applications, crowdfunding, and image annotations. These works mainly resolved the centralization challenge by eliminating the central authority and replacing it with the blockchains' distributed nature.

None of the earlier works have targeted crowd-forecasting applications and mostly focus on the traditional one-to-one solution for crowdsourcing applications. They use blockchains to gather responses as a storage system. However, as discussed earlier, forecasting applications must process the stored information (responses) and achieve the required "knowledge summary." In other words, *to support blockchain-based crowd-forecasting applications, there is a need to transfer blockchains from simple storage to processing and knowledge-based systems*. Earlier works rarely discuss agents' reputations as an active part of the decision-making process and mostly base the reputations of requesters' evaluation that could be malicious. Moreover, since these concepts do not apply to forecasting applications, they have not been used for security applications. These could be of particular interesting, given the recent interest in crowd-supported security applications.

To that extent, this paper presents CrowdFAB, a novel framework for Crowdsourced Forecasting Applications using Blockchains. CrowdFAB fills the abovementioned gaps and proposes a blockchain-based solution designed explicitly for crowd-forecasting applications that securely and efficiently produce the knowledge summary. We analyze CrowdFAB in terms of security and overhead. We further use the framework for security applications, specifically targeting a distributed malware detection use case. We evaluate the performance in terms of accuracy and other metrics. Results indicate the superior performance of CrowdFAB compared to other existing works in the literature. To summarize, the main contributions of this paper can be outlined as follows:

1) We conceptualize CrowdFAB as a blockchain-based paradigm for crowd-forecasting applications. CrowdFAB inherently resolves the centralization challenges. More importantly, CrowdFAB uses our extended blockchain paradigm, i.e., knowledge-based blockchains, [18], [19][1], which was proposed for collaborative decision-making applications. The paradigm transforms blockchains from storage to knowledge-based systems, guaranteeing a secure and efficient knowledge summary.
2) We extend our earlier proposed reputation scheme in [20] to meet the security requirements and use the CrowdFAB paradigm to provide agents' reputations based on their past performance. This is used to manage agents' contributions to decision-making and eliminate/disregard malicious agents who try to manipulate the knowledge summary.
3) We formulate the CrowdFAB framework that combines knowledge-based blockchains with the proposed reputation scheme. The framework is implemented on top of the Quorum blockchain platform to evaluate its overhead. We emphasize that the framework can be deployed on any blockchain platform, given that it satisfies the system assumptions and application requirements.
4) We analyze the proposed scheme's security, proving it is fraud resilient. This is achieved by two features: resiliency against malicious agents who try to control the knowledge summary and resiliency against malicious nodes who try to downgrade good agents' performance.
5) We validate our approach by building passive malware detection as a crowd-forecasting security assessment use case. Using machine learning (ML) models learned from previously published datasets. We simulate multiple malware agents that can reflect real-world scenarios. We then empirically evaluate the system in terms of several performance metrics. Our results show that CrowdFAB can enhance prediction performance compared to recently published works in the same settings.

The rest of the paper is organized as follows. Section II provides a brief background on blockchains and knowledge-based blockchains. Section III presents the proposed CrowdFAB system, including its components, architecture, system assumptions, threat models, and security goals. Section IV details the CrowdFAB framework and its implementation. Section V evaluates the proposed concepts in terms of performance and security analysis. Section VI presents and evaluates our malware detection case study. Section VII reviews earlier works related to CrowdFAB. Finally, Section VIII summarizes the main results and conclusions from the paper.

II. Background

This section provides a brief background on blockchains and knowledge-based blockchains, which are needed to understand the rest of the paper.

A. Blockchain Technology

A blockchain is a distributed ledger that consists of two main components: a chained database and a distributed network of nodes. The database is a set of time-ordered transactions that are signed and recorded. Several transactions are verified and bundled together to form a block, and agreement is achieved using predefined techniques known as consensus algorithms [21]. Examples of consensus algorithms include

[1]Originally named as probabilistic blockchains. Note that probabilistic blockchains and knowledge-based blockchains are the same. We changed the name for clarity

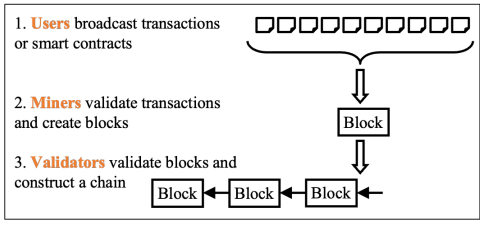


Fig. 1: Illustration of Traditional Blockchains process

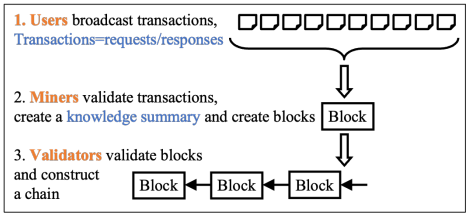


Fig. 2: Illustration of Knowledge-based blockchains process

Proof of Work (PoW) used in Bitcoin and Proof of Stake (PoS) used in Ethereum. Each block has a header that includes a hash value of its predecessor. Thus, all transactions are linked together in a chained database.

The database is maintained by many globally distributed nodes that form the blockchain network. Although all nodes can access blocks and transactions, the transactions cannot be altered or denied once committed to the chain. This novel architecture yields many appealing characteristics, including distributed management, decentralized consensus, immutability, and non-repudiation guarantees [22].

As shown in Fig. 1, one can divide the nodes' roles in a blockchain system into two categories: *users* and *blockchain nodes (miners*[2] *and validators)*. Users are clients that submit transactions as fundamental interactions in the system. They do not necessarily hold the chained database and interact with the system using predefined application programming interfaces (API). Miners validate transactions and form blocks that will eventually be validated by other blockchain nodes in the system. We refer to these other blockchain nodes as validators.

B. Knowledge-based Blockchains

In crowd-forecasting applications, many agents make assessments or predictions for requests. For example, in distributed and collaborative malware analysis, multiple malware inspectors use human intelligence, machine learning, or predefined rules to evaluate whether the submitted software, mobile update, or file is malicious. These evaluations or inspections can be stored in blockchain transactions. Moreover, they can be processed when forming the blocks, and a collaborative knowledge summary about the request can be made [18], [19].

Thus, three requirements that should be in any blockchain-based crowd-forecasting application. First, *transactions for these applications should be of two types: requests and responses.* Request transactions include the event to be assessed; response transactions include the agents' individual assessments. Second, *responses may include probabilistic values or vectors with the risk or uncertainty involved in the assessment*. The probabilities reflect how certain an agent is about its decision. Third, *processing these assessments to produce the knowledge summary should be done within the blockchain process.* To do so, we need to enhance blockchains from simple storage to knowledge-based systems. This is done by processing the included transactions (i.e., generating the knowledge summary) within the blockchain block generation process.

Conventional blockchains (e.g., the technology used for Bitcoin) would not meet the three requirements. This is because transactions in platforms like traditional Bitcoin are of one type and cannot be extended without enhancing the blockchain. Also, they cannot store probabilistic values since all monetary transactions are deterministic. Further, they cannot process these transactions within the blockchain system as the system is designed only to be data storage (ledger).

With smart contracts, one can theoretically make such a transfer and meet all requirements. However, this cannot be done in practice with current smart contracts' implementations due to several limitations. First, existing smart contract languages (e.g., Solidity) do not support floating points as a variable type in their implementations. Thus, probabilistic values cannot be stored unless converted to real values integers, which is undesirable. More importantly, smart contracts processing is very much limited by gas usage and maximum gas limits. Even simple computational operations, such as sorting operations, would result in gas costs that exceed the maximum gas limit, resulting in the failure of smart contracts. This is especially the case as the number of decisions/responses gets large, which is expected for crowd-forecasting applications. Thus, achieving the knowledge summary is impractical using the currently available smart contracts languages or platforms.

Our earlier work has proposed knowledge-based blockchains to resolve this challenge and transform blockchains from storage to processing and knowledge-based engines. Knowledge-based blockchains extend the traditional blockchains in the following ways, see Fig. 2 [18]. Note that this generalized framework highlights how the technology can be extended. The details, e.g., what the request/response transactions contain and how the knowledge summary is generated and validated, are application specifics that will come later in the paper.

1) The users can broadcast the two types of transactions: *request transactions and response transactions*. Request transactions are conventional transactions that include user requests. Response transactions include forecasts and specific probabilities that reflect the risk involved in that forecast. Thus, response transactions are extended conventional transactions that need additional variables to reflect these probabilities.
2) The miners validate the transactions by verifying signatures as in the traditional blockchains. More importantly,

[2] We use the term "miner" for any node that does transaction validation and block generation function. All blockchains have this function, but some do not use this name for nodes responsible for this function, e.g., in some private blockchains.

they combine several related responses into a **knowledge summary** and include this summary in the block. For example, a knowledge summary can be the mode of several response transactions that state if a specific file has malware. This mode, along with the modes of responses for other files, is the knowledge summary and is stored in the block header. Hence, the knowledge summary is generated within the blockchain process (as the system makes the "knowledge" required for forecasting applications out of stored forecasts) and can be accessed by other nodes in the system.

3) Validator or blockchain nodes validate the blocks and construct the chain. While doing so, they verify that the generated knowledge summary correctly summarizes the included response transactions.

In knowledge-based blockchains, the role of users has been replaced by two new roles: *Requesters* and *Agents*. This is done to distinguish requests from responses. However, in terms of nodes' involvement in a blockchain system, requesters and agents are both users. Any node with a valid public key can be a requester or an agent in permissionless (or non-restricted access) blockchains, while the roles can be prespecified in permissioned (or restricted access) blockchains. For the rest of this paper, we refer to various nodes as: *requesters, agents, and blockchain nodes (miners and validators)*. Note that these are roles, and a node can serve multiple roles.

III. CrowdFAB System Model

This section discusses the CrowdFAB system model, architecture, assumptions, and security goals. To simplify the discussion, we focus on malware detection in files as an example application and use it to describe the framework, which can be applied to numerous other applications. Table I presents some notations used throughout this paper. Variables in bold are vectors/matrices. Non-bold variables are scalars. To be consistent, *subscript i* will be used for requests, *subscript j* will be used for agents, *subscript k* will be used for miners, and *subscript l* will be used for validators.

TABLE I: Symbolic notation for the Malware Example

Notation	Explanation
Q	A requester
$\boldsymbol{T}$=$\{T_1, T_2, \ldots, T_t\}$	Set of t sequential request transactions submitted by the requester. i^{th} request here consists of inspecting file$_i$. In a general application, a request would be a question posted by a requester. Each request has an ID that is denoted by $ReqID_i$.
$\boldsymbol{F}_i$=$\{F_{1i},F_{2i}, \ldots, F_{fi}\}$	Set of f features for $file_i$. Agents can use these features to determine their responses
$\boldsymbol{N}_i$=$\{N_1,N_2, \ldots, N_{n_i}\}$	Set of n_i agents that respond to the i^{th} request
$\boldsymbol{D}_i$=$\{D_{i1},D_{i2}, \ldots, D_{in_i}\}$	n_i Responses submitted by n_i agents responding to the i^{th} request
$\boldsymbol{P}_i$=$\{P_{i1},P_{i2}, \ldots, P_{in_i}\}$	Each response D_{ij} includes a probability P_{ij} that the file is malware
C_i	Knowledge summary of responses for $file_i$
R_j	Reputation of $agent_j$
$\boldsymbol{V}$=$\{V_1, V_2, \ldots, V_v\}$	Set of v validator nodes
$\boldsymbol{M}$=$\{M_1, M_2, \ldots, M_m\}$	Set of m miners
G	The summary function specified by the requester to summarize responses

A. CrowdFAB System Components and Architecture

In a CrowdFAB-based malware detection system, requesters are individuals posting requests for file inspection. Without loss of generality, we use only one requester Q in the context of this paper. Each request consists of several file features $\boldsymbol{F}_i$=$\{F_{1i}, F_{2i}, ..., F_{fi}\}$ that can be used by the agents to identify if the file is malicious or not. These features will be included in a **request transaction** $\boldsymbol{T}_i$. Similar to traditional blockchain transactions, these transactions include the requester's signature, timestamp, and feature data.

Agents, identified by $\boldsymbol{N}$=$\{N_1, N_2, ...,N_{n_i}\}$, inspect T_i submitted by the requester Q. n_i here refers to the number of agents that respond to the i^{th} request, T_i. For malware detection, these agents are the security experts who can inspect a file for malware possibility based on predetermined rules, ML models, or any other inspection strategy. Agent *j* submits his/her response transaction D_{ij} consisting of probability P_{ij} that this file is malware along with other conventional variables such as signatures and timestamps. Thus, each T_i results in a response transaction set $\boldsymbol{D}_i$=$\{D_{i1}, D_{i2}, ..., D_{in_i}\}$ along with a set of probabilities $\boldsymbol{P}_i$=$\{P_{i1}, P_{i2}, ..., P_{in_i}\}$. We allow all willing agents to participate in the request. Based on their performances, each agent gets a reputation R_j that determines the "goodness" of its predictions. Note we can have an array of reputations $\boldsymbol{R}_j$, one for each topic or expertise. For example, an agent can be an expert, i.e., has a high reputation in financial forecasting, but not good, i.e., has a low reputation in weather forecasting. However, to simplify this work, we are considering one reputation per agent, i.e., R_j. An agent may respond to all submitted requests, only some requests, or not at all.

Blockchain nodes, including miners and validators denoted by $\boldsymbol{M}$=$\{M_1, M_2, ..., M_m\}$, and $\mathbf{V}$=$\{V_1, V_2, ..., V_v\}$, generate/validate blocks. Compared to traditional blockchains, these nodes here have an additional task of creating/validating a knowledge summary of all responses received.

CrowdFAB architecture consists of two layers: the blockchain layer and the crowd-forecasting layer, as illustrated in Fig. 3. The crowd-forecasting layer consists of the crowd entities, i.e., requesters and agents. These entities connect to the underlying blockchain infrastructure by a "Blockchain API," which can be a web interface, hardware, or software.

The blockchain layer consists of the application sublayer, the consensus sublayer, and the distributed storage sublayer

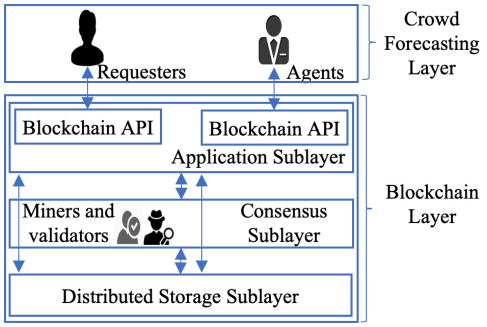


Fig. 3: Layered architecture of CrowdFAB

[23]. The application sublayer includes the blockchain interface with its specific functionalities, including transactions, smart contracts, underlying rules, and chain codes. The consensus sublayer consists of the blockchain nodes that agree on one consistent state of the blockchain database. These two sublayers, i.e., the consensus sublayer and the application sublayer, perform the logical operations of CrowdFAB, including collecting requests and responses and achieving the knowledge summary. The distributed storage sublayer is the shared layer that includes the database system stored on the nodes that maintain the blockchain database. All sublayers are connected to maintain the blockchain functionalities and provide a secure and efficient underlying infrastructure for the crowd-forecasting layer.

B. System Assumptions

CrowdFAB has a few system assumptions for the applications that can use the proposed concepts. These assumptions include passive processing and delay constraints, no monetary rewards, enough storage and processing powers, and that most participants are efficient.

Passive processing and delay constraints: Due to the distributed nature of the blockchains, meeting real-time requirements and processing is still a significant challenge in blockchain-based solutions. Real-time crowdsourcing is a considerable research challenge to date [24]. Thus, *we assume non-real-time and passive crowd-forecasting security applications in the context of this paper*. As will be discussed, malware detection could passively analyze files to find encountered breaches or malicious activities.

Majority efficient participants: *CrowdFAB also assumes that the majority of agents and blockchain nodes are honest*. Dishonest members are a minority. This assumption allows for boosting the reputations of honest agents and makes the decision-making task more efficient. Dishonest or incompetent agents have their reputations reduced. Note that we do not require that the agents make correct responses all the time but that the majority of them make correct responses every time.

No monetary rewards: *We also assume no monetary rewards and limit the rewards in terms of reputation and popularity in the system*. We argue that reputations and popularity benefits are as critical as monetary rewards in decision-making systems. Further, most existing crowd-forecasting systems are free of charge, especially those targeting security applications and malware detection such as Virustotal [8].

Enough storage and processing powers: Finally, *we assume that blockchain nodes have enough storage/processing power to perform blockchain operations and hold the chain when needed*. In addition, they should have enough storage for agents' reputations and computational power to calculate the summary. This assumption applies to all nodes in the blockchain layer, while nodes in the crowd-forecasting layer do not necessarily need these capabilities.

For the security of the CrowdFAB, we assume that the underlying blockchain uses elegant cryptographic techniques for transactions and block signatures. In other words, we assume the blockchain is secure against crypto-breaking attacks that try to change the signature or transaction data while keeping valid signatures. Elliptical curve cryptography (ECC) is used in most blockchain platforms, which is difficult to break [25]. *We also assume that nodes' private keys are securely stored and resilient against hardware attacks*.

For the privacy of agents and requesters participating in the system, *we only require pseudo-anonymity as provided by the underlying blockchain*. Providing fully private and encrypted data is not an objective of this paper.

C. Threat Assumptions

We consider *malicious agents who try to maximize their reputations by manipulating the knowledge summary* to match their response. This can be done either individually or collaboratively with other agents. Individually, they can provide one response that manipulates the knowledge summary if an inefficient summary function is used. For example, suppose the summary function used to summarize responses and generate knowledge summary is the "max" function (taking the maximum value of $\boldsymbol{P}_i$ as knowledge summary). In that case, a malicious agent j can give a high P_{ij} as a response resulting in the knowledge summary being his/her P_{ij}. In addition, they can send many response transactions for the same request to dominate the summary with their responses. Multiple agents may also collaborate to match their responses that falsify the summary.

Underlying blockchain roles that include miners and validators are also considered in our threat assumptions. We consider *malicious miners/validators who maximize profits by colluding with other requesters or agents to break the system's operations*. They can also individually try to force a non-desirable summary by proposing or validating/adding falsified blocks. Note that here we only care about the threats related to crowd forecasting. Other malicious threats are platform dependent, some of which have been discussed in [26]. However, we do assume correct operations in the case of leader-based consensus algorithms. These operations do not include the generation of a correct knowledge summary, as this is to be validated by other blockchain nodes. However, they include bundling all submitted transactions and not discriminating agent responses by not bundling their transactions. In other words, we do assume otherwise honest miners/validators who are curious to manipulate the knowledgeable summary by generating a falsified one.

It should be noted that we do exclude some threats from the system. First, a malicious agent may maximize its reputation by delaying its response. In this case, the agent can calculate the summary after many responses have been made and use it as its submitted response. These agents are generally called curious agents. Privacy-preserving protocols such as multi-party computations can help to resolve this issue. However, we will not consider this threat due to our earlier privacy assumptions and the non-harmful nature of curious agents.

Further, we do not consider Sybil attacks in our threat assumptions. Attackers can join the network with many accounts and submit false responses in such attacks. This can lead to a falsified knowledge summary as these "fake" agents will dominate the submitted responses and manipulate the summary to their desire [27]. Such an attack can be avoided using identity authentication or monetary bids for each response made. Identity authentication validates the agents' legibility before they can join the system. This leads to permissioned blockchains assumption as each agent will join only by permission that validates its legibility. Monetary bids can be used for permissionless blockchains, which can follow already available blockchain bidding solutions [28]. To generalize CrowdFAB regardless of the underlying blockchain, we do not address this attack and consider it a future direction.

D. Security Goals

Our CrowdFAB framework should be resilient against summary manipulations and reputation manipulation. Formally, we define two security properties followed by an overall security goal as follows:

Definition 1. **Knowledge summary manipulation resiliency.** The system is resilient against knowledge summary manipulations if it can prevent the occurrence of the following conditions:

1) An agent controls the knowledge summary by sending one or multiple false responses
2) Several agents collaboratively agree on one false response that dominates, and thus controls, the knowledge summary
3) Miners/validators manage to force a random falsified knowledge summary that favors an agent.

Definition 2. **Reward manipulation resiliency:** CrowdFAB system rewards agents by increasing their reputations, thus, popularity in the system. The system is reputation manipulation resilient if *its miners and validators do not decrease agents' reputations; therefore, they do not deny individual responses.*

Definition 3. **Full fraud-resiliency:** A full fraud-resilient system is a system that is both summary manipulation resilient and reputation manipulation resilient

For this paper, we target full fraud resiliency as a security goal for CrowdFAB. The discussion on how CrowdFAB meets this goal and other security properties will be in Section V-C.

IV. Proposed CrowdFAB Framework and its Implementation

This section starts with a CrowdFAB overview, summary functions, and reputation scheme. Then, we detail the CrowdFAB framework and its functions. We also highlight the implementation of Quorum-based CrowdFAB, which serves as proof of the framework's feasibility and is later used for overhead analysis.

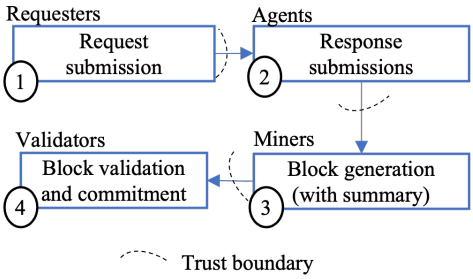


Fig. 4: CrowdFAB data flow

A. CrowdFAB Overview and Data Flow

The proposed data flow for CrowdFAB can be formulated as shown in Fig. 4. Requesters, agents, and blockchain nodes do not necessarily trust and possibly compete with each other. Thus, as shown in the figure, a trust boundary that defines the trust zone between nodes exists. Members of each zone have to ensure that members of other zones are not manipulating the results. The numbers in Fig. 4 indicate the four steps in CrowdFAB. Its main functionalities lie in Steps 3 and 4, as these two steps build the underlying infrastructure for crowd-forecasting applications. In the following, we detail these four steps of the data flow process.

1) *Requests submissions*: In the first step, a requester Q submits the request in a request transaction T_i. This request can be submitted by revoking a smart contract that does the same thing. The request is broadcast to all nodes in the blockchain and will be visible to all agents in the system. In addition, it will be added to the request set $\boldsymbol{T}$. A request may include a deadline that the knowledge summary should meet. If a deadline exists, no further response will be processed after this deadline.
2) *Responses submissions:* Upon receiving T_i or seeing it in the pool of requests, an interested agent N_j, will evaluate the request. Such an evaluation can be done using an ML model built on the agent's own data, based on a prespecified rule-based model, or any decision-making technique that the agent chooses to use. The output of this evaluation is a decision along with a probability. After evaluation, the agent broadcasts a response transaction D_{ij} that includes its response (decision) along with a probability P_{ij}. Alternatively, the response can be submitted by revoking a smart contract that does the same thing. Different agents would assess the same request and return their response transactions, forming a response transaction set $\boldsymbol{D}_i$.
3) *Block and summary generation:* A miner, M_k, forms blocks from the submitted response transactions in this step. Specifically, the miner gets $\boldsymbol{D}_i$ (i.e., $\boldsymbol{D}_i$ for each T_i) and calculates the knowledge summary C_i as will

be discussed later in Section IV-D. This results in a newly formed block that includes response transactions along with the knowledge summary C_i in the headers. For example, if the mean was used as the knowledge summary, the formed blocks can include the mean of the set of probabilities P for all request transactions T included in the block.

4) *Block validation and commitment:* The formed block is broadcast and validated by validators V as discussed earlier. If the block and its knowledge summary are valid, it will be added to the chain, making the summary visible to all nodes. More details about this process is discussed in Section IV-D.

B. Summary Function and its Examples

As discussed, each agent's response is broadcast as a response transaction in CrowdFAB. Each request T_i will have a vector of response transactions $\mathbf{D}_{i,}$. As specified by the requester, a summary function, G, will be used by blockchain nodes to summarize $\mathbf{D}_{i,}$ for a particular request. In Table I, we referred to this knowledge summary earlier as C_i.

$$C_i = G(\boldsymbol{D}_i).$$

To secure this summary against manipulation attacks, G should not be easy to manipulate such that one (or a minority of) agent(s) cannot change the knowledge summary significantly. Examples of a lousy G would be the "minimum" or the "maximum" function of all assessments.

More details about the G function and its examples can be found in [18]. The functions used in this work include the mean, the median, and the mode. The mean summarizes the probability vector $\boldsymbol{P}_i$ constructed from all responses to the ith request by taking their mean. The median summarizes these probabilities by their median value. The mode is the value at which the probability mass function of $\boldsymbol{P}_i$ takes its maximum value. These functions are chosen in our examples to show that even simple functions can result in better decisions than sophisticated individual agents. The functions will be used along with their weighted version, where the weights are agents' reputations calculated by the reputation scheme presented next.

C. Reputation Scheme

CrowdFAB builds its reputation scheme on agents' past performance. Each agent is assigned a reputation that defines how well it has performed so far. Agents whose responses are close to the knowledge summary are considered good and popular or trusted by the system. Thus, these agents get high reputations and are assumed to have a higher chance of making correct predictions in the near future.

Our earlier work put forward seven requirements that should be met in any reputation scheme designed for decision-making [20]. Details on these requirements can be found in that paper; however, we briefly summarize some of them here. The agent's reputation is always between 0 and 1, where 1 is the most reputable and 0 is the least reputable. Each agent starts with a reputation of 0.5, indicating that it is 50% trusted. Further, a mistake made long in the past should not be treated the same as a recent mistake. The increase and decrease in reputation should be configurable, allowing different applications to have different values. However, as a penalty, the rise in reputation after a mistake should be slower than a standard increase. Finally, the decrease after a single incorrect response should be inversely proportional to how long the agent's performance has been excellent. This provides fairness to the good long-lasting agents.

In addition to the previously mentioned considerations, security applications have one additional requirement. Namely, not all errors are equally damaging. Missing a malicious file by classifying it as benign (false negative) is much more harmful than classifying a benign file as malicious (false positive). Therefore, a false negative (FN) prediction should penalize an agent's reputation more than a false positive (FP) prediction.

Based on the above requirements, our proposed reputation function is derived from an extended version of the exponential weighted average function. This function's formulation and properties were discussed in earlier works [20]. Other functions for reputation could follow the reputation scheme in [29] and [30] with some modifications to meet the requirements.

The j^{th} agent's reputation at time t *is denoted by* R_{jt}. However, for simplicity, we omit the subscript j and write it as R_t here. It is updated as follows:

$$R_t = \frac{1}{2}\begin{cases} 0, & p=0, n=0 \\ \beta\frac{-100n}{p+100n} + (1-\beta)(2R_{t-1}-1), & \text{FP is made} \\ \gamma\frac{-100n}{p+100n} + (1-\gamma)(2R_{t-1}-1), & \text{FN is made} \\ \alpha\frac{p}{p+100n} + (1-\alpha)(2R_{t-1}-1) & \text{otherwise} \end{cases} + 0.5$$

where α, β, and γ are configurable parameters for the increase in reputation when a correct response is made, decrease where an FP error is made, and decrease when an FN error is made, respectively. p and n are the numbers of correct and incorrect predictions, where the generated summary or even the ground truth is compared to the agent's decision. An agent response is set to 1 (malware class) if the response's probability is greater than 0.5; otherwise, 0.

Note that we are dealing with 1's and 0's rather than the probability or closeness of the prediction. An alternative approach can compare the probability in the knowledge summary to the individual agent's probability and use fractional values for p and n. This can also be extended to consider different formulations for p and n, where these variables represent probabilities along with how far an agent's decision is from the ground truth (in case the application is forecasts numbers instead of binary 1's and 0's).

Unlike traditional crowdsourcing schemes that rely on third parties, CrowdFAB, and other blockchain-based crowdsourcing applications, should implement the reputation within the blockchain system. In CrowdFAB, R_j is used by blockchain nodes when summarizing responses and validating the summary. It can also be visible to requesters if they want to verify the summary and agents who can use it to enhance their future performance. Therefore, $\boldsymbol{R}$ should either be recorded in the

blockchain or more likely, kept on an off-chain decentralized database with a pointer to it and a hash-like change protection mechanism stored in the blockchain. In either case, the database contains *p, n, and* R_j for each agent. In this work, reputations are stored off-chain in a decentralized database (maintained by nodes), and a hash-like change protection mechanism is stored in the blockchain. They can be updated by blockchain nodes each time a knowledge summary is committed.

D. CrowdFAB Framework

CrowdFAB framework implements the data flow presented in Fig. 4. Steps 1 and 2 are request and response transactions collection, which can be done using the knowledge-based blockchains paradigm [18]. Thus, our focus here for CrowdFAB is on Steps 3 and 4. The framework consists of five functions discussed below, with a pseudo-code provided in Algorithm 1.

Algorithm 1 CrowdFAB Framework

Input: $\boldsymbol{D}_i$ *(all* response transactions, i.e., $\boldsymbol{D}_i$ for each T_i), $\boldsymbol{T}$(Request transactions)
Output: a New valid block is added to the chain

1: **function** COLLECTREQID($\boldsymbol{D}_i$)
▷ Get the requests that have some responses
2: **for** i in $\boldsymbol{D}_i$ **do**
3: **ID.**append($i.ReqID$)
4: **return ID**
5: **function** GETPREVIOUSSUMMARY(**ReqID**)
6: $Blocks \leftarrow$ Get last 50 blocks in chain
7: **for** B in Blocks **do**
8: ID.append($B.Header.Cast.ReqID$)
▷ Cast.ReqID is stored in the block header
▷ Cast is C_i for all i
9: **for** id in **ReqID** **do**
10: **if** id in **ID** **then**
11: **Summaries**.append(id, $Cast.Summary$)
12: **return Summaries**
13: **function** GETREPUTATION($\boldsymbol{D}_i$)
14: **for** d in $\boldsymbol{D}_i$ **do**
15: $Rep_{agent} \leftarrow$ Get R_i for $d.ID$
16: **R**.append(Rep_{agent})
return R
17: **function** CALCULATESUMMARY($\boldsymbol{D}_i$)
18: **R** $\leftarrow$ GETREPUTATION($\boldsymbol{D}_i$)
19: **ReqIDs** $\leftarrow$ COLLECTREQID($\boldsymbol{D}_i$)
20: **PS** $\leftarrow$GETPREVIOUSSUMMARY(**ReqIDs**)
21: **for** i in $\boldsymbol{D}_i$ **do**
22: **if** $i.ReqID$ in **PS** **then**
23: $C_i \leftarrow PS(d.ReqID) + G(R, D)$
24: **else**
25: $C_i \leftarrow G(R, D)$
26: **C**.append(C_i)
27: **return C**
28: **function** UPDATEREPUTATION($newSummary, \boldsymbol{D}_i$)
29: **for** d in $\boldsymbol{D}_i$ **do**
30: **if** $d.vote$==newSummary($d.ReqID$) **then**
31: $p \leftarrow p + 1$ ▷ increase the agent's p value in the off-chain database
32: **else**
33: $n \leftarrow n + 1$ ▷ increase the agent's n value in the off-chain database
34: $R_t \leftarrow$ Calculate R_t from Section V-B
35: **function** CONSTRUCTBLOCK($\boldsymbol{D}_i, \mathbf{T}$)
36: **NewSummary** $\leftarrow$ CALCULATESUMMARY($\boldsymbol{D}_i$)
37: UPDATEREPUTATION(**newSummary,** $\boldsymbol{D}_i$)
38: $newBlock.Header.Cast \leftarrow$ **NewSummary**
39: Conventional Block construction process
40: **function** VALIDATEBLOCK($newBlock$)
41: **ValidSummary** $\leftarrow$ CALCULATESUMMARY(
42: $newBlock.\boldsymbol{D}_i$)
43: UPDATEREPUTATION(**ValidSummary,**
44: $newBlock.\boldsymbol{D}_i$)
45: **if** **ValidSummary**! $=$ $newBlock.Header.Cast$ **then**
46: **return** $False$
47: **else**
48: Conventional validation process

CalculateSummary: A function that takes all response transactions $\boldsymbol{D}_i$ (i.e., $\boldsymbol{D}_i$ for each T_i) in a block and returns their knowledge summary. It first gets the reputations of agents in these response transactions using the GetReputation function discussed next. Following that, it extracts all requests (*i*'s) included in $\boldsymbol{D}_i$ by calling the CollectReqID function and gets the prior summary of these requests if present. To do so, the CalculateSummary function calls the GetPreviousSummary function to check if the request has been summarized in the last 50 blocks, where 50 is an implementation choice that can be adjusted. Finally, the CalculateSummary function generates a new summary using the prespecified summary function, the latest transactions, and the prior summary if it exists.

UpdateReputation: This function is responsible for updating the reputations of participating agents after the summary has been made. It takes the summary and $\boldsymbol{D}_i$. For each agent, it compares the generated summary to the agent's response and calculates the reputation based on Section V-B. The reputations are then updated in the off-chain database along with p and n, if applicable.

GetReputation: A simple function that takes the list of response transactions and returns updated reputations. CalculateSummary uses this function to calculate the next summary that involves agents' updated reputation as weights, as discussed later in the case study.

ConstructBlock: A function run by miners (Step 3 in Fig. 4) to form the block from the available transactions. It takes available request and response transactions and returns a valid block with a valid summary. It first summarizes the response transactions using the CalculateSummary function. Then, it updates agents' reputations using the UpdateReputation function. The new summary, referred to as Cast in algorithm 1

(ConstructBlock and ValidateBlock functions), is added to the new block header. It then follows the conventional blockchain process, bundling all transactions together to generate a new block.

ValidateBlock: A function used by validators to validate the newly generated block, Step 4 in Fig. 4. It takes the new block and returns whether it is valid. It calls CalculateSummary to calculate the summary of transactions included in the new block. Then, it compares the generated summary to the summary in the block header. If the summary is incorrect, it returns false. Otherwise, it follows the normal blockchain process for validation.

Note that **ConstructBlock** and **ValidateBlock** functions are blockchain specific, while any centralized crowd-forecasting can do the others. The focus of Algorithm 1 and its above description is how to do the crowd-forecasting process (i.e., get reputations and calculate summaries) within the blockchain rather than how the blockchain will collect individual responses or requests. Responses and requests can be made by simply sending response/request transactions or triggering smart contract functions that add requests/responses to $\boldsymbol{T}/\boldsymbol{D}_i$.

E. CrowdFAB Framework Implementation

As discussed earlier, we extended the traditional Quorum blockchain to a Quorum-based CrowdFAB. The detailed discussion of Quorum is out of this paper's scope and can be found in [31]. This implementation aims to determine the feasibility of CrowdFAB and the overhead it introduces over traditional Quorum.

In our implementation, we use a mean and standard deviation tuple as the summary along with the ReqID. The ConsturctBlock and ValidateBlock functions extend the conventional block generation and validation processes, respectively. Meanwhile, CalculateSummary and its associated functions have been added as additional functions used by miners and validators. The UpdateReputation and GetReputation functions are implemented to check agent reputations, as discussed in the next section.

V. Performance and Security Analysis

In this section, we first analyze the overhead of Quorum-based CrowdFAB in terms of transaction delay and throughput. We then present an analysis of the proposed reputation scheme. Finally, we discuss CrowdFAB security features proving that the framework is fully fraud-resilient.

A. Analysis of Quorum-Based CrowdFAB

The first analysis measures the overhead of Quorum-based CrowdFAB compared to the conventional Quorum. This is done in terms of throughput and transaction delay (standard transaction in Quorum and response transaction in CrowdFAB). In other words, we measure how many transactions can be made per second, and how fast a transaction can be seen in the blocks.

To measure the performance, we used Blockbench as a blockchain evaluation platform to evaluate Quorum with multiple settings [32]. The chosen consensus algorithm for these analyses is Raft [33], the default consensus algorithm for Quorum. The block generation time, or the time between two blocks, is set to the default 50 milliseconds.

As a leader-based consensus algorithm, Raft, assumes an honest leader or miner who performs correct operations to generate the block. Thus, there is a general assumption to trust the leader. As discussed earlier, in the specific case of leader-based consensus algorithms, we assume that the miner is honest in including all transactions but curious about manipulating the summary. It should be noted that this is not a general assumption or requirement for CrowdFAB.

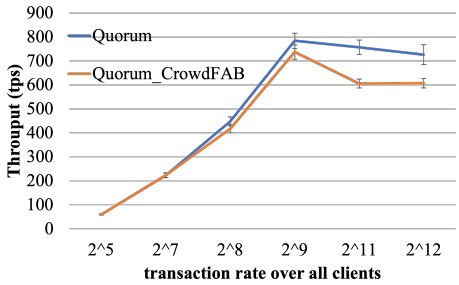
Our setup includes 16 nodes divided as follows: eight blockchain nodes, including one miner, one requester, and seven agents. We have used 16 CloudLab virtual machines [34] as blockchain nodes connected in a peer-to-peer fashion. We use the "smallbank" workload [32] to simulate dummy transactions (standard and response transactions). We vary the transaction rate, i.e., transactions sent per second, and measure the delay and throughput as discussed next. Each experiment has been repeated 35 times to estimate the variability and consistency of the results.

All the measurements are done at the requester node. The throughput is measured by summing the number of transactions available in blocks received in one second. The delay is measured by assuming that the first response transaction (subsequently first knowledge summary) is made in one block after the request has been made. That is, request transactions made in one block will be followed by response transactions in the next block. We measure the delay between the request and the first response received at the requester node.

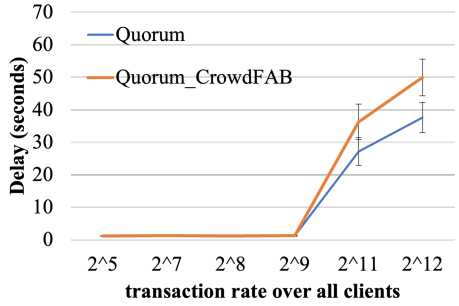
The results from the above experimental designs are presented in Fig. 5. Comparing Quorum-based CrowdFAB to the conventional Quorum performance, as can be seen from Fig. 5a, the throughputs are very close at lower transaction rates with an overhead of less than 1%. The throughputs in both cases increase steadily until they reach a limit of 512, or 2^9, transactions per second (tps). This limit is generally referred to as the knee point, where the system reaches its capacity, and a constant throughput is noticed afterward [35]. As can be seen, after the knee point, CrowdFAB throughput is about 500-700 tps, while it is higher in the conventional Quorum. Thus, CrowdFAB overhead shows up after that knee performance, resulting in a 20% decrease in throughput. This decrease is reasonable given the amount of processing in generating the knowledge summary with a large number of transactions.

As shown in Fig. 5b, the delays behave similarly before the knee point, where they average about 1 second with an overhead of about 1%. After the knee, transaction delays increase rapidly for both Quorum and CrowdFAB. This is reasonable as the delay should be constant and minimal before the knee and increase exponentially after the knee [35]. CrowdFAB shows about a 30% increase in delay, jumping from about 30 seconds to 50 seconds at a 2^{12} transaction rate. Again, such overhead is reasonable due to the amount of processing required in summarizing transactions.

Overall, these results show that CrowdFAB has a throughput overhead of less than 20% and a delay overhead of less than 30%. This overhead is reasonable, given the processing



(a) Throughput analysis



(b) Delay analysis

Fig. 5: Quorum probabilistic blockchain analysis

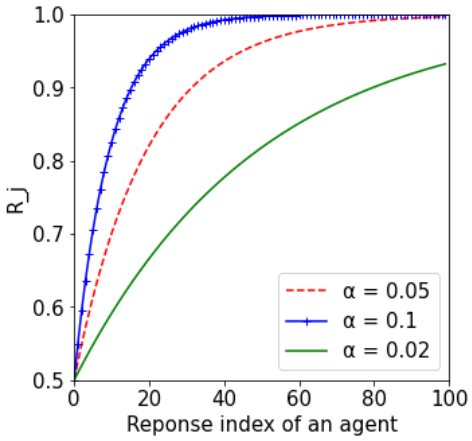
introduced by CrowdFAB. Nevertheless, it can be seen that the delay in generating the summary does not exceed one minute, or 60 seconds, which is typical for crowdsourcing applications [24]. Hence, the overhead that CrowdFAB introduces is minimal compared to its gains in crowd-forecasting applications.

B. Agents Reputation Analysis

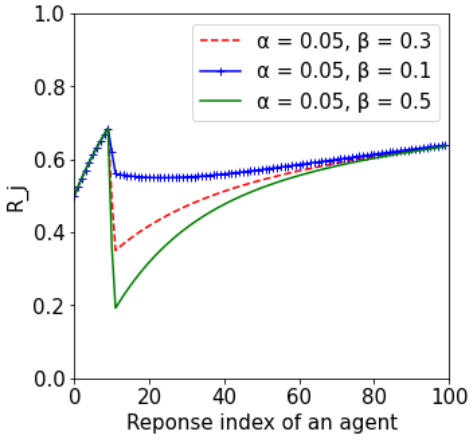
Next, we analyze the proposed reputation scheme and its associated functions to show different competent and incompetent agents' behavior. As discussed before, α, β, and γ are configurable parameters that need to be analyzed. α is the increase in reputation when a correct response is made, β is the decrease where an FP error is made, and γ is the decrease when an FN error is made.

Fig. 6a shows the reputation increase with multiple values of α when no mistake is made. As can be seen, with $\alpha = 0.1$, a rapid increase is observed. An agent can reach a reputation of 1 only after 40 correct responses. With $\alpha < 0.1$, for example, α= 0.05 or α= 0.02, a moderate increase is observed, and an agent can reach a reputation of 1 only after 80 or more correct responses. Thus, choosing a value for α that is less than 0.1 is recommended. The chosen α for this paper is α= 0.05.

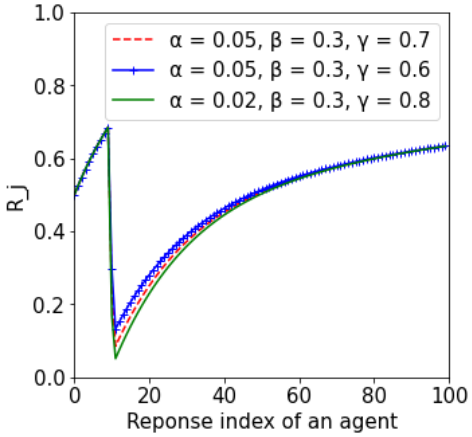
Fig. 6b illustrates the increase/decrease in the reputation when $\alpha = 0.05$, different values of β are chosen, and two FP mistakes are made at indexes 10 and 11. As can be seen, the reputation decreased rapidly (from around 0.7 to 0.2) when $\beta = 0.5$. As discussed before, such a decrease is not desirable for security applications when an FP mistake is made. The figure also shows that with $\beta = 0.1$, the recovery after an



(a) Different α values under no mistake. With $\alpha = 0.1$, a rapid increase is observed. $\alpha < 0.05$ is desirable



(b) Different β values with $\alpha = 0.05$ and two FPs at indexes 10 and 11. The reputation decreases rapidly with $\beta = 0.5$. $0.1 < \beta < 0.5$ is desirable.



(c) Different γ values with $\alpha = 0.05$ and two FNs at indexes 10 and 11.$\gamma > 0.5$ is desirable.

Fig. 6: Reputation analysis with different α, β, and γ settings

FP mistake becomes difficult, which is undesirable. Thus, it is recommended to use a *beta* value of $0.1 < \beta < 0.5$. The chosen β for this work is $\beta = 0.3$.

Similarly, Fig. 6c shows the reputation increase/decrease when $\alpha = 0.05$, different values of γ are chosen, and two FN

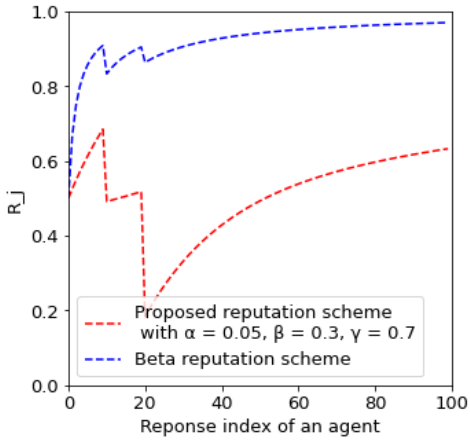


Fig. 7: Reputation changes when $\alpha = 0.05, \beta = 0.3, \gamma = 0.7$. One FP at index 10 and one FN at index 20.

mistakes are made at indexes 10 and 11. As can be seen, with $\gamma > 0.5$, a rapid decrease is observed in all cases. Thus, using a value of γ greater than 0.5 is recommended. The chosen γ for this work is $\gamma = 0.7$.

Fig. 7 shows the reputation increase/decrease when $\alpha = 0.05, \beta = 0.3, \gamma = 0.7$, one FP is made at index 10 (first decrease), and one FN is made at index 20 (second decrease). As can be seen, the second decrease when an FN is made is more than double the first decrease when an FP is made, which is desirable for our security applications. That said, α, β, and γ are configurable parameters that can be set differently for applications based on the different requirements.

Fig. 7 also compares the proposed reputation scheme to the Beta reputation scheme [36], as one of the traditional and most widely adopted reputation formulations used in reputation systems. In terms of formulation, the Beta reputation scheme provides an agent reputation as fellows:

$$R_t = \frac{p+1}{(p+n+2)} \tag{1}$$

where p is the number of correct responses, and n is the number of incorrect responses, no matter if they are FN or FP. As can be seen, the traditional beta reputation does not meet the previously stated reputation requirements as it has a rapid increase in reputation, and the decrease is asymmetric when FN and FP mistakes are made. Also, the decrease is minimal when any mistake is made. Thus, our proposed reputation scheme is more representative and better suit security applications than the traditional Beta reputation scheme. It should be noted that one can extend the Beta scheme to meet the reputation requirements, which would be outside the scope of this paper.

C. Security Analysis of CrowdFAB

CrowdFAB design fulfills our earlier security goals and provides other security properties. We discuss these properties in this subsection.

Knowledge summary manipulation resiliency: According to Definition 1, the knowledge summary can be manipulated if agents can individually or collaboratively send falsified transactions or collaborate with miners or validators to force a random falsified summary. Property 1 will analyze how CrowdFAB is resilient against summary manipulation, assuming an honest majority of agents.

Property 1. CrowdFAB is resilient against summary manipulation attack

Proof. To prove this resiliency, we show that each of the four conditions discussed in Definition 1 cannot be met unless the majority of the crowd, and the blockchain network is compromised.

Using a proper summary function that satisfies the hard-to-manipulate requirement, an agent cannot control the summary by sending one or a few falsified transactions. We have also assumed no Sybil attacks, so an agent cannot generate these falsified responses from multiple accounts, i.e., fake agents. Let's assume there are *n* good agents that respond correctly to a request. These responses were denoted earlier by $\boldsymbol{N}_i$. To flip the summary, a malicious agent must submit at least *n* falsified responses, assuming that the mode is the most straightforward summary function. In other words, to control the summary, a malicious agent needs to control the crowd by falsifying most responses. As *n* gets large, achieving such an attack becomes impractical unless this malicious agent can compromise most other agents' responses. Further, with the use of reputation schemes, as discussed before, the system will make it almost impossible to launch the attack for an extended time. Thus, one malicious agent cannot flip the knowledge summary.

The same reasoning applies to collaborative attack launches, where multiple malicious agents would send falsified responses. Again, as *n* gets large, the number of malicious agents required for the attack to succeed becomes large unless these attacks can compromise the majority of the agents' responses. Also, with proper reputation management, such attacks can be avoided.

Another way to manipulate the summary is by targeting the underlying blockchain security, i.e., by controlling miners or validators. Miners can generate wrong summaries; validators can invalidate a correct summary or validate an incorrect summary. These can be done by CalculateSummary in Algorithm 1, which is called by both miners (ConstructBlock function) and validators (ValidateBlock function). Since the summary is a part of the validation process, a wrong summary would result in an invalid block by the process of ValidateBlock in Algorithm 1. Therefore, the generated block would be dropped, and the summary will not be committed to the chain.

With the immutability feature of blockchains, changing the summary after it has been committed is also impossible unless the majority of blockchain nodes are compromised. Thus, CrowdFAB is resilient against summary manipulation attacks.

Reward manipulation resiliency: According to Definition 2, reputations can be manipulated if miners and validators can unreasonably decrease agents' reputations. Next, we discuss how CrowdFAB is resilient against such manipulation.

Property 2. CrowdFAB is resilient against reputation manipulation attacks.

Proof. Reputation manipulation can happen if miners and validators collaborate to decrease or increase the reputation.

We have discussed in Section IV-C that reputations are stored off-chain in a decentralized database, and a hash-like change protection mechanism is stored in the blockchain. They can be updated by blockchain nodes each time a knowledge summary is committed. Thus, reputation manipulation can happen either by changing the already stored reputations or when updating the reputations.

Since blocks are signed, the hash and blockchains' nature protects the change of already stored reputations, resulting in non-repudiation guarantees. In this case, reputation manipulation cannot happen unless a node manages to manipulate the hash stored in the blockchain, i.e., changing the blockchain data. With public blockchains, this can happen by a "chain fork" that occurs if a valid but modified chain is longer than the healthy chain [37]. With the honest majority assumption, the success probability of a malicious fork is critically low [11]. For Quorum CrowdFAB implementation, one may change the hash if the miner is corrupt. This invalidates our assumptions, so CrowdFAB is resilient against changing the already stored reputations.

Falsified reputation updates lead to invalid blocks by the process of the CrowdFAB framework discussed in Section IV-D. As shown in Algorithm 1, UpdateReputation is called by blockchain nodes when calculating the summary (CalculateSummary function). False reputations by miners result in an incorrect knowledge summary and an invalid block, as discussed in Property 1 and shown by ValidateBlock in Algorithm 1. Therefore, CrowdFAB is protected against falsified reputation updates.

By proving that reputations cannot be changed while stored or updated, we can say that CrowdFAB is resilient against reputation manipulation attacks.

Full fraud-resiliency: By proving that the system is summary and reward manipulation resilient, we can say that the system is fully fraud-resilient, according to Definition 3.

Security against free-riding attacks. A free-riding attack occurs when an agent attempts to increase its reputation without proper responses, i.e., making random predictions. Using our reputation function, incorrect responses result in a substantial decrease in reputation. Further, the slow increase will prevent the agent from affecting the knowledge summary unless they make many good responses in the future, as seen in Fig. 6. Thus, CrowdFAB is resilient against free-riding attacks. Note that an agent can predict this summary from the responses made so far, increasing its reputation without having a proper model. However, this requires the agent to collect responses and predict the summary. Since this action has a minimal effect on the knowledge summary, we do not consider it an attack.

No single point of failure: With the decentralized nature of blockchain-based systems, CrowdFAB guarantees no single point of failure. Many nodes hold the blockchain database, making requests and responses available to query at any time. On the assumption that there are m ($m > 2$) blockchain nodes in CrowdFAB and more than $m/2$ are honest, there exists at least one node that can be accessed for CrowdFAB services. Note that reputations are stored in a distributed database maintained by different nodes; thus, accessing and changing reputations by one node is not possible.

No trusted third party is required: This again comes from the inherent design of blockchain systems. They work without requiring a trusted third party.

VI. Case Study: Passive Malware Detection

This section empirically applies the proposed CrowdFAB framework to a passive malware detection case study. We consider a case where several malware detection agents analyze whether a particular mobile app is malicious. First, we put forward an application scenario for the case study. We then discuss the dataset used, the experimental setup, the evaluation metrics, and the results from these evaluations.

A. Collaborative Malware Detection Application

The growth of malware over the last decade has posed serious security threats, especially to mobile networking and applications. Collaborative malware detection approaches have been discussed extensively in both academia and industry [38]. A famous collaborative malware detection example is Virustotal, a website where a file is uploaded as a request and analyzed by many antivirus products and online scan engines [8]. However, this approach poses a centralization challenge and requires the requester to trust the website to host the services involved. By utilizing CrowdFAB, requesters can submit requests to analyze files or software. Multiple malware detection agents can analyze the request to provide elegant analyses with proper security guarantees. We emphasize that we consider passive malware analysis. This should be distinguished from a real-time malware detection application where the scan is performed online by a central agent or computer.

B. Malware Datasets and Experimental Setup

To empirically evaluate this case study, we have used a previously published Malware dataset named Drebin [39], [40]. We used a portion of the dataset comprising 215 features with 5560 malware apps and 9475 benign apps [41]. This is divided into training and test sets, where the assumption is that the test set will be the requests while the training set is data held at different agents and used to build their models.

To simulate a large number of agents with many experience levels, we have used the training data with two different strategies, i.e., horizontal and vertical partitions. In the horizontal partition, agents can only access a subset of the training dataset to train their models. This reflects the experience level since an "old" agent would have seen more data than a "new" agent. Using this reflects real-world scenarios since no agent can have full or "perfect" data to train on, and just-starting agents are assumed to have minimal training data. For this strategy, agents are randomly given 33%, 66%, and 99% of the training dataset.

The vertical partition allows agents to access only a subset of the features to train individual models. This partition simulates different complexities of agents' models as more features will result in a higher complexity, i.e., expensive or higher-cost models. Thus, some low resources agents train

on only a few features that might not be sufficient. Others, with many resources, would choose to build complex models with a significant number of features. However, it should be noted that we do not mean here that having all features would result in better agents. Instead, this is just a reflection of the agents' model complexity. To simulate this, the number of features agents use is randomized between 1 and 200, whereas the original dataset has 215 features. Generally, an agent with above 100 features performs relatively well for the used dataset, but this comes with a cost of complexity.

Using the above strategies, we simulate 1000 individual agents that train their models based on the provided training set and features. At this stage, we have unified the learning algorithm to the random forest, given that the agents' performance can be randomized with the horizontal and vertical partitions. Random forest performs relatively well in malware detection applications and has a lower computation cost than more sophisticated learning algorithms [42]. After the training, the test set is used as a set of requests for agents to analyze whether a particular application is malicious. We assume that all agents participate in all requests.

Note that simulating the crowd agents using ML algorithms is reasonable considering the current popularity of ML algorithms in decision-making applications, especially malware detection applications. Even the most popular malware detectors, such as Norton 360 [43] and MacAfee [44], use ML to make their decisions internally. Thus, by using ML algorithms, we are reflecting on a single real-world agent (crowd participant) that uses ML learned from its data to predict malware. Using ML alone is insufficient, and the crowd is still needed to make proper and secure responses. This will be proven by comparing our work to two recent works that use ML algorithms [45] and [46]. One may argue that using the new federated learning (FL) ML paradigm resolves the above issues and would be preferred over the crowd. However, as will be seen, crowd forecasting still performs better compared to recent works that use sophisticated FL algorithms [47] and [48].

After the agents make their predictions in response transactions, CrowdFAB summarizes these responses using six different summary functions, as discussed in Section IV-B. These functions are mean, median, and mode in their weightless and weighted forms. As discussed, the mean summarizes the probability vector P_i constructed from all responses to the ith request by taking their mean. The median summarizes these probabilities by their median value. The mode is the value at which the probability mass function of Pi takes its maximum value. The weights here are agents' reputations calculated by the reputation scheme presented in Section V-B and then normalized. For example, a weighted mean is a weighted average of the response probabilities, where an agent's weight is its normalized reputation.

To show the superior performance of CrowdFAB, we will be using four earlier works that have used the same dataset with different ML/FL algorithms [45]–[48]. We will also compare CrowdFAB performance to the average performance of the involved agents and an optimal model. An optimal model trains on the whole dataset and is used to show CrowdFAB's superior performance. Note that this optimal model, expected to perform best, is rarely achievable in real-world settings as none of the agents can access to all training data.

C. Metrics for Evaluation

To evaluate the ML models and CrowdFAB, we have used three metrics: accuracy, F_1 score, and Safety score [49]. Accuracy is the most commonly used ML evaluation metric which measures the percentage of correctly classified samples. F_1 score is the harmonic mean between precision and recall, where the precision is the fraction of positive samples (malware) classified as positives, and the recall is the ability of a model to detect all positive samples. The safety score is our extended accuracy metric that gives weights for each prediction category, i.e., true positive (TP), true negative (TN), false positive (FP), or false negative (FN), based on their associated costs [49]. Mathematically, these metrics are:

$$\text{Accuracy} = \frac{TP + TN}{FP + FN + TP + TN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{F}_1 \text{ score} = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

$$\text{Safety score} = \frac{w_{TP}TP + w_{TN}TN}{w_{FP}FP + w_{FN}FN + w_{TP}TP + w_{TN}TN}$$

For a malware application, TP is a malware sample that has been correctly detected. TN is a correctly classified benign sample. FP is a benign sample that has been categorized as a malware sample. FN is a malware sample that is classified as a benign sample.

The weight in the Safety score is application-dependent. For our case study, FNs have high costs as malware samples are missed. Thus, they should have high weights. FPs are less critical, but they cost more than TP and TN, and their weights should be lower than FN. Similarly, TP should have weights higher than TN. Given these considerations, we set Safety score weights for our application as follows:

$$w_{FN} = 5, w_{FP} = 1, w_{TP} = 0.2, w_{TN} = 0.1$$

D. Results and Analysis

The results of the experimental setup are presented in Table II. To analyze the results, we first look at a simple summary function without reputations (mean, median, and mode). As can be seen, using CrowdFAB summary functions enhances the average prediction performance of individual agents. The mean, which performs the worst for this use case, improves the performance over the average performing individual agent by 20% in terms of Safety score and performs slightly better in terms of accuracy and F_1 score. The mode, which performs the best for this use case, enhances the performance by 7% in accuracy, 8% in F_1 score, and more than 20% in Safety score.

TABLE II: Performance of CrowdFAB malware detection with different summary functions

Algorithm	**Accuracy**	**F_1 score**	**Safety score**
Earlier Works Performance using ML/FL			
ML [45]	97.7	97.12	-
ML [46]	97.24	97	-
FL [47]	-	97.64	-
FL [48]	98.25	97.63	-
Best Agent Performance			
Best model (train on all data)	98.60	98.10	70.07
CrowdFAB Performance			
Average of agents	91.89	92.2	54.56
Mean	95.88	94.85	73.32
Median	97.92	96.47	76.97
Mode	98.67	98.25	74.6
RMean*	97.29	96.51	78.03
RMedian*	97.92	97.27	79.69
RMode*	98.67	98.25	72.47

*Summary function with the use of reputation. e.g., RMean is a weighted mean where weights are agents' reputations

Compared to earlier works, the mode and median have similar performance, while the mean has a lower performance. The accuracy has been slightly enhanced using the mode, while the F_1 score has a slightly lower performance using the median. The mode overperforms the earlier works in all cases. Note that we are not showing Safety score analysis for earlier works since this is a recently proposed metric not been used in both earlier works. Compared to the optimal model, the mode and the median have comparable performance in terms of accuracy and F_1 score and better performance in terms of Safety score. Note that the Safety score is considered the most meaningful metrics for security applications.

More importantly, it can be seen that summary functions with the proposed reputation management outperform, or at least perform similarly to, the same summary functions without reputation management. The mean and the median have better performance, where the enhancement is about 3% for all metrics. This also shows that a weighted mode summary function, or a weighted majority vote [50], has a similar performance to the classic mode, as can be seen from Mode and RMode results. Compared to earlier works and the optimal model, all summary functions with reputation management have good performance in terms of accuracy and F_1 score and even outperform the optimal model in terms of Safety score.

E. Discussion

The results presented in the previous subsection show that the mode, or the majority vote, performed the best with this dataset. However, it should be emphasized again that the summary function is application-dependent. The best function would change from one application to another. That said, our objective from these analyses was not to show the best summary function. Instead, we aimed to show that any summary function overperforms the average agent performance in the system and performs better than earlier works. This indicates that using a crowd-forecasting system, i.e., using the crowd's wisdom, can enhance the prediction performance over a singular almost-perfect model. Further, the strengths of CrowdFAB are not in prediction performance but in its security and decentralization approach. We expect any prediction enhancement that CrowdFAB makes to be minimal and restricted by the best agent in the system.

VII. Related Work

This section discusses some of the earlier works that are most closely related to CrowdFAB.

A. Crowdsourcing Security Applications

Many earlier works addressed security assessment solutions using traditional crowdcasting models. For example, Yang et al. leveraged crowdcasting for malicious user node detection in large-scale social networks [51]. Raff et al. proposed a crowdcasting approach for malware detection from log files [6]. Similarly, Christoforidis et al. discussed a crowdsourcing approach to protect against novel malware [52]. Burgura et al. proposed Crowdraid, an Android malware detection applications that utilizes crowdsourcing for collected malicious applications from several experts [53]. The works in [54], [55] discussed a crowdsourcing approach for network intrusion detection.

As discussed earlier, all these works use centralized models, which impose several centralization challenges. With knowledge-based blockchains and a proper reputation scheme, CrowdFAB advances these works, overcomes the centralization challenges, and provides a proper solution.

B. Blockchain-based Crowdsourcing

With blockchains' appealing characteristics, it is no surprise that the technology has been used for crowdsourcing applications. CrowdBC was the first attempt to build a concrete blockchain-based framework for crowdsourcing applications. It utilized smart contracts to perform traditional crowdsourcing processes and was implemented on the Ethereum platform [11]. ZebraLancer advanced these works by providing privacy and strong anonymity using Zero-Knowledge proofs [56]. In addition, the works in [13], [17], [57]–[63] used other privacy-preserving mechanisms to provide privacy guarantees for crowdsourcing applications.

Other works focused on proposing blockchain-based frameworks for a specific crowdsourcing application or problem. For example, recent work in [64] proposed and implemented a blockchain-based framework for crowdsourcing in 5G-enabled smart cities. Another recent work in [17] built a blockchain-based mobile crowdsourcing framework with a privacy protection mechanism. Fu et al. in [65] proposed BFCRI, a blockchain-based framework for crowdsourcing with the addition of reputations and incentives. BFCRI utilizes agents'

reputations to select capable workers when matching agents to work and uses contracts as an incentive mechanism to attract more workers. Finally, Lai and Zhao focused on providing an accountable and trusty blockchain-based crowdsourcing framework using a verification scheme and a reputation-based trust model [66].

All these works targeted crowdsourcing applications that provide a one-to-one matching solution. CrowdFAB advances these works by targeting forecasting applications.

C. Reputation Schemes in Blockchain-based Crowdsourcing

Reputation schemes are employed across various domains and applications, including blockchain-based crowdsourcing systems. Reputation serves as a metric to measure the reliability of participating entities. It is utilized not only as a measure of trustworthiness but also as an incentive mechanism to motivate engagement in such systems.

In the context of CrowdBC, the pioneering blockchain-based crowdsourcing framework, reputation is based on past direct interactions between agents and users [11]. The reputation of an agent is determined based on miners' confirmation of the correctness of its submitted results. Agents can only participate in crowdsourcing tasks if their reputation surpasses a predefined threshold, which is typically set as the average weight of all workers' reputations. Several extensions have been proposed to enhance the reputation scheme in CrowdBC [63], [65], and [67]–[69]. The reputation models proposed in some of these works, such as those in [63] and [69], incorporate additional factors to refine the reputation calculation. For instance, the reputation model introduced by [63] adds an importance factor assigned by a Certification Authority to assess their reputation. Others, like the reputation model presented in [69], consider feedback similarity factors to distinguish honest agents from malicious ones.

It is worth noting that many of these reputation schemes rely partially on indirect reputation ratings, which may introduce some level of unreliability [70]. Additionally, none of the previous works differentiate between different types of errors made by agents, which is crucial in security applications, as discussed earlier. The proposed reputation scheme in this paper addresses these limitations and fulfills the need for Blockchain-based crowd-forecasting security applications.

D. Blockchain-based Crowd-Forecasting

Prediction markets are a notable application of Blockchain-based Crowd-Forecasting. Existing prediction markets are based on smart contracts between the event creator and the participants based on deadlines to report the final outcomes. Augur is a well-known crowd-forecasting prediction market using Ethereum Blockchain proposed by Peterson et al. [71]. The creators/users of Augur can create and trade in prediction markets where participants can place bets on the outcome of future events, such as forecasting wind power [72]. The final outcome is based on a decentralized oracle that relies on multiple sources to securely transfer real-world results from off-chain to on-chain. Other examples of crowd-forecasting systems are Gnosis [73] and [74].

CrowdFAB differs from these works in providing on-the-go summaries without waiting for the deadline to pass. It also summarizes responses within the blockchain process, thus achieving a distributed and secured "knowledge summary." Further, using CrowdFAB for security applications is novel and has many potentials to resolve the challenges with crowdsourcing security solutions.

E. Blockchain-based Security Assessment

There have been some attempts to explore blockchains in risk assessment security applications. For example, the authors of [75]–[77] discussed how blockchains could be used to build intelligent intrusion detection systems. The Telecooperation group focused on blockchain-based intrusion detection, where nodes exchange alerts concerning their role in the system [75]. The works in [78] and [79] proposed blockchains for malware detection in general cloud malware and portable executable (PE) files. In addition, Gu et al. addressed the problem of detecting and eliminating malicious codes in malware files using blockchains [80].

These works use blockchains as a distributed database. The discussions are theoretical without practical consideration of how the alerts or final decisions should be made. CrowdFAB can advance these works by making on-the-go collaborative assessments as responses appear in the chain. In addition, the use of proper reputation management can provide a novel incentive mechanism to favor competent agents and detect incompetent agents and misbehaving blockchain nodes.

VIII. Conclusions

Despite extensive adaptations of crowd-forecasting models, they suffer several challenges, including centralization, potential attack surfaces, and inefficient summarization. In this paper, we presented the design of CrowdFAB, a novel framework for Crowd-Forecasting Applications using Blockchains. CrowdFAB uses knowledge-based blockchains as a novel paradigm that transform blockchains from storage systems to knowledge and processing systems. Further, it builds a reputation scheme that assigns reputations to agents based on their past performance. This allows requests to be submitted to a crowd of agents who provide their responses without needing a trusted third party. An efficient and secure summarization can be done in a fully decentralized manner.

The framework applies to any risk management or forecasting application, and we specifically targeted security assessment in this paper. We implemented the CrowdFAB framework on top of the Quorum blockchain platform to show the feasibility and efficiency of the proposed approach. Finally, we empirically evaluated the proposed scheme for passive malware detection in a mobile application use case. Our results demonstrated the sustainable performance of the proposed method compared to single-agent detections. This, combined with the added security and decentralization features, shows the superiority of CrowdFAB compared to existing approaches.

We expect CrowdFAB, and generally blockchain-based crowd forecasting, to grow as an alternative to traditional crowd-forecasting models. Future works would investigate

CrowdFAB for other non-security applications, including financial predictions and recommendation systems. In addition, the analysis of CrowdFAB for public blockchains is to be done and compared to Quorum-based CrowdFAB.

ACKNOWLEDGEMENT

This publication was made possible by the NPRP grant # NPRP11S-0109-180242 from the Qatar National Research Fund (a member of The Qatar Foundation), and in part by Prince Sattam Bin Abdulaziz University, Al-Kharj, Saudi Arabia. The findings achieved herein are solely the responsibility of the authors.

REFERENCES

[1] J. Howe, "The rise of crowdsourcing," *Wired magazine*, vol. 14, no. 6, pp. 1–4, 2006.

[2] Wikipedia, "Uber," 2020 (Accessed 8 June 2023). [Online]. Available: https://en.wikipedia.org/wiki/Uber

[3] Wiki, "Upwork," 2020 (Accessed 8 June 2023). [Online]. Available: https://en.wikipedia.org/wiki/Upwork

[4] Wikipedia, "Airbnb," 2020 (Accessed 8 June 2023). [Online]. Available: https://en.wikipedia.org/wiki/Airbnb

[5] S. Perez, "Facebook tests forecast, an app for making predictions about world events, like covid-19," 23 June 2020 (Accessed 8 June 2023). [Online]. Available: https://techcrunch.com/2020/06/23/facebook-tests-forecast-an-app-for-making-predictions-about-world- / events-like-covid-19/

[6] A. Raff, D. Peri, and A. Lotem, "System and methods for malware detection using log based crowdsourcing analysis," Aug 27, 2019, uS Patent 10,397,246.

[7] "Polysawm," (Accessed 8 June 2023). [Online]. Available: https: //polyswarm.io/

[8] Wiki, "Virustotal," 2020 (Accessed 8 June 2023). [Online]. Available: https://en.wikipedia.org/wiki/VirusTotal

[9] M. Benali, A. R. Ghomari, and L. Zemmouchi-Ghomari, "Crowdsourced collaborative decision making in crisis management: Application to desert locust survey and control," in *Computational Intelligence and Its Applications*. Cham: Springer International Publishing, 2018, pp. 533–545.

[10] W. Feng and Z. Yan, "Mcs-chain: Decentralized and trustworthy mobile crowdsourcing based on blockchain," *Future Generation Computer Systems*, vol. 95, pp. 649 – 666, 2019.

[11] M. Li, J. Weng, A. Yang, W. Lu, Y. Zhang, L. Hou, J. Liu, Y. Xiang, and R. H. Deng, "Crowdbc: A blockchain-based decentralized framework for crowdsourcing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 6, pp. 1251–1266, 2019.

[12] N. Quoc Viet Hung, N. T. Tam, L. N. Tran, and K. Aberer, "An evaluation of aggregation techniques in crowdsourcing," in *Web Information Systems Engineering – WISE 2013*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–15.

[13] S. Zou, J. Xi, H. Wang, and G. Xu, "Crowdblps: A blockchain-based location-privacy-preserving mobile crowdsensing system," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 6, pp. 4206–4218, 2020.

[14] V. Jacynycz, A. Calvo, S. Hassan, and A. A. Sanchez-Ruiz, "Betfunding: A distributed bounty-based crowdfunding platform over ethereum," in *Distributed Computing and Artificial Intelligence, 13th International Conference*. Cham: Springer International Publishing, 2016, pp. 403–411.

[15] H. Zhu and Z. Z. Zhou, "Analysis and outlook of applications of blockchain technology to equity crowdfunding in china," *Financial Innovation*, vol. 2, no. 1, p. 29, 2016.

[16] F. Buccafurri, G. Lax, S. Nicolazzo, and A. Nocera, "Tweetchain: An alternative to blockchain for crowd-based applications," in *International Conference on Web Engineering*. Springer, 2017, pp. 386–393.

[17] W. Wang, Y. Wang, P. Duan, T. Liu, X. Tong, and Z. Cai, "A triple real-time trajectory privacy protection mechanism based on edge computing and blockchain in mobile crowdsourcing," *IEEE Transactions on Mobile Computing*, pp. 1–18, 2022.

[18] T. Salman, R. Jain, and L. Gupta, "Probabilistic blockchains: A blockchain paradigm for collaborative decision-making," in *2018 9th IEEE Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*, 2018, pp. 208–211.

[19] R. Jain and T. T. Salman, "Systems and methods for probabilistic blockchains," May 7, 2020, uS Patent App. 16/671,780.

[20] T. Salman, R. Jain, and L. Gupta, "A reputation management framework for knowledge-based and probabilistic blockchains," in *2019 IEEE International Conference on Blockchain (Blockchain)*, 2019, pp. 520–527.

[21] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey," *International Journal of Web and Grid Services*, vol. 14, no. 4, pp. 352–375, 2018.

[22] M. Pilkington, "Blockchain technology: principles and applications," in *Research handbook on digital transformations*. Edward Elgar Publishing, 2016.

[23] M. Ali, J. Nelson, R. Shea, and M. J. Freedman, "Blockstack: A global naming and storage system secured by blockchains," in *Proceedings of the 2016 USENIX Conference on Usenix Annual Technical Conference*, ser. USENIX ATC '16. USA: USENIX Association, 2016, p. 181'194.

[24] N. Nandan, A. Pursche, and X. Zhe, "Challenges in crowdsourcing real-time information for public transportation," in *2014 IEEE 15th International Conference on Mobile Data Management*, vol. 2, 2014, pp. 67–72.

[25] Wikipedia, "Elliptic-curve cryptography," 2020 (Accessed 8 June 2023). [Online]. Available: https://en.wikipedia.org/wiki/Elliptic-curve_cryptography

[26] M. Saad, J. Spaulding, L. Njilla, C. Kamhoua, S. Shetty, D. H. Nyang, and D. Mohaisen, "Exploring the attack surface of blockchain: A comprehensive survey," *IEEE Communications Surveys Tutorials*, pp. 1–1, 2020.

[27] Wikipedia, "Sybil attack," 2020 (Accessed 8 June 2023). [Online]. Available: https://en.wikipedia.org/wiki/Sybil_attack

[28] Y.-H. Chen, S.-H. Chen, and I.-C. Lin, "Blockchain based smart contract for bidding system," in *2018 IEEE International Conference on Applied System Invention (ICASI)*, 2018, pp. 208–211.

[29] G. D. Putra, V. Dedeoglu, S. S. Kanhere, and R. Jurdak, "Trust management in decentralized iot access control system," in *IEEE International Conference on Blockchain and Cryptocurrency*, 2020.

[30] J. Yu, D. Kozhaya, J. Decouchant, and P. Esteves-Verissimo, "Repucoin: Your reputation is your power," *IEEE Transactions on Computers*, vol. 68, no. 8, pp. 1225–1237, 2019.

[31] N. SINGH, "Quorum blockchain ultimate guide," 15 March 2019 (Accessed 8 June 2023). [Online]. Available: https://101blockchains.com/quorum-blockchain-tutorial

[32] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang, "Untangling blockchain: A data processing view of blockchain systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 7, pp. 1366–1385, 2018.

[33] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIX ATC'14. USA: USENIX Association, 2014, pp. 305–320.

[34] "Cloudlab," (Accessed 8 June 2023). [Online]. Available: https: //www.cloudlab.us

[35] R. Jain, *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. John Wiley & Sons, 1990.

[36] R. Ismail and A. Josang, "The beta reputation system," in *Proceedings of the 15th Bled electronic commerce conference*, vol. 5. Citeseer, 2002, pp. 2502–2511.

[37] Wikipedia, "Fork (blockchain)," 2020 (Accessed 8 June 2023). [Online]. Available: https://en.wikipedia.org/wiki/Fork_(blockchain)

[38] P. Yan and Z. Yan, "A survey on dynamic mobile malware detection," *Software Quality Journal*, vol. 26, no. 3, pp. 891–919, 2018.

[39] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket." in *Ndss*, vol. 14, 2014, pp. 23–26.

[40] M. Spreitzenbarth, F. Freiling, F. Echtler, T. Schreck, and J. Hoffmann, "Mobile-sandbox: Having a deeper look into android applications," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, ser. SAC'13. New York, NY, USA: Association for Computing Machinery, 2013, pp. 1808–1815. [Online]. Available: https://doi.org/10.1145/2480362.2480701

[41] S. Yerima, "Android malware dataset for machine learning 2," 2018 (Accessed 8 June 2023). [Online]. Available: https://figshare.com/articles/Android_malware_dataset_for_machine_learning_2/5854653

[42] M. S. Alam and S. T. Vuong, "Random forest classification for detecting android malware," in *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, 2013, pp. 663–669.

[43] P. C. Croft and C. McNally, "Norton 360 antivirus review 2023: Still one of the best," Jun 2023 (Accessed 7 June 2023). [Online]. Available: https://allaboutcookies.org/norton-360-antivirus-review

[44] V. Varadaraj, "The what, why, and how of ai and threat detection," June 2021 (Accessed 7 June 2023). [Online]. Available: https://www.mcafee.com/blogs/internet-security/the-what-why-and-how-of-ai-and-threat-detection/

[45] S. Y. Yerima and S. Sezer, "Droidfusion: A novel multilevel classifier fusion approach for android malware detection," *IEEE Transactions on Cybernetics*, vol. 49, no. 2, pp. 453–466, 2019.

[46] M. S. Rana, S. S. M. M. Rahman, and A. H. Sun, "Evaluation of tree based machine learning classifiers for android malware detection," in *International Conference on Computational Collective Intelligence*, Bristol, United Kingdom, 2018, pp. 377–385.

[47] X. Pei, X. Deng, S. Tian, L. Zhang, and K. Xue, "A knowledge transfer-based semi-supervised federated learning for iot malware detection," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 3, pp. 2127–2143, 2023.

[48] A. Chaudhuri, A. Nandi, and B. Pradhan, "A dynamic weighted federated learning for android malware classification," in *Soft Computing: Theories and Applications*. Singapore: Springer Nature Singapore, 2023, pp. 147–159.

[49] T. Salman, A. Ghubaish, D. Unal, and R. Jain, "Safety score as an evaluation metric for machine learning models of security applications," *IEEE Networking Letters*, vol. 2, no. 4, pp. 207–211, 2020.

[50] Wikipedia, "Weighted majority algorithm (machine learning)," 2023 (Accessed 8 June 2023). [Online]. Available: https://en.wikipedia.org/wiki/Weighted_majority_algorithm_(machine_learning)

[51] G. Yang, S. He, and Z. Shi, "Leveraging crowdsourcing for efficient malicious users detection in large-scale social networks," *IEEE Internet of Things Journal*, vol. 4, no. 2, pp. 330–339, 2017.

[52] C. Christoforidis, V. Vlachos, and I. Androulidakis, "A crowdsourcing approach to protect against novel malware threats," in *2014 22nd Telecommunications Forum Telfor (TELFOR)*, 2014, pp. 1063–1066.

[53] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: Behavior-based malware detection system for android," in *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, ser. SPSM'11. New York, NY, USA: Association for Computing Machinery, 2011, pp. 15–26. [Online]. Available: https://doi.org/10.1145/2046614.2046619

[54] F. Saab, I. Elhajj, A. Kayssi, and A. Chehab, "A crowdsourcing game-theoretic intrusion detection and rating system," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, ser. SAC'16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 622–625. [Online]. Available: https://doi.org/10.1145/2851613.2851933

[55] D. R. Choffnes, F. E. Bustamante, and Z. Ge, "Crowdsourcing service-level network event monitoring," in *Proceedings of the ACM SIGCOMM 2010 Conference*, ser. SIGCOMM'10. New York, NY, USA: Association for Computing Machinery, 2010, pp. 387–398. [Online]. Available: https://doi.org/10.1145/1851182.1851228

[56] Y. Lu, Q. Tang, and G. Wang, "Zebralancer: Private and anonymous crowdsourcing system atop open blockchain," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 853–865.

[57] X. Xu, Q. Liu, X. Zhang, J. Zhang, L. Qi, and W. Dou, "A blockchain-powered crowdsourcing method with privacy preservation in mobile environment," *IEEE Transactions on Computational Social Systems*, vol. 6, no. 6, pp. 1407–1419, 2019.

[58] J. Wang, M. Li, Y. He, H. Li, K. Xiao, and C. Wang, "A blockchain based privacy-preserving incentive mechanism in crowdsensing applications," *IEEE Access*, vol. 6, pp. 17 545–17 556, 2018.

[59] S. Zhu, Z. Cai, H. Hu, Y. Li, and W. Li, "zkcrowd: A hybrid blockchain-based crowdsourcing platform," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 6, pp. 4196–4205, 2020.

[60] S. Zhu, H. Hu, Y. Li, and W. Li, "Hybrid blockchain design for privacy preserving crowdsourcing platform," in *2019 IEEE International Conference on Blockchain (Blockchain)*, 2019, pp. 26–33.

[61] C. Lin, D. He, S. Zeadally, N. Kumar, and K.-K. R. Choo, "Secbcs: a secure and privacy-preserving blockchain-based crowdsourcing system," *Science China Information Sciences*, vol. 63, no. 3, pp. 1–14, 2020.

[62] D. G. Kogias, H. C. Leligou, M. Xevgenis, M. Polychronaki, E. Katsadouros, G. Loukas, R. Heartfield, and C. Z. Patrikakis, "Toward a blockchain-enabled crowdsourcing platform," *IT Professional*, vol. 21, no. 5, pp. 18–25, 2019.

[63] W. Tong, X. Dong, Y. Shen, Y. Zhang, X. Jiang, and W. Tian, "Chchain: Secure and parallel crowdsourcing driven by hybrid blockchain," *Future Generation Computer Systems*, vol. 131, pp. 279–291, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X22000309

[64] L. Tan, H. Xiao, K. Yu, M. Aloqaily, and Y. Jararweh, "A blockchain-empowered crowdsourcing system for 5g-enabled smart cities," *Computer Standards & Interfaces*, vol. 76, p. 103517, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S092054892100012X

[65] S. Fu, X. Huang, L. Liu, and Y. Luo, "Bfcri: A blockchain-based framework for crowdsourcing with reputation and incentive," *IEEE Transactions on Cloud Computing*, pp. 1–15, 2022.

[66] R. Lai and G. Zhao, "Validatorrep: Blockchain-based trust management for ensuring accountability in crowdsourcing," in *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*, 2022, pp. 716–725.

[67] J. Kang, R. Yu, X. Huang, M. Wu, S. Maharjan, S. Xie, and Y. Zhang, "Blockchain for secure and efficient data sharing in vehicular edge computing and networks," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4660–4670, 2018.

[68] X. Zhu, Y. Li, L. Fang, and P. Chen, "An improved proof-of-trust consensus algorithm for credible crowdsourcing blockchain services," *IEEE Access*, vol. 8, pp. 102 177–102 187, 2020.

[69] L. Sun, Q. Yang, X. Chen, and Z. Chen, "Rc-chain: Reputation-based crowdsourcing blockchain for vehicular networks," *Journal of Network and Computer Applications*, vol. 176, p. 102956, 2021.

[70] H. Afzaal, M. Imran, M. U. Janjua, and S. P. Gochhayat, "Formal modeling and verification of a blockchain-based crowdsourcing consensus protocol," *IEEE Access*, vol. 10, pp. 8163–8183, 2022.

[71] J. Peterson and J. Krug, "Augur: a decentralized, open-source platform for prediction markets," *arXiv preprint arXiv:1501.01042*, vol. 507, 2015.

[72] M. Shamsi and P. Cuffe, "Towards the use of blockchain prediction markets for forecasting wind power," in *2020 6th IEEE International Energy Conference (ENERGYCon)*, 2020, pp. 847–852.

[73] GNOSIS, "Gnosis chain," 2023 (Accessed 7 June 2023). [Online]. Available: https://www.gnosis.io

[74] A. Carvalho and M. Karimi, "How blockchain can bring trust and transparency to the payment of crowd forecasters." in *ICIS*, 2020.

[75] J. Huang, L. Kong, H. Dai, W. Ding, L. Cheng, G. Chen, X. Jin, and P. Zeng, "Blockchain-based mobile crowd sensing in industrial systems," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 10, pp. 6553–6563, 2020.

[76] N. Alexopoulos, E. Vasilomanolakis, N. R. Ivánkó, and M. Mühlhäuser, "Towards blockchain-based collaborative intrusion detection systems," in *Critical Information Infrastructures Security*. Cham: Springer International Publishing, 2018, pp. 107–118.

[77] M. Banerjee, J. Lee, and K.-K. R. Choo, "A blockchain future for internet of things security: a position paper," *Digital Communications and Networks*, vol. 4, no. 3, pp. 149 – 160, 2018.

[78] W. Meng, E. W. Tischhauser, Q. Wang, Y. Wang, and J. Han, "When intrusion detection meets blockchain technology: A review," *IEEE Access*, vol. 6, pp. 10 179–10 188, 2018.

[79] A. Malvankar, J. Payne, K. K. Budhraja, A. Kundu, S. Chari, and M. Mohania, "Malware containment in cloud," in *2019 First IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, 2019, pp. 221–227.

[80] J. Gu, B. Sun, X. Du, J. Wang, Y. Zhuang, and Z. Wang, "Consortium blockchain-based malware detection in mobile devices," *IEEE Access*, vol. 6, pp. 12 118–12 128, 2018.

Tara Salman is an assistant professor of computer science at Texas Tech University, Lubbock, Texas, USA. Previously she was a Graduate Research Assistant at Washington University in St. Louis (2015-2021) and a Research Assistant at Qatar University (2012-2015). She received her Ph.D. from Washington University in St. Louis in May 2021, and her B.S. in computer engineering and M.S. degrees in computer networking from Qatar University Doha, Qatar, in 2012 and 2015, respectively. Her research interests span Blockchains, network security, distributed systems, the Internet of Things, and financial technology. She is an author of 1 book chapter, a patent, and more than twenty internationally recognized conferences and journals.

Mohamed Baza is an assistant professor at the Department of Computer Science at the College of Charleston, SC, USA. He received his Ph.D. in Electrical and Computer Engineering from Tennessee Tech University, Cookeville, Tennessee, United States, in Dec. 2020, and B.S. and M.S. degrees in Electrical & Computer Engineering from Benha University, Egypt, in 2012 and 2017, respectively. He also has more than two years of industry experience in information security at Apache-Khaleda Petroleum Company, Egypt. Dr. Baza is the author of numerous papers published in major IEEE conferences and journals, such as IEEE Transactions on Dependable and Secure Computing, IEEE Transactions of Vehicular Technology (TVT), IEEE Transactions on Network Science and Engineering, IEEE Systems Journal and conferences such as IEEE Wireless Communications and Networking Conference (IEEE WCNC), IEEE International Conference on Communications (IEEE ICC), IEEE Vehicular Technology Conference (IEEE VTC). His research interests include Blockchains, cyber-security, machine learning, smart-grid, and vehicular ad-hoc networks.

Ali Ghubaish received a B.S. degree (Hons.) in computer engineering (minor in networking) from Prince Sattam Bin Abdulaziz University, AlKharj, Saudi Arabia, in 2013 and an M.S. degree in computer engineering (minor in networking) from Washington University in St. Louis, MO, USA, in 2017, where he is currently pursuing a Ph.D. degree in computer engineering. From 2013 to 2014, he worked as a Teaching Assistant at Prince Sattam Bin Abdulaziz University. Since 2018, he has been working as a Graduate Research Assistant at Washington University in St. Louis. His research interests include network and system security, the Internet of Things, the Internet of Medical Things, healthcare systems, and unmanned aerial vehicles (UAVs) communications.

Raj Jain is the Barbara J. and Jerome R. Cox, Jr., Professor of Computer Science and Engineering at Washington University in St. Louis. Dr. Jain is a Life Fellow of IEEE, a Fellow of ACM, a Fellow of AAAS, a recipient of the 2018 James B. Eads Award from St. Louis Academy of Science, the 2017 ACM SIGCOMM Life-Time Achievement Award, 2015 A.A. Michelson Award from Computer Measurement Group and ranks among the most cited authors in Computer Science. Previously, he was one of the Co-founders of Nayna Networks, Inc - a next-generation telecommunications systems company in San Jose, CA. He was a Senior Consulting Engineer at Digital Equipment Corporation in Littleton, Mass, and then a professor of Computer and Information Sciences at Ohio State University in Columbus, Ohio. He is the author of "Art of Computer Systems Performance Analysis," which won the 1991 "Best-Advanced How-to Book, Systems" award from the Computer Press Association.

Roberto DiPietro is a Full Professor in Cybersecurity at HBKU-CSE. Previously, he was the Global Head of Security Research at Nokia Bell Labs and Associate Professor (with tenure) of Computer Science at the University of Padova, Italy. He also served 10+ years as a senior military technical officer. Overall, he has worked in cybersecurity for 23+ years, leading technology-oriented and research-focused teams in the private sector, government, and academia (MoD, United Nations HQ, EUROJUST, IAEA, WIPO). His main research interests include security and privacy for wired and wireless distributed systems (e.g., Blockchain technology, Cloud, IoT, Online Social Networks), virtualization security, applied cryptography, computer forensics, and data science. Besides being involved in M&A of start-ups and having founded one (exited), he has produced 230+ scientific papers and patents over the cited topics, co-authored three books, edited one, and contributed to a few others. He is an AE for ComCom, ComNet, PerCom, Journal of Computer Security, and other Intl. journals. In 2011-2012 he was awarded a Chair of Excellence from University Carlos III, Madrid. In 2020 he received the Jean-Claude Laprie Award for significantly inïluencing the Dependable Computing theory and practice.

Kim-Kwag Raymond Choo (Senior Member, IEEE) received a Ph.D. in information security from Queensland University of Technology, Brisbane, QLD, Australia, in 2006.

He holds the Cloud Technology Endowed Professorship at The University of Texas at San Antonio (UTSA), San Antonio, TX, USA. In 2015, he and his team won the Digital Forensics Research Challenge organized by Germany's University of Erlangen-Nuremberg. Dr. Choo was the recipient of the 2019 IEEE Technical Committee on Scalable Computing Award for Excellence in Scalable Computing (Middle Career Researcher), 2018 UTSA College of Business Col. Jean Piccione and Lt. Col. Philip Piccione Endowed ResearchAward for Tenured Faculty, British Computer Society's 2019Wilkes Award Runner-Up, Best Paper Awards from 2019 EURASIP JWCN, IEEE TrustCom and ESORICS 2015, Korea Information Processing Society's JIPS Survey Paper Award (Gold) 2019, IEEE Blockchain 2019 Outstanding Paper Award, Inscrypt 2019 Best Student Paper Award, Fulbright Scholarship in 2009, 2008 Australia Day Achievement Medallion, and British Computer Society's Wilkes Award in 2008. He is also a Fellow of the Australian Computer Society and Co-Chair of the IEEE Multimedia Communications Technical Committee's Digital Rights Management for Multimedia Interest Group.