One IOTA of Countless Legions: A Next-Generation Botnet Premises Design Substrated on Blockchain and Internet of Things

Haoyu Gao, Leixiao Li, Hong Lei, Ning Tian, Hao Lin, and Jianxiong Wan

***Abstract*—Although botnet had been at the top of the list of main threats to the cyber world for an extended period of time, its harmfulness has been constrained nowadays due to the development of kaleidoscopic network security enforcing tools and people's increasing awareness. And the underlying technology of the botnet has been stagnant ascribing to many drawbacks such as inadequate protection of the identity of the Botmaster and weak resilience of the botnet's infrastructure. In this article, we first introduce a new classification of the botnet based on botnets' underlying network, then briefly analyze the main flaws of the traditional botnet and some looming Blockchain-based botnets, with pros and cons of leveraging Blockchain to construct botnets. Furthermore, we propose one IOTA of countless legions (OICL), a newfangled versatile botnet infrastructure that overcomes the bottlenecks that other contemporaries cannot eliminate. It leverages Blockchain, also known as distributed ledger technology (DLT), to be its premises and uses many advantages of it without paying too many tradeoffs. Also, we invent a whole set of communication protocols for OICL and a novel scheme called Proof of Honest (PoH) to identify the espionage infiltrated into the botnet to further promote the robustness. In addition, we discover and propose a mechanism called collateral damage binding (CDB), which proves that the botnet has it such as OICL is far more robust than those who do not. Performance evaluations show that OICL is effective, more cost-saving, and fast-responding compared with the Bitcoin-based botnets as baselines.**

***Index Terms*—Blockchain, botnet, cyber security, distributed ledger technology (DLT), IoT.**

Manuscript received 9 April 2023; revised 30 August 2023; accepted 2 October 2023. Date of publication 9 October 2023; date of current version 21 February 2024. This work was supported in part by the National Key Research and Development Program of China under Grant 2021YFB2700601; in part by the Finance Science and Technology Project of Hainan Province under Grant ZDKJ2020009; in part by the National Natural Science Foundation of China under Grant 62163011; in part by the Research Startup Fund of Hainan University under Grant KYQD(ZR)-21071; in part by the Inner Mongolia Autonomous Region Special Program for Engineering Application of Scientific and Technical Payoffs under Grant 2020CG0073 and Grant 2021CG0033 in part by the Inner Mongolia Autonomous Region Key Research and Development and Achievement Transformation Project under Grant 2022YFSJ0013 and Grant 2022ZY0169; in part by the Inner Mongolia Autonomous Region Higher Education Youth Science and Technology Talent Support Program Project under Grant NJYT22084 and Grant NJYT23055; and in part by the Inner Mongolia Autonomous Region Postgraduate Science and Technology Innovation Project under Grant SZ2020068 and Grant JY20220078. *(Corresponding author: Hong Lei.)*

Haoyu Gao is with the School of Cyberspace Security (School of Cryptology), Hainan University, Haikou 570228, China, and also with the Blockchain Research Group, Oxford-Hainan Blockchain Research Institute, Chengmai 571924, China (e-mail: 1095055672@qq.com).

Leixiao Li and Jianxiong Wan are with the College of Data Science and Application, Inner Mongolia University of Technology, Hohhot 010051, China, and also with the Research Center of Large-Scale Energy Storage Technologies, Ministry of Education of the People's Republic of China, Beijing 100816, China (e-mail: llxhappy@126.com; jxwan@imut.edu.cn).

Hong Lei is with the School of Cyberspace Security (School of Cryptography), Hainan University, Haikou 570228, China, and also with the Blockchain Research Group, Oxford-Hainan Blockchain Research Institute, Chengmai 571924, China (e-mail: leiluono1@163.com).

Ning Tian is with the School of Engineering, University of Warwick, CV4 7AL Coventry, U.K., and also with the School of Computer Science and Technology, Hainan University, Haikou 570228, China (e-mail: tianning410@163.com).

Hao Lin is with the School of Computer Science and Engineering, Tianjin University of Technology, Tianjin 300384, China (e-mail: suzukaze_aoba@foxmail.com).

Digital Object Identifier 10.1109/JIOT.2023.3322716

I. Introduction

BOTNET, rendered as a relentless threat to the digital world, is like a ghost that changes in form or shape but never dissipates. Since the first Internet relay chatting (IRC)-based botnet emerged in 1993, it has become a dangerous threat that is difficult to detect and dismantle [1] from time to time. The botnet is a network of compromised machines, individually referred to as bots or zombies, and controlled remotely by a malicious entity known as the Botmaster. For decades, the most everlasting characteristic of the botnet is its large scale, i.e., the number of victims can reach up to or more than 10 000 [2], such as Srizbi [3], Shamoon [4], Kraken [5], Zeus [6] [7], Retadup [8], [9], etc. Thereafter, added with new features of specialization and fine-grained control, botnets are evolving and tailored continuously to fit the advanced persistent threat (APT) [10], a stealthy threat actor, typically a nation-state or state-sponsored group, which gains unauthorized access to a computer network and remains undetected for an extended period.

What is more, the wide use of IoT makes things worse [11], [12]. Mirai [13], [14], an IoT-focused malware, by exploiting a list of default usernames and passwords, which most users never change, had been able to infect hundreds of thousands of connected devices, from smart energy meters to home CCTV cameras and connected baby monitors (after its author identified and arrested, it soon crumbled). Although the application scenarios and the technologies for implementing botnets keep changing, the underlying command and control (C&C) structure, which is a channel of communication maintained by the Botmaster to dispatch instructions toward the compromised network [15], can be considered relatively as a fixture. According to the topology of C&C structure, botnets can be classified into three types: 1) centralized; 2) decentralized; and 3) hybrid [16].

1) *Centralized Structure:* From the perspective of topology, destroying the centralized botnets is pretty easy since

once the main C&C server gets offline, the traffic of the whole botnet will be locked.

2) *Decentralized Structure:* Compared with the centralized botnets, decentralized ones, typically those adopting P2P as a basic communication protocol, have more sturdiness against tearing down because their C&Cs are more or less distributed among all the participating nodes, in other words, the bots per se can be considered as C&C. Due to the fact that there is no central server to be the coordinator to synchronize actions, in order to join and keep pace with the swarm, a P2P bot needs to sustain a list of its neighbors to stay connected with the botnet. This list is called the neighbor list (NL). Thus, the dependence on NL is the Achilles' Heel of decentralized botnets. In addition, botnets of these kinds are nearly impotent in finding the spying nodes accurately (technically, spying nodes are sensors/honeypots deployed by security or government agencies to probe the intelligence of the botnet). There are some mitigations that aim the NL as the target for sabotaging the botnet. By injecting spies into the botnet and collecting as much intelligence of the botnet as possible, security agencies can commit multiple attacking methods leveraging the NL to undermine the decentralized botnet, such as NL poisoning/polluting [17], sink-holing, and Sybil/eclipse attacks [18], [19], [20].
3) *Hybrid Structure:* A botnet of this sort combines P2P and centralized for the purpose of obtaining the benefits of both, however, it typically suffers the weakness of both.

Apart from topology-oriented classification, we introduce another taxonomy that divides botnets into two types depending on whether they rely on the underlying network: the autarky botnet and the parasite botnet. Autarky means that the entire botnet infrastructure is completely implemented by its author, including designing the communication protocols, constructing the C&C channels, and starting C&C servers and domain names. Comparatively, the parasite botnet constructs its C&C channel directly based on existing digital services (typically public ones), such as Skype [21], social media [22], and even a legal website [23]. We argue and prove that the parasite botnet has more resilience than the autarky ones by proposing a concept, the collateral damage binding (CDB), to roughly quantify the extent of parasitism. In short, if the security agencies desire to destroy the parasitic botnet, huge collateral damages to the host will be entailed such as harming benign users and disrupting normal transactions. Intuitively, to destroy a botnet of this kind, the only available options are letting security agencies detain the Botmaster hidden behind or requiring the host to clear the botnet.

Conclusively, the essential countermeasures against botnets can be generalized to: 1) destroying the bulk of it; 2) jamming the network traffic of it; and 3) capturing the Botmaster who owns it.

So, if a botnet is able to survive the three mitigations above-mentioned and endure to exist, it can be considered to have strong resilience. Apparently, the traditional centralized (varieties of DNS/Telnet botnet) and decentralized P2P) botnet can hardly survive all three countermeasures. But the emergence of Blockchain technology [interchangeable with distributed ledger technology (DLT)] and the development of IoT technology may shed light on the retaliation for the three seemingly mighty mitigations since Blockchain offers a strong autonomous working mode to protect its participants' identity [24] and it has innate resistance against Single Point of Failure (SPoF). And if properly designed, the pervasive IoT devices can be leveraged to be botnet maintainers and contribute a lot in keeping on the go of it. Thus, a couple of cutting-edge researches attempt to harness the power of Blockchain to enhance the botnet by synthesizing the botnet with it to derive the capabilities to resist the mitigations aforementioned. However, due to the fact that doing things on most of the public blockchains can rarely be cost-effective, thereby, deploying a hyper-scale botnet with these proposals is more or less infeasible in practice.

In this article, we conduct a brief analysis of the main drawbacks of both the traditional botnet and Blockchain-based ones. And in the view of the attacker, based on our previous researches [25], [26], we find that these disadvantages can be eliminated and the resilience could be strengthened by IOCL, a novel botnet infrastructure we propose. We finally discuss probable mitigations against this new kind of botnet. The one IOTA of countless legions (OICL) we propose has advantages.

1) *Blockchain-Boosted:* It leverages a new DLT implementation, the IOTA, to be its premises deriving all the advantages that Blockchain has.
2) *Cost-Saving and Low-Latency:* The OICL overcomes the main bottlenecks, namely, the cost and latency, that other Blockchain-based botnets have.
3) *Massive-Maintainers:* With the aid of IOTA's Tangle, it recruits benign IoT devices as its maintainers.
4) *NL-Eradication:* It eliminates the dependence on NL that traditional decentralized botnet suffers.
5) *Versatility:* It can be put into use for two typical scenarios of a botnet, one-way communication for Distributed Denial of Service (DDoS) and bidirectional communication for remote control.
6) *Identification Friend or Foe (IFF):* We integrate an advanced IFF mechanism, the Proof of Honest (PoH), with the OICL to accurately identify the espionage nodes within the botnet.
7) *Strong Resilience:* The OICL binds the collateral damage with the DLT as enforcement to promote its resilience.

The remaining sections of this article are organized as follows. Section II presents a botnet chronicle to introduce researches on the traditional botnet and some Blockchain-based botnets are briefly discussed. In order to explain why we utilize IOTA's Tangle as the infrastructure of OICL, Section III outlines some preliminaries of it to demonstrate its advantages over other DLTs. Section IV presents the extensive model design of OICL, including its communication protocols, C&C premises, and a novel polygraph mechanism, the PoH, detecting espionage inside the OICL and promoting the OICL's robustness. The effectiveness and performance of the OICL are shown and evaluated through experiments in Section V. The security analysis is discussed in Section VI. Section VII sheds light on possible measures to further strengthen the OICL and mitigations against it. Finally, Section VIII concludes this article.

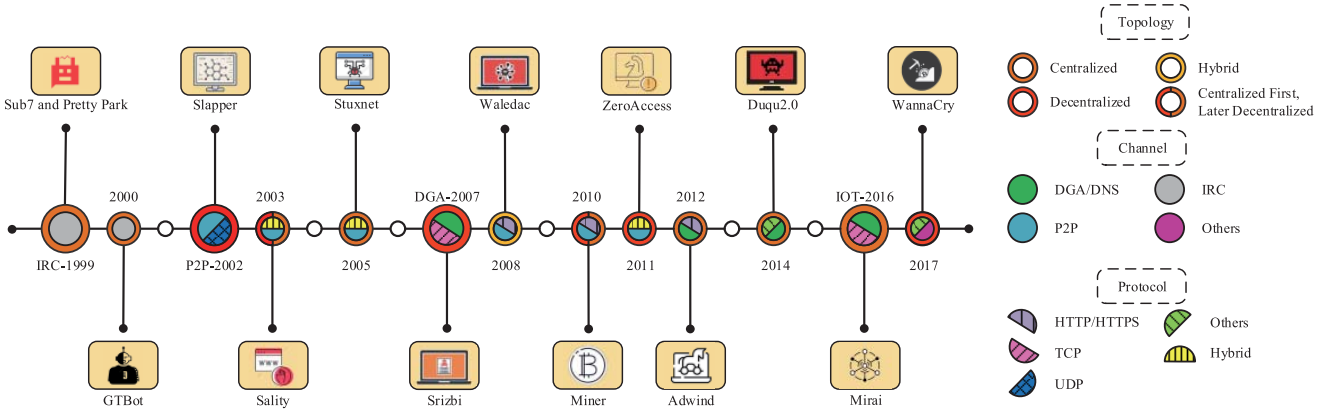

Fig. 1. Chronicle of botnet.

TABLE I
TRADITIONAL BOTNETS AND THEIR FRAILTIES

Strain of botnet	Anti-SPoF	Botmaster protection	Censorship-Resistant	Versatile	Resilience against intelligence gathering
IRC	×	×	×	×	×
DGA/DNS	×	×	×	×	×
P2P	√	×	×	×	×

II. LITERATURE REVIEW

A. Traditional Botnets—Now and Then

In retrospect, dating back to the 1990s, the pioneers of botnet using IRC, which was envisioned in 1988 and was a popular chatting tool among hackers at that time, as their C&C channel, such as Sub7 and Pretty Park [27]. Their successor, GTBot [28], came into known in the year 2000 and was the first to integrate DDoS ability into its design. In 2002, Slapper, seen as the pathfinder of P2P decentralized botnet, improved the robustness and efficiency of the P2P networking capabilities of its ancestor's, Apache Scalper [29]. And it was wormable to infect more computers. The domain generation algorithm (DGA) was leveraged by an infamous botnet, Srizbi, spread among 450 000 computers, sited in 2007, to make it difficult for law enforcement to effectively eliminate it. The emergence of a botnet that aims IoT devices as its target appeared in 2016, and its name was derived from a Japanese word, Mirai (which means future) [13], accurately grasping the trend of IoT and its weakness in security. Later, as a wormable ransomware, WannaCry was spread across the world to extort bitcoin from the victims by ciphering their files. Other records of botnets and their protocol as well as their topologies and channels are depicted in Fig. 1 as a chronicle.

All the entries in the chronicle are tagged as traditional botnets, since "traditional botnet" means that their frailties are obvious such that the identity of Botmaster is under little protection, and in addition, their resilience is not strong enough to resist destruction. What is more, they are not qualified enough for all the hacking scenarios that require multiple functions, such as confidential data stealing, file uploading, spam, screenshot, video and audio recording, click fraud, advertising, DDoS, remote control and access, and command execution. According to their topologies, they can be classified into two sorts: 1) centralized and 2) decentralized. Almost all the DGA/DNS botnets are centralized and they are subject to SPoF. Conversely, although P2P botnets can tolerate SPoF at a certain degree since they are decentralized, the NL still made it Achilles' Heel since once the NL is poisoned, the nodes will be at risk of losing contact with the Botmaster. The frailties of the traditional botnet are concluded in Table I.

B. Blockchain-Based Botnets—The Pros and Cons

Although there are some emerging researches [30], [31], [32], [33], [34], [35], [36], [37] on leveraging DLT to construct certain components of the botnet, they are all subject to some obvious flaws such like great cost, long latency in communication, and decreased performance on large file transmission [25] and they are not full-function botnet framework in that mutual communication channels are not all implemented [38]. However, these researchers do demonstrate the advantage of utilizing DLT as the infrastructure of the botnet in that DLT provides strong anonymity, anti-audit, and stealthiness, making tracing the Botmaster behind these DLT-based botnets almost intractable.

III. PRELIMINARY

We leverage IOTA's Tangle[1] to build OICL's infrastructure. Instead of a chain structure, IOTA's Tangle uses a directed acyclic graph (DAG) as the structure of its ledger. Compared with other public DLTs, it has some peculiarities [40], [41] listed below.

1) *Highly Scalable:* IOTA's Tangle uses a DAG data structure allowing transactions to be added in parallel, unlike those chain-structured DLTs.
2) *Low Computational Resource Requirements:* Designed for IoT devices, such as sensors, to participate in a low-energy network.

[1]After some updatings and patchings [39], IOTA has become more robust than its earlier versions.

TABLE II
COMPARISON OF IOTA'S TANGLE AND OTHER DLTS IN THE USE CASE OF BOTNET

DLT-based botnet	Zero transaction fee	Parallelizable	Permanent data Storage	Anti-forensic	Cost in deploying bots	Low latency channel	Degree of decentralization	Computation resource consume
Bitcoin-based	No	No	Yes	Good	High	Lacking	High	High
Ethereum-based	No	No	Yes	Good	High	Lacking	High	High
EOS-based	Yes	No	Yes	Good	High	Lacking	Relatively Low	Low
IOTA-based	Yes	Yes	No	Better	Very Low	Have	High	Low

3) *Zero-Fee Transactions:* Unlike other public DLTs, IOTA's Tangle has no transaction fees, no matter how large the transaction is.
4) *Fast Transaction Confirmation:* IOTA transactions are confirmed within seconds.

These advantages make IOTA's Tangle an outstanding groundwork for designing and instantiating our proposal.

Furthermore, IOTA's Tangle has another unique trait that all the other DLTs do not have, the Snapshot. A Snapshot is done to prevent the Tangle from expanding too much in size. Snapshotting saves all the balances while removing the history and data, including messages, of all the transactions. The address with balances will act like a new address, and no previous history or data will be attached. Compared with other DLTs, the difficulty of digital crime forensics in IOTA's Tangle is greater than it is in others since the botnet's traffic will not be stored permanently on the full node while unlike other public DLTs such as Bitcoin and Ethereum, IOTA's Tangle doesn't keep transaction data permanently. Thereby, it has a stronger ability for anti-forensic than other public DLTs do when applied in a botnet since the botnet's traffic will not be stored permanently on it. The features that are compared for the scenario of applying botnet of the DLTs are rendered in Table II. Although there is no evidence that a botnet using EOS as its infrastructure was put into use neither in research nor wild at the time of writing this article, we still compare it with others for the sake of possibility. Due to the fact that the participants who have sovereignty to mine new blocks in EOS Blockchain are monopolized by 21 super nodes, the resistance of SPoF in EOS is not as firm as in its contemporaries.

IV. SYSTEM DESIGN

A. Forgery of the Cornerstones of the OICL's Communication Protocol

The communication protocol for the construction of the C&C of the OICL is mainly composed of three channels with different privileges designed to preserve the security and privacy of OICL's traffic. They are the bootstrap channel (BC), the upstream channel (UC), and the specialty commands dispatch channel (SCDC). The basic communicating unit of the three channels is a transaction (equivalent to a TCP packet in TCP). The UC and BC are implemented by masked authenticated messaging (MAM), whose fundamental constitution is transactions as well, but customized ones. The SCDC is built on raw transactions. By leveraging IOTA's full nodes as communication brokers, the OICL's traffics are relayed, making them difficult to trace. Thereby, when a transaction with commands dispatched by a Botmaster is received by a bot, it has been relayed and propagated by the whole DLT's network.

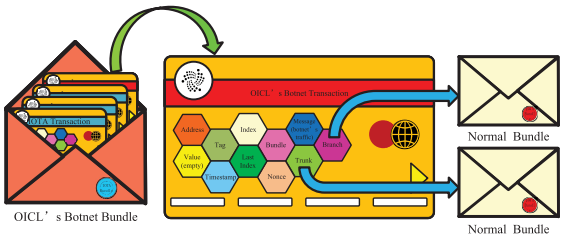

Fig. 2. Relationship between an OICL's botnet transaction and other normal transactions in a Bundle.

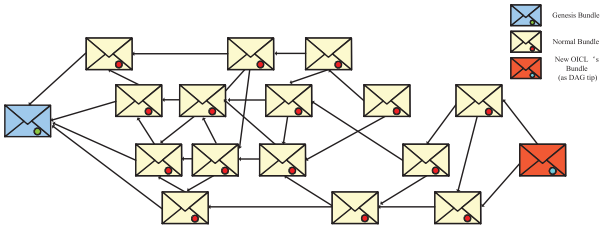

Fig. 3. OICL's Bundle mixed with normal Bundles as a new tip in the DLT's DAG structure.

1) Transaction: Initially, some relevant elements of the transaction need to be tweaked for the OICL's protocol. In OICL, the elements of a transaction we use are: the *Addresses* of transaction's sender and recipient, *Value* to send, *Tag* and *BundleHash* of the transaction, the *Index* of the transaction in the Bundle,[2] *Nonce* used for DDoS protection, *Message* used for sending user-defined data, and the *Trunk* and *Branch* parameters referencing to those transactions in another two bundles in the Tangle [42]. Among these elements, the *Tag* and *Message* parameters are requisitioned for OICL's communication protocol. Fig. 2 depicts the components of an OICL's transaction and its relationship with other normal transactions in a *Bundle* is depicted in Fig. 2

As Fig. 2 shows, our OICL's transaction is packed with other normal transactions altogether into a Bundle. The Bundle that carries OICL's traffic is sent to the full nodes of the DLT and blended with other normal Bundles, as shown in Fig. 3.

As Fig. 3 shows, the orange Bundle that has OICL's transactions is sent and joined with other normal Bundles in IOTA's Tangle, forming a DAG structure.

2) Masked Authenticated Massaging: Among these elements, Message is used to implement the MAM, which is the infrastructure of the BC and UC channels. In OICL,

[2]A Bundle in IOTA is equivalent to a block in other public DLTs.

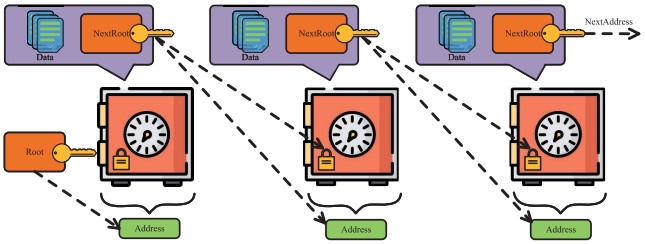

Fig. 4. Chain structure and components of the public MAM.

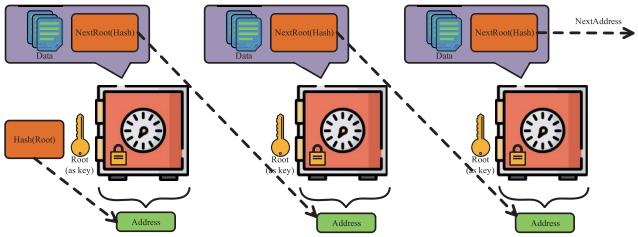

Fig. 5. Chain structure and components of the private MAM.

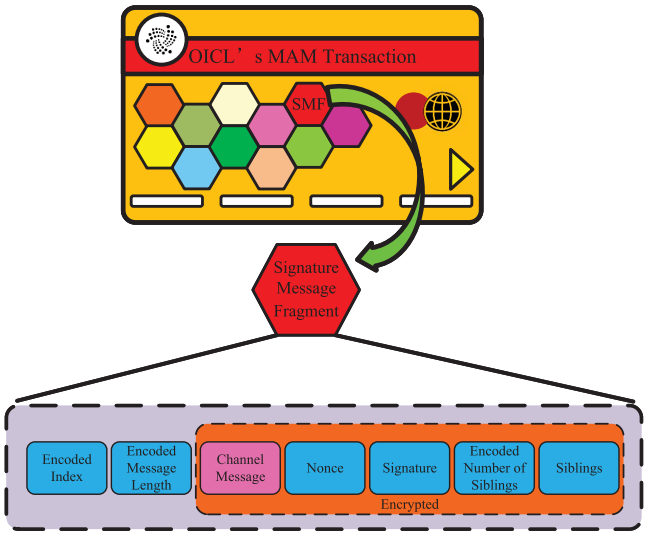

Fig. 6. OICL's MAM payload and its main components.

there are two roles for MAM. One is the owner, as the data publisher, and the viewer, who subscribes to the owner via MAM. Thereby, MAM works like a one-way tunnel where the owner puts data into it on one side and subscribers receive them on the other side of it. This ownership is implemented and secured by the Seed [43], which can be, for simplicity, considered as private keys. Only the Seed's owner can publish data to the channel.

In OICL, the MAM has two privileges.

1) *Public:* Everyone who has the channel root which is the same as the first address to the message chain attached to can view the content within, and the masked message is decrypted using root. The chain structure and details of public MAM are shown in Fig. 4.
2) *Private:* Only the Seed owner can access, and the masked message is decrypted using root. MAM's address on Tangle is the hash of the root, thus making it impossible to decrypt the masked messages since there is no feasible way to derive the root from its hash. Thus, the secrecy of messages is guaranteed provided that the Seed is held confidential. The chain structure and details of private MAM are shown in Fig. 5.

The parameter, root (or the hash of the root), which works as a message identifier, is given to viewers so as to find messages from Tangle. Although MAM works like a tunnel or chatting room, its basics are still transactions. As a common sense on DLT, once an address sends a transaction, it becomes public and everyone is able to query its history of activity on the chain. As a result, if the adversary keeps an eye on a certain address, he can obtain those patterns that compromise the stealth of the botnet. For example, if an entity in our botnet wants to post data every 15 min, without MAM, it has to post every message to the same address. Because any distributed ledger including the Tangle is publicly accessible, it is easy for adversaries to identify such an address that updates every 15 min. Fortunately, MAM posts messages to different addresses with verbose information connecting them, forming a message chain. As seen from Figs. 4 and 5, on the message chain, from one generation to the next generation, the older message always leads to newer ones, namely, the predecessor and successor in the MAM. The Message parameter of a transaction is redefined as signature message fragment (SMF) and used as a MAM's payload, whose constitutions are depicted in Fig. 6. Seen from Figs. 4 and 5, the message and other parts are ciphered by Root or Sidekey, depending on what privilege the MAM owner adopts.

As Fig. 6 shows, after all these widgets of SMF are set, the transaction that has the MAM's payload will be treated as a normal transaction and propagated to the DLT's network.

B. Network Topology and the Communication Protocol of the OICL

The most important component of a botnet is its C&C. Our proposed OICL's communication C&C combines the advantages of existing DNS botnets and P2P botnets and eliminates their drawbacks. Contrary to the ordinary DNS botnets, in our system bots do not need to contact any controlling server. In addition, unlike the P2P botnet that needs to maintain an NL, in our architecture, participants connecting to full nodes (typically IoT gateways), are independent of each other such that one is unaware of the existence of its neighbors.

1) Overview: The overview of the topology of our proposal is depicted in Fig. 7. Bots in our system communicate indirectly with Botmaster via either Hornet [44] node or Bee [45] node (which are called the full nodes) that usually runs on IoT gateways that communicate via the Gossip protocol. Also, as shown in Fig. 7 there are many IoT devices and other applications like wallets connecting to the full nodes via the Tangle's communication API.

The set of dedicated channels shown in Table III is designed for robust and fully functional communications between bots and Botmaster with different privileges. The public BC can be only used by Botmaster to broadcast launching instructions (LIs) to each node, including those belonging to the attacker, with full node information and basic commands. After receiving the LI, a connection is established between the bot and

TABLE III
THREE CHANNELS OF THE OICL AND THEIR TRAITS

Channel name	Privilege	Implemented on	Encrypted by	Spam Protection (SP)	Collateral Damage Binding (CDB)
The Bootstrap Channel (BC)	Public	MAM	-	√	√
The Upstream Channel (UC)	Private	MAM	Bot's seed	-	√
The Specialty Commands Dispatch Channel (SCDC)	Private	Raw Transaction	Bot's seed	-	√

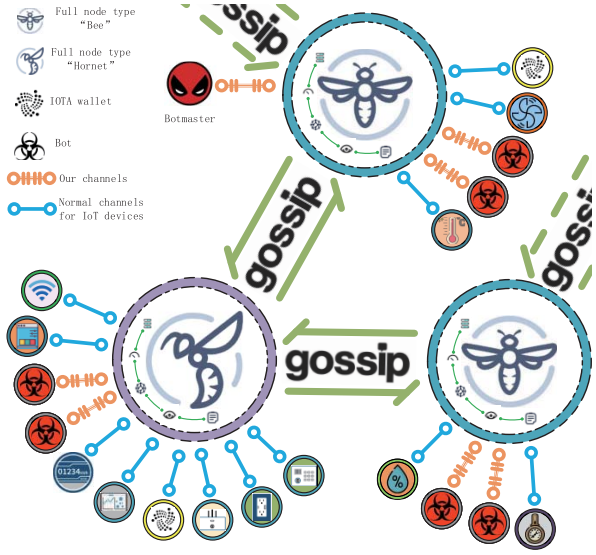

Fig. 7. Topology of OICL.

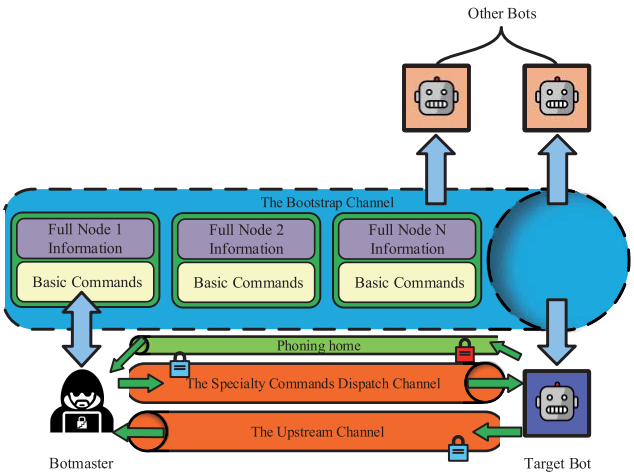

Fig. 8. Three channels of OICL's C&C.

a selected full node. Thereafter, the bot and Botmaster can exchange data through the private UC and SCDC. All channels are under the protection of CDB. In contrast, spam protection (SP) is only applied to BC since UC and SCDC are not visible to the adversary, thus making them unable to be DDoSed. Logically, all channels in Fig. 8 connect Botmaster with bots directly, whereas in practice these channels constitute many intermediate full nodes helping relay traffic.

Fig. 9 depicts the OICL communication protocol composed of four stages. Note that the stage before the Bootstrap at the top of Fig. 9 initiates at the beginning of the whole protocol,

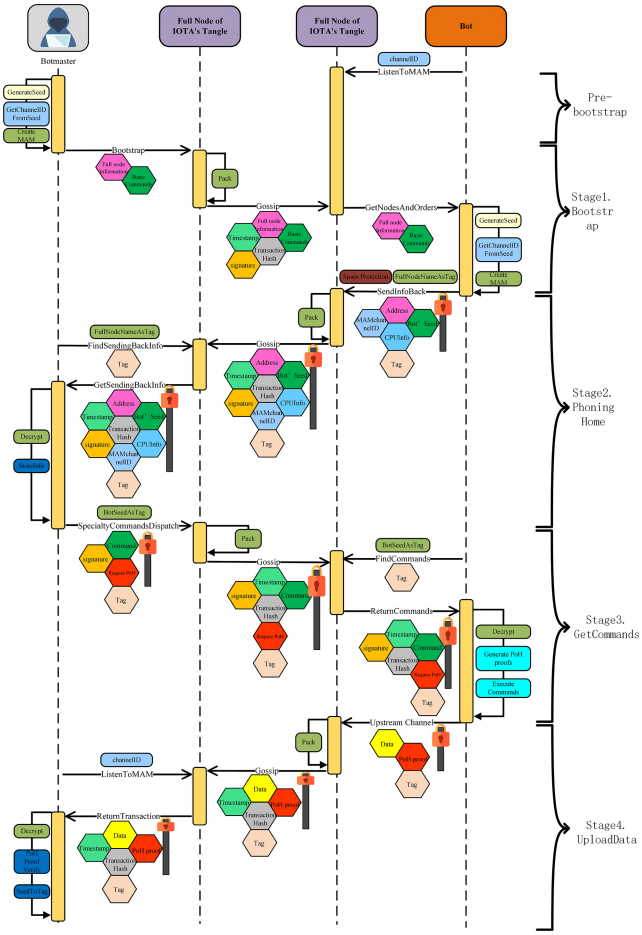

Fig. 9. OICL communication protocol.

indicating that the id of BC is derived in this stage. The BC in this stage will be empty until stage 1 brings LI in. First, in the bootstrap stage, Botmaster initiates a channel identified by a channel ID (hardcoded in the binary execution file in advance for infecting a computer and then turning it into a bot), namely, the BC, with LI putting in. LI has the full node information and basic commands. The full node information consists of the IP address and port number of the full nodes selected by Algorithm 1, And basic commands consist of requiring a description of the bot's CPU details, and those actions that do not need feedback such as DDoS. LI is sent to a full node to be packed in a transaction and propagated to other full nodes. After receiving the LI, the full node Botmaster connected to pack the LI into a valid MAM transaction, which was attached with a timestamp and a transaction hash, and broadcast it to all other full nodes on the Internet through Gossip protocol. This is the process of constructing the BC, the bootstrap stage.

Then, in the second stage, bots create and record MAM channels by IDs for uploading data back. The MAM channel is used for UC in stage 4, with the channel ID as its identifier. (The MAM was created by the bots, not Botmaster. Do not confound it with BC's ID in stage 1.) Once the bots that listen to the BC channel have obtained the LI, they will send their CPU information, public address, seed, and UC channel ID back to the Botmaster, in the form of cyphered transactions with the full node it connecting to as the transaction's tag. The encryption key used to cipher the feedback is Botmaster's public key, which is hardcoded in the binary execution, guaranteeing that bots cannot decipher each other's data. This stage is called "phoning home."

In the third stage, Botmaster will search the phoning home information that bots have sent in stage 2 with the full nodes' name as a tag, decrypt them with his private key, and then store the UC's information locally. Thereafter, he uses the hash of the bot's seed as a transaction tag to dispatch commands and verifiable delay function (VDF) rounds (which is PoH related, the next section introduces its detail) required to execute on the bot, and listens to the UC to receive data and execution results of the commands as well as VDF proofs. These commands are particular to the bots that need to transmit data back, such as those sending screenshots or files back, listing file directories, VDF proofs, and so on. As an optional optimization, Botmaster can distribute commands to the target bot by sending transactions with data ciphered by the bot's seed.

In the final stage, bots put their data and execution results of commands derived from the Botmaster in the UC at the last stage with the VDF proof. Then, the Botmaster listens on the other side of the UC can obtain data and use the VDF proof to check whether the bot is honest or not.

2) BC Generating Algorithm: The BC generating algorithm listed in Algorithm 1 constructs a BC through which the set of full nodes can be connected by bots.

This algorithm applies to the first step for a new recruit bot to connect to the botnet. The BC is incorruptible and immune to spam attacks provided that the ownership of the MAM's seed is under the control of Botmaster. The BC is also sheltered by CDB as shown in Table III. In other words, it is highly trustworthy for disseminating LIs issued by Botmaster.

As Algorithm 1 shows, in fact, a full node enumerator begins with a node's neighboring nodes (this node is hardcoded in the binary executions), puts them into the channel, then enumerates their neighbors and their neighbors' neighbors, and repeats this process to find as many full nodes as it can. Although CDB protects the full nodes from being shut down or handed over to the adversary, we still believe that the increasing storage of full nodes' information can make the OICL strong if in the worst case that some full nodes get offline somehow. Another reason why Algorithm 1 enumerates as many full nodes as possible is that the more full nodes the OICL gets, the more CDB it binds to. The bots of OICL can connect to these full nodes with no hindrance. One of the simple ways to destroy the botnet is to take down these full nodes that Algorithm 1 obtains and the more full nodes get offline, the more damage the DLT sustains.

Algorithm 1 Constructing of the BC

Input:

portal: The bootstrap server of IOTA's Tangle;
publickKey: The public key of Botmaster;
channelID: Used for publishing full nodes IP addresses; This is a MAM channel created by Botmaster in advance (Before Bootstrap). The channelID is hardcoded in the binary execution;
Node: An instance of a full node in Tangle;
interval: The original time duration for proof of honest. Technically, it is VDF rounds.
command: The initial commands that Botmaster dispatched to bots, including those commands requiring no feedback such as executing DDoS task or mail spam;

Output:

fullNodes[]: The list of neighboring full nodes;
Initialize *portal* to a IOTA's Full node;
iota = connect(*portal*)// Connect to the *portal* then obtain a connection instance *iota*
Use the instance Iota to get the full node's neighboring full node *sneighbor Address* = *Iota*.getNeighbors()
while TRUE **do**
  // Connect to the portal then obtain a connection instance iota
  for I in neighbor Address[] **do**
    connect to I to obtain a connection instances *node* ← connect(i)
    Broadcast full node's information, DDoS/basic commands, and VDF time interval into the channel *Iota.publishToChannel(channelID,node, interval, command)*
    Add this full node to output *fullNodes[]*
  end for
  Empty the *neighbor Address* set.
  Find neighbor's neighboring nodes
  for *j* in *fullNodes* **do**
    Add *j's neighbors into neighbor Address*
  end for
  Remove all the duplicated elements;
end while

3) UC Algorithm and Bot's Phoning Home: After bootstrapping, a bot will connect to one of the full nodes derived in BC, and those bots applied for DDoS or mail spam are ready to begin their sortie. So, the BC alone is enough for these bots. However, those who need to upload data must find a way to transfer it back to Botmaster, and they can do this by sending a transaction that contains the seed as its identifier as well as the ChannelID to use the channel. Then all the stuff is encrypted by the public key of the Botmaster, making these transactions mutually confidential among bots. So, if a bot falls into the adversary's hands, the data sent by other bots in transactions will be unreadable to the adversary.

Through searching specific transactions with full nodes' names as tags (IOTA's Tangle has provided a set of APIs to filter and find transactions with parameters of a transaction as searching keys, such as addresses and tags), the Botmaster can

Algorithm 2 Phoning Home and UC Construction

Input:

publickKey: The public key of Botmaster; it is hardcoded in the binary execution;
channelKey: The encryption key of the channel that makes the channel private and encrypted. It can be derived from the seed's hash;
Privilege: The privilege of the Channel will be private here;
seed: The seed of the bot, used to generate MAM Channel and Address;
Address: The address of the bot on Tangle, generated by
seed:;
CPUinfo: The processor information of the bot;

Output:

channelID: The ID of the private channel on Tangle used byBotmaster to receive data uploaded by a bot;
content: The content of transactions constituted by seed, Address,channelID, CPUinfo, and publicKey;
channelKey: used as encryption key for private channel;
tag: the tag of transactions sent by the bot to inform the Botmaster about its existence. It can be used by Botmaster as a filter to search specific transactions. Therefore, it must be known to Botmaster in advance like the names of the full nodes put into the BC;
Initialize:
the *tag* tag as the name of the full node it connecting to
Obtain the full node's name that the bot is connecting to *FullNodeName:= GetNodeName()* ;
Use the name as the tag of the transaction *tag* := *FullNodeName*
Set the privilege of the channel to private *Privilege*:= private
Get the hash of the bot's seed, then use it to encrypt the channel in the next step *channelKey*:= Hash(*seed*)
Create the encrypted channel, then get the id as the identifier of the channel*channelID*:= MAMCreate (*seed,privilege,channelKey*)
Encrypt the bot's seed, address, channelID and CPUinfo as the content for the phoning home transaction with Botmaster's public key hardcoded in binary execution *content* := encrypt(*seed,Address,channelID,CPUinfo,publicKey*)
Pack the content into a transaction then send it to a full node with which the bot is interacting. And the address that the transaction was sent to matters not since Botmaster only needs the tag to find the transaction.*transaction* := MakeTransaction(*tag,content*)
SendTransaction(*transaction*)

obtain the reverse-connecting bot's seed and address readily and decrypt them as well as the CPUInfo (in cipher) that they sent back with his private key.

In addition, when a bot needs to upload big files or huge amounts of data, the MAM that the UC leveraged is not qualified for the task as MAM is designed for message queuing telemetry transport (MQTT), not for big file transmission. Thus, we introduce the decentralized storage solution, such as Swarm [46] or IPFS [47] as an enhancement for big file uploading. Designed to be versatile, the OICL integrates the Swarm as a big file-uploading enhancement since Swarm has a stronger anti-censorship stance than IPFS does. It incentivizes content-agnostic collective storage (block propagation/distribution scheme), implementing plausible deniability with implausible accountability through a combination of obfuscation and double masking.

On Swarm, each file uploaded will generate a hash as its identifier. A bot merely needs to send the Swarm hash to Botmaster via UC instead of directly putting the big file into it. After the enhancement, compared with other DLT-based botnet implementations like [30] and [31], OICL has a greater ability to accelerate the speed of big file uploading.

4) Specialty Commands Dispatch Channel: After the upstream conduits are accomplished, Bots that have digital assets raiding tasks are ready to receive peculiar orders assigned by Botmaster, such as confidential theft, screenshots, audio/video recording, file uploading, ransom, etc. Given the seeds and addresses of the bots and details of the complete nodes they are connected to, Botmaster is able to assign commands via transactions tagged with this information. Callbacks can be registered by bots to subscribe to these tagged transactions by bots to follow Botmaster's further instructions. Due to the fact that the seeds are ciphered by the public key of Botmaster in stage 2, the adversary is not authorized to decrypt these ciphers, thus securing the privacy of the bot's data. Consequently, these tagged transactions can be considered invisible to the adversary. The SCDC is quarantined from corruption by the adversary. The flow of achieving this channel is illustrated by Algorithm 3.

As Algorithm 3 demonstrated, the commands within the dispatching transaction are encrypted with the bot's seed as the symmetric encryption key, since the seed is only accessible to the bot itself and Botmaster, the content in SCDC will remain confidential. Besides, in order to receive data uploaded by bots, Botmaster listens to the UC in Algorithm 2 marked with channelID and channel Key derived in stage 2. Bots can simply use their seed's hashes as tags to find SCDC transactions and decrypt them with their seed as key while protecting it from compromising secrecy.

C. Proof of Honest

PoH is a novel scheme we propose to check whether a bot is controlled by the adversary and promote the robustness of OICL. Its processes are depicted in Fig. 10. In order to hijack a device to become a bot, the Botmaster will first need to deliver the bot's payloads to the target. Depending on whether the target device is controlled by the adversary, after becoming a bot, the target can be defined as follows.

1) *Honeypots and Sensors:* The targets may be controlled by the adversary who waits to analyze the botnet's traffic by letting them be hijacked on purpose. These trap-like targets are called honeypots or sensors. Sensors are spy nodes that collect all available intelligence,

Algorithm 3 Constructing of the SCDC

Input:

Privatekey: owned by Botmaster used to decrypt data sent back by bots;
*commands:*a set of orders Botmaster needs to sendtobots;
vdfrounds: The round parameter of VDF's eval function, pre-selected by Botmaster;
FullNodeNames: The name of full node obtained in algorithm 1, used as searching key to obtain bot's phoning home transactions;
*Address:*bot's address on Tangle, has been sent back in stage 2;
tag: search key of SCDC transaction used by bot to find Botmaster's commands. And it can be derived by the hash of the seed of the bot;
data: data of the transactions sent by the targeted bot in phoning home, including CPU information, bot's seed and address, as well as id of UC;

Output:

seed: bot's seed sent back in its phoning home transaction;
address: bot's address sent back in its phoning home transaction;
channelKey: The encryption key of the UC that makes the channel private. It is obtained by Botmaster in the phoning home stage. The bot can use its seed as the UC's encryption key;
channelID: The ID of UCof the bot used for sending data back to Botmaster;
Initialize:
tag_n for n $\in \{1, 2, \ldots, n\}$ in *FullNodeNames*
The phoning home transaction was tagged with the full node name of the bot. Find it with *withFullNodeNameas* search key *transaction* := findTransaction(*tagn*)
Get contentfrom the transaction then decrypt it with Botmaster's privte key *data* := getAndDecryptContentOfTransaction(*transaction,privatekey*);
Obtain the seed of the bot from data and hash it to get the UC encryption key *seed* := getSeed(*data*), *channel*Key := Hash(*seed*)
Find the UC's id in data *channelID* := getChannelID(*data*)
Listen to the UC channel to wait for the bot to send data (VDF proof and time consumed, files, results of command executions, and others) back MAMListen(*channelID,channelKey*)
Get the bot's address on Tangle and its CPU data *Address* := getAddress(*address*),*CPUInfo* := getCPUInfo(*data*)
Encrypt and pack the commands and VDF rounds with the bot's seed into a transaction, and send it to the targeted bot. And the address that the transaction was sent to matters not since the bot only needs the tag to find the transaction.

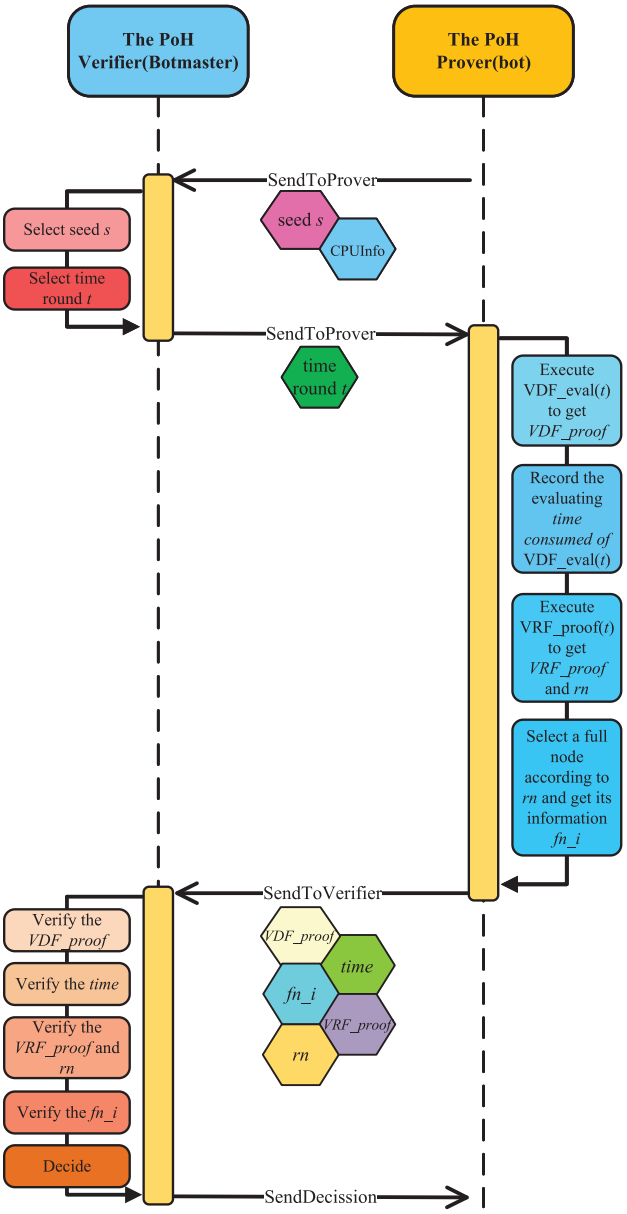

Fig. 10. Working flow of PoH.

based on which the botnet can be subverted via various approaches such as executing sinkholing attacks, and NL poisoning [48]. They infiltrate and monitor the botnet by camouflaging themselves as benign bots except for some hostile characteristics, such as a) sensors do not execute any Botmaster's commands; b) most sensors block all outbound botnet traffic; and c) some smart sensors send falsified or trash data back to Botmaster for protecting themselves from exposure. Based on the fact that the adversaries such as law enforcement officials are bound by legal, ethical, and technical constraints such that sensors dispatched should not participate in any action nor send any data as digital trophies back to the Botmaster, thus a)–c) can be conducted.

2) *Nexuses:* In view of network topology, there is a situation in which a large number of bots connect to the botnet via a single full node of the DLT. In this case, the full node with a great many bots connecting to is called the nexus node for OICL. Although the collateral damage to the DLT will be great if the adversary decides to shut down these nexuses, all the bots connecting to it

will lose contact with the botnet. As a countermeasure, we propose a novel scheme called PoH. PoH can: a) discern whether a bot is in a sensor or honeypot deployed by the adversary for intelligence probing and b) disperse the bots across the full nodes obtained by Algorithm 1, lest they all converge to a single full node of the DLT. In brief, PoH is a polygraph mechanism that separates sensors/honeypots from normal bots thus promoting the robustness of the botnet.

1) Honeypot and Sensor Detecting: It is a challenge for Botmaster to identify sensors and honeypots. Previous approach [48] leveraged the difference of the clustering coefficient (CC) for the NL between bots and sensors/honeypots. However, in OICL, due to the elimination of the NL, all the detecting plans depending on the NL fail. Conclusively, we argue and believe that rather than designing a complicated detecting scheme based on graph theory or a reputation system to assess the loyalty of a bot, grasping the very essential discrepancy between bots and sensors/honeypots is the key to distinguishing them. The discrepancy is that the sensors and honeypots do not participate in botnet activity or send truthful and credible information back to Botmaster. Thereby, the difficulty now shifts to how to identify if the data sent back is treacherous. We propose a honeypot detecting scheme to judge whether the basic information (CPUInfo) sent back is true or not. Based on the fact that different CPUs have different computation powers, we leverage the verifiable computing issue in the VDF [49], [50] to confirm the validity of the CPUInfo sent by a bot.

The VDF is an important tool used for adding delay in decentralized applications. A VDF satisfies three properties, which are prescribed evaluation time, sequentiality, and uniqueness [51] such that a credible timer can use VDF as its cornerstone. Basically, it is a function $f : t \rightarrow proof$ that takes a prescribed time to compute, even on a parallel computer. However, once computed, the output can be quickly verified by anyone. Moreover, every input x must have a unique valid output y. Among two existing VDF algorithms [52], [53] and their implementation [54], we evaluated both to figure out which one is more appropriate for our scheme. Through comparing the two implementations, in a range of 0 to 500 000 of the time round t, we found that the larger t is, the bigger the size of the proof will be in [53] (linearly increased), whereas in [52], the size of the VDF proof is a constant value.

This indicates that [52] is more space and bandwidth-saving, thus making it our adoption. In OICL, the simplified flow of the polygraph is as follows.

1) VDF_eval(t)$\rightarrow$ (proof) takes time rounds t (designated by the Botmaster) and outputs a proof.
2) VDF_verify(proof,t) $\rightarrow$ {True,False} outputs True if proof is the correct evaluation of t.

The prover, namely, the bot proving its loyalty to the Botmaster, is required to execute VDF_eval(t), and then provide its output y as time proof for the verifier, the Botmaster, to check out whether it is valid or not (parameter t is consigned by Botmaster, and can be adjusted). Another vital strait of VDF is that the time entailed to solve the puzzle varies in different CPU types since different CPUs have different computing power. Consequently, Botmaster can demand the details of a bot's processor when its upload channel is set up. Then via comparing the elapsed time the bot provided with that it should have consumed for its alleged specific CPU type, Botmaster can adjudicate if a bot is a perfidious node or not.

By leveraging the verifiable computing issue of VDF, based on the fact that the time consumed in executing VDF_eval() on different CPUs varies, through experiments, a dictionary for polygraph is provided.

After receiving the elapsed time spent for executing the VDF_eval for specific rounds from a bot, By querying the polygraph dictionary Table IV, the Botmaster is able to find out whether the CPUInfo in Algorithm 2 is true or not. For example, assuming that a bot, as a prover, claims that its CPU type is "Intel Core i7-9700" in the phoning home stage. Then, the Botmaster designates t as the VDF_eval rounds, sends it to the bot, and waits for the response, which is composed of a VDF proof for VDF_verify and a time parameter that the bot has spent in executing the VDF_eval. Using the CPU type and VDF round parameter t to query the polygraph dictionary, then by comparing the time sent back with the predetermined time in the dictionary, the Botmaster can tell if the bot lies to him. In addition, the Botmaster can use t as a timer for heartbeat detection. In brief, the Honeypot and Sensor Detecting scheme interrogates bots on three trials.

1) Send heartbeat message back to the Botmaster or considered to be an adversary's sensor.
2) Send valid VDF proofs to Botmaster or convicted of espionage.
3) Prove that their CPU information sent back is not forged according to the time of rounds consumed by running VDF or condemned to be a spy.

Conclusively, either the proof or its CPU information was phony or the bot simply provisions nothing, then it would be regarded as a treacherous node serving the adversary with high possibility. A bot may try to send messages back but keep losing its data packets in poor network conditions when it is interacting with Botmaster, performing very much like a sensor. In this scenario, giving it specific commands is unnecessary as it merely is a waste of Botmaster's network resources. Thus, these kinds of nodes can be seen as sensors in the view of Botmaster. Then abrogation of listening to their uploading channel and allotting them no further orders are reasonable measures.

2) Bots Dispersing: An excess of bots crowded on a single full node undermines the robustness of the botnet since the adversary can sever them off the botnet by simply snuffing out the full nodes. In this situation, Although switching to another full node is always allowed, the perturbation of bots' losing contact is unavoidable. Thus, we provide a scheme based on verifiable random function (VRF) [55] to make bots evenly distributed to all the full nodes enumerated by Algorithm 1. In OICL, the simplified flow of the scheme is as follows.

1) VRF_proof(sk_{bot},t)$\rightarrow$ (rn,p_{vrf}) takes an input t (designated by the Botmaster) and bot's private key, and outputs a random number rn and proof p_{vrf}.
2) VRF_verify(rn,p_{vrf}) $\rightarrow$ {true,false} outputs true if rn and p_{vrf} is the correct evaluation of the VRF on input t.

TABLE IV
POLYGRAPH DICTIONARY

CPU Type	Time consumed in seconds for VDF_eval executing t Rounds						Average time increased for every 1000 rounds
	200000 rounds	220000 rounds	240000 rounds	260000 rounds	280000 rounds	300000 rounds	
Intel(R) Core(TM) i7-6700	18.22+-0.12(s)	19.93+-0.14	21.87+-0.12	25.42+-0.1	26.72+-0.13	28.75+-0.15	0.96
Intel(R) Core(TM) i7-6700K	17.10+-0.13	18.91+-0.11	20.65+-0.16	22.02+-0.12	23.79+-0.19	25.34+-0.18	0.85
Intel(R) Core(TM) i7-9700	13.12+-0.18	14.36+-0.17	15.59+-0.12	16.90+-0.13	18.08+-0.15	19.33+-0.13	0.65
Intel(R) Xeon(R) Platinum 8163	24.36+-0.14	27.53+-0.15	29.88+-0.18	32.33+-0.13	34.91+-0.15	37.16+-0.14	1.21
Intel(R) Xeon(R) Platinum 8269CY	19.64+-0.16	21.52+-0.12	23.38+-0.16	25.17+-0.1	27.03+-0.13	28.83+-0.12	0.92

The prover, namely, the bot, is required to execute VRF_-proof, and then use its output rn and p_{vrf} to randomly pick a full node i obtained by Algorithm 1 and connect to it to require the basic information fn_i of the full node. To be specific, fn_i is a tuple of multiple elements that provide the current status of the full node, such as its clock time, the number of transactions it processed, the number of its neighbors, its version, and its IP address. Once the bot has connected to the full node and fn_i is obtained, it sends them to the Botmaster. As a verifier, the Botmaster checks out whether they are valid or not by:

1) use VRF_verify to check whether p_{vrf} and rn are valid;
2) connect to the full node i that the bot selected and obtain its information fn_i';
3) check out if fn_i is equal to fn_i'.

After integrated with Honeypot & Sensor Detecting and Bots Dispersing, PoH operates as follows:

The whole progress of PoH spans stage 2 to stage 4 of the OICL protocol. First, the bot that serves as the prover sends its seed as a unique identification to Botmaster. Then after receiving the seed, the Botmaster chooses the time-round parameter t, and sends it to the bot. The bot uses it to execute 1) the VDF evaluation to obtain the VDF proof p_{vdf}, and records the time consumed t_c in evaluating and 2) the VRF evaluation to generate a random number rn and a VRF proof p_{vrf} that proves the number obtained is really randomly generated. Then, the bot uses rn to randomly choose a full node i enumerated by Algorithm 1 to connect to and obtain the full node's information fn_i. When p_{vdf}, t_c, p_{vrf}, rn, fn_i is successfully obtained, the bot sends them back to the Botmaster to prove its loyalty. The Botmaster then launches the PoH check, taking these arguments to scrutinize the bot's faithfulness. Thereby, the PoH can be utilized to implement a heartbeat test that requires bots to send responses back at intervals to prove their liveness and loyalty to Botmaster. There are four criteria to condemn the bot to commit espionage.

1) p_{vdf} or p_{vrf} is not validating.
2) The bot sends nothing back.
3) The evaluating time parameter t_c does not match the time of the bot's CPU type.
4) fn_i does not match the Botmaster's fn_i'.

In the progress of uploading data via UC in stage 4 of Fig. 9, bots are required to send PoH proofs and time as heartbeat packets at regular intervals, which can be adjusted dynamically on Botmaster's will. The purposes of this time-based heartbeat testing component are as follows.

1) To examine the bots if they do as the Botmaster demands faithfully.
2) To adjudicate those bots that misstate their computing power as espionage.
3) To give Botmaster some leeway to save bandwidth and computing resources when many bots attempt to send data simultaneously back and further elevate the defense against potential spams.

Algorithm 4 PoH Proofs Generation

Input:

channelKey: The encryption key of the channel that makes the channel private. It is derived from the seed's hash;
channelID: The ID of the UC used for sending data back to the Botmaster;
data: The data bot needs to upload;
t: The time parameter t designated by the Botmaster for the execution of VDF_eval() and VRF_proof();
t_c*:* The time duration consumed by bots executing eval function of VDF, varying in CPU types;
command: orders received from Botmaster via algorithm 4;

Output:

p_{vdf}*:* VDF's time proof;
data The results of the commands execution;
proofs: A tuple of encrypted elements including proofs of VRF and VDF and the result of the execution of the commands;

Initialize:

Execute the VDF_eval with required rounds (p_{vdf},t_c) := VDF_eval(t);
Execute the VRF_proof with t and the bot's private key sk_{bot} (p_{vrf},rn) := *VRF_proof*(sk_{bot},t)
Use rn to select a full node to connect to fn_i := *connectAndGetInformation*(rn);
Encrypt the p_{vdf}, t_c, p_{vrf}, rn and fn_i with bot's seed, obtain the proofs *proofs* := *encrypted*(p_{vdf},*data*,t_c,p_{vrf},rn,fn_i,*channelKey*)
Put the *proofs* into the UC channel *publishToChannel(channelID,proofs)*

The PoH scheme requires a prover, in the context, a bot, and a verifier (the Botmaster), to cooperate in the process that a prescribed time interval designated and delivered by the Botmaster to bots via the BC. After receiving the PoH proofs, he can check them to verify if the bot is faithful or not. Algorithm 4 is used by the bot to prove itself to be trustworthy and Algorithm 5 is for Botmaster's use in a prosecutor's manner.

Bots in the OICL run Algorithm 4 to generate the PoH proofs to be verified, and connect to a selected full node of the DLT. When the Botmaster receives the PoH proofs through our

Algorithm 5 PoH Proof Checkout

Input:

privateKey: The private key of Botmaster;
channelKey: The encryption key of the channel that makes the channel private. It can be derived from bot's seed, received in Algorithm 2;
channelID: The ID of the UC of the bot, received in Algorithm 2;
t_c*:* The Executing time of VDF_eval() Rounds t that bots send back);
CPUInfo: The information of the processor of the bot's. This was received in a transaction assembled in Algorithm 2 issued by the bot with seed and address of the it when it phones home;
CPUList: This is a database recording a bunch of types of processors. The entry of it is the time interval that a specific CPU needs to take to solve the time puzzle (t rounds) of VDF_eval();
comprativeTimeDuration this value is known by Botmaster in advance, derived by searching in CPUList with CPUInfo as the key;

Output:

proofs: encrypted PoH proofs bot sent back, including the result of the command execution;
decision: a Boolean value (true or false) derived from the proof check adjudicates whether the bot has executed the PoH faithfully. "True" represents that the prover is an actual bot, while "false" means that the prover is unreliable;

Initialize:

Get data from the bot's UC *data* = *ucdata* := *listenToUCChannel(channelID, channelKey)*
Unpack the data and obtain proofs, the result of command execution and time interval of executing VDF_eval. (p_{vdf},data,t_c,p_{vrf},rn,fn_i) := getProofAndInterval(ucdata)
Check whether the p_{vdf} is validate or not *decision* := VDF_verify(p_{vdf},t);
if *decision* == *True* **then**
    Check whether the p_{vrf} and rn are validate or not *decision* := VRF_verify(p_{vrf},rn);
end if
if *decision* == *True* **then**
    Check whether the fn_i is validate or not fn_i' := connectAndGetInformation(rn); *decision* := fn_i' == fn_i
end if
if *decision* == *True* **then**
    if the proof passes the checkout, then further check whether its CPU type complies with the table's entry possessed by Botmaster*cpuinfo* := getCPUInfo(*data*)
    compares the VDF execution time committed by this type of CPU, in order to find out if the bot had sent fake CPU information. If so, it is considered to be espionaget_c' := *CPUList* .find(*CPUInfo*, t) decision := t_c' == t_c
end if
if *decision* == *True* **then**
    The bot is an actual bot. The Botmaster can communicate with it without fear.
else
    The bot is treacherous, disconnect from its UC and send no more commands to it via SCDC lest it collects further intelligence from the botnet Disconnect(*channelID,channelKey*)
end if

designed communication protocol, he launches Algorithm 5 to verify the proofs, assessing the loyalty of the bot.

V. Efficacy and Performance Evaluation

In this section, the efficacy and performance of our botnet design are evaluated and proved via experiments and discussion. Researches like [30], [31], [34], [36], and [56] are the most representative since they are all Bitcoin based botnets, and thus, we use them as baselines in this section.

A. Efficacy of Full Node Enumeration

First, the effectiveness of Algorithm 1, namely, the BC, was examined, and part of the full nodes enumerated is listed in Table V. Most of these nodes are maintained by blockchain communities, foundations, and enterprises, constituting the backbone of the DLT and its applications. Thus, they are residents online and under heavy protection in that it is difficult to bring them down. Due to the fact that the more nodes obtained, the more robust our proposal is (because we can switch to other nodes to resume the OICL if the one we currently connect to is down), Algorithm 1 discovers as many nodes as possible. The entries listed in Table V provide the efficacy of the algorithm. As Table V shows, when we discover these nodes, some additional information is also obtained such as the port number used, the node's version, whether remote PoW is needed or not, SSL support, and the neighboring number of the node.

TABLE V
Full Nodes Enumerated by Algorithm 1

Full Node Address	Port	IOTA Version	Remote PoW	SSL	Neighbors
Node3.itrocks-latus.de	14267	0.5.6	Yes	Yes	4
Node2.itrocks-latus.de	14267	0.5.6	Yes	Yes	4
Node1.itrocks-latus.de	14267	0.5.6	Yes	Yes	5
nodes.devnet.iota.org	443	0.5.6	Yes	Yes	3
iota.fan.ch	14267	0.5.6	No	Yes	5
tr.cbus.io	443	0.5.6	Yes	Yes	1
node.cbus.io	443	0.5.6	No	Yes	4
node1.rosipay.net	443	0.5.6	Yes	Yes	5
stirrlink.dyndns.org	14267	0.5.6	Yes	Yes	5
wallet1.iota.town	443	0.5.6	Yes	Yes	4
wallet2.iota.town	443	0.5.6	No	Yes	5
pow.iota.community	443	0.5.6	Yes	Yes	6
nodes.iota.cafe	443	0.5.6	Yes	No	3
207.180.218.59	14265	0.5.6	No	No	6
207.180.202.219	14265	0.5.6	No	No	3
207.180.225.229	14265	0.5.6	No	No	4
turbor.ddns.net	14265	0.5.6	No	No	3
195.201.114.197	14265	0.5.6	No	No	5
node.iotadev.org	14265	0.5.6	Yes	No	6
81.169.167.13	14265	0.5.6	No	No	4
node2.iota.town	14265	0.5.6	Yes	No	5
173.249.16.120	14265	0.5.6	No	Yes	5
185.53.131.77	14265	0.5.6	No	Yes	4
papa.iota.family	14267	0.5.6	Yes	Yes	4
mama.iota.family	14267	0.5.6	Yes	Yes	5
hornet.eiter.tirol	443	0.5.6	Yes	Yes	4
xeevee.net	14267	0.5.6	Yes	Yes	2
65.21.34.166	14265	0.5.6	No	Yes	3
3.122.198.108	14265	0.5.6	No	Yes	2
95.217.89.99	14265	0.5.6	No	Yes	3
gewirr.com	14267	0.5.6	Yes	Yes	4

B. Response Time Comparison of OICL and the Baseline

To construct a large-scale botnet, the response time is the headfirst factor affecting the communication of the Botnet in the way of response time. Hereby, to examine the response time of our proposal, we distributed our bots to different continents, which are Europe, Southeast Asia, and North America, and put Botmaster to these places to test the latency between them, respectively (and pinned the location of the full node to Germany, because most of the full nodes of the DLT are in Europe). We define a bot's response time as the time period from when the Botmaster issues an instruction and it is successfully received by the bot over the DLT's network.

TABLE VI
EXPERIMENT RESULTS OF A BOT'S FETCHING INSTRUCTIONS AND THE GEOGRAPHICAL LOCATIONS OF THE BOT AND THE FULL NODE

Location of Bot	Full Node's Location	Minimum Fetching Time (ms)	Maximum Fetching Time (ms)	Average Fetching Time (ms)	Standard Deviation (ms)	Amount of Fetching
Kansas City, US	Mannheim, Germany	1704	2206	1864	96.706	100
Singapore, Singapore	Mannheim, Germany	3067	4273	3586	262.577	100
Mannheim, Germany	Mannheim, Germany	167	219	185	8.163	100

TABLE VII
EXPERIMENT RESULTS OF THE BOTMASTER'S PUBLISHING INSTRUCTIONS AND THE GEOGRAPHICAL LOCATIONS OF THE BOT AND THE FULL NODE

Location of Bot	Full Node's Location	Minimum Fetching Time (ms)	Maximum Fetching Time (ms)	Average Fetching Time (ms)	Standard Deviation (ms)	Amount of Fetching
Kansas City, US	Mannheim, Germany	2613	5336	3123	490.415	100
Singapore, Singapore	Mannheim, Germany	3906	5278	4466	283.045	100
Mannheim, Germany	Mannheim, Germany	758	1562	934	136.651	100

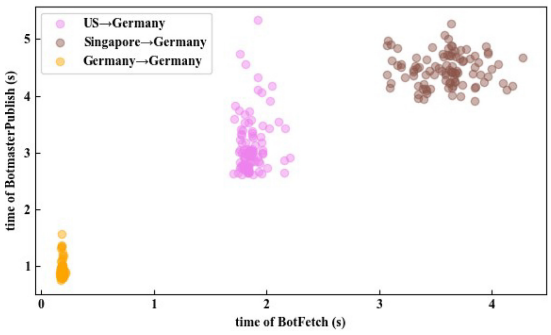

Fig. 11. Latency for data fetching and publishing on three locations.

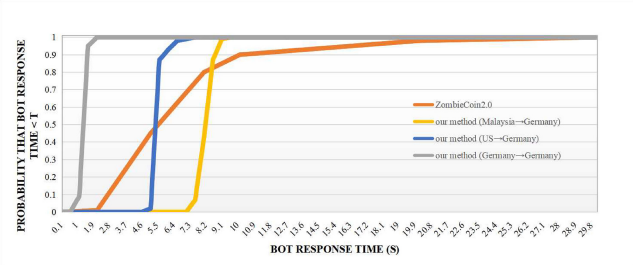

Fig. 12. Comparison of a cumulative probability distribution of bot response time between OICL and the baseline (ZombieCoin2.0).

The detail of the parameters of the experiment is listed in Tables VI and VII, and the result is depicted in Fig. 15.

The whole set of experiments is divided into two groups. In the first group, we measured the time period from the time when Botmaster issued a command to it being successfully published on Tangle (meaning that bots can obtain it) in three locations, respectively. The latency of the bot's fetching commands is also examined in these three places in the second group and each test within the two groups listed in the table was committed 100 times. The response time, referring to how fast commands or data can be brought from Botmaster to the bot (and vice versa), of three locations are rendered in Fig. 11 by its horizontal and vertical axis.

In Fig. 11, since the full node's location is constant (Germany), it stays on the right side of the three arrows in the cutline upper left of the figure. The positions of the Botmaster and bots are volatile, whose value ranges are U.S., Singapore, and Germany. So, they are arrayed on the left side of the three arrows in the cutline.

As we can see from Figs. 11 and 12, the latency is the shortest when both Botmaster and bots are in Germany, in which the full node was located. And it is the medium in the U.S. and the longest in Singapore since the geographical distance between Singapore and Germany is the longest compared with the other two. Actually, the experiment of Singapore–Singapore is an extreme situation in which both bot and Botmaster are in the same place where they are far away from the full node they are connecting to. Thereby the response time can be drastically reduced if they choose a nearby full node to link to. Therefore, Botmaster can assign a specific full node to a bot to interact with via the SCDC. Also, we compared our proposal with ZombieCoin2.0, our baseline, and as Fig. 12 shows, the result of the experiment in Germany (Botmaster, bot, and full node are all in one nation) is much better than the baseline. Another advantage of our proposal that the baseline fails to have is that the outliers of response time in ours are not too far away from the main bulk, whereas they are 100–260 s in the baseline (the longest instance is 8 s in our proposal by adding up publishing time and fetching time in Singapore).

It is obvious that the total distance between the bot, Botmaster, and full node plays as the main factor affecting the latency of communication, thereby making the selection of the full nodes enumerated by Algorithm 1 vital from the view of Botmaster. The closer, the faster (and the less number of outliers). So, we recommended that Botmaster should

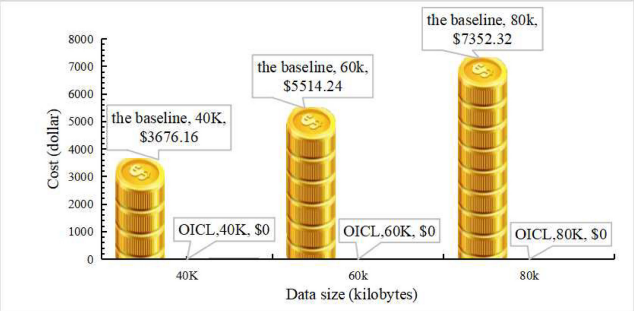

Fig. 13. Cost comparison of the data (in different sizes) transferred by a bot on the Bitcoin-based botnet and the OICL.

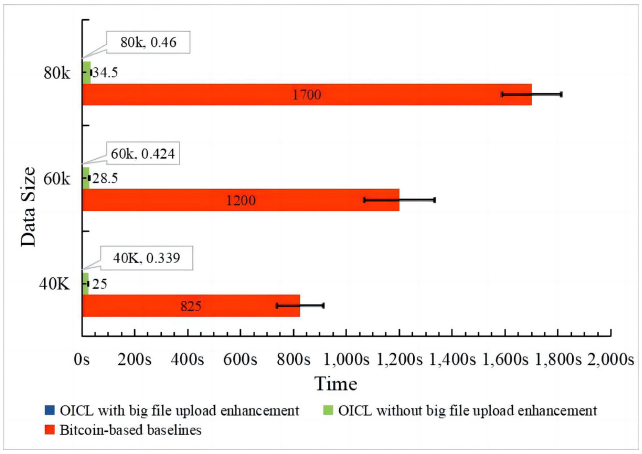

Fig. 14. Time cost comparison of the data (in different sizes) transferred by a bot on the baseline and the OICL (with and without the big file uploading enhancement).

proactively choose those full nodes as near as possible to the location of the bots or his in order to make the bots respond to his commands as quickly as they can, thereby keeping his bots on an even keel.

C. Cost Comparison

1) Economic Cost Comparison: In the experiences of cost comparison, the baselines are [30], [31], [34], [36], [37], and [56] which are all Bitcoin-based botnets. Thus, we refer to them as the baseline in this section. The cost of deploying the botnet on these implementations and the OICL is evaluated and compared in Fig. 13. The latency comparison of transferring data in multiple sizes is shown in Fig. 14.

Fig. 13 illustrates the cost of a bot sending data in multiple sizes in dollars on the Bitcoin-based baselines and the OICL. Benefiting from zero transaction fees, sending data on the OICL is free of charge compared with the baselines. The reason why sending large data is so expensive on the Bitcoin-based baseline can be found in

$$C_{\mathrm{BTC}} = n \cdot \lceil x/40 \rceil \cdot f. \tag{1}$$

In (1), n means the number of bots in the botnet that can be considered as the scale of it. And x represents the total size of the data needed to upload per day by a bot. And f is the average transaction fee in dollars on the Bitcoin network

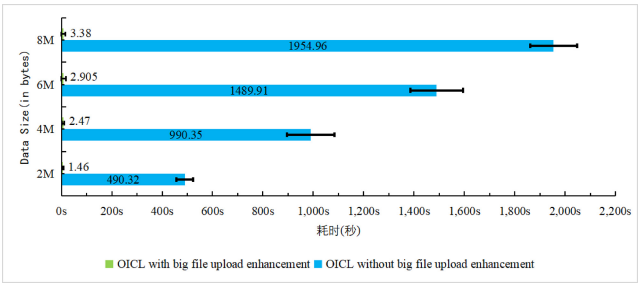

Fig. 15. Time comparison between upload enhanced and not enhanced.

(by the time of writing this article, it is U.S. $3.590 per transaction). Formula (1) demonstrates the fact that deploying a super-scale botnet on Bitcoin (Ethereum shares similar results [25] and [26]) is, to a large extent, not affordable for individual Botmasters.

2) Time Cost Comparison: Since there is no miner in the OICL, sending a transaction is far faster than other PoW-based (Bitcoin-based) baselines do. We have evaluated and compared the time cost of uploading data in multiple sizes in OICL (with and without the big file uploading enhancement) and the baseline. The result is rendered in Fig. 14.

As Fig. 14 shows, we have evaluated a bot that uploads files in 40k, 60k, and 80k bytes on the baselines and on the OICL with and without the big file uploading enhancement, respectively. It is obvious that the OICL is far faster in file uploading than it is on the Bitcoin-based baseline. We also evaluate the OICL with and without the big file uploading enhancement, respectively, in 2M, 4M, 6M, and 8M data. The results are rendered in Fig. 15.

As Fig. 15 displays, after enhancement, the OICL is not constrained by the capacity of the underlying MAM. The experiment shows that after upload enhanced, uploading data in the UC is much faster than not enhanced. Also, it is worth noting that during experiments on Bitcoin, we perceive that not just the long latency incurred by low TPS was a nuisance but also the number of transactions hindered—full nodes would ban those clients issuing a lot of transactions in a short period of time from accessing the Bitcoin network for the purpose of DDoS defense. Thereby, those who equip Bitcoin as a botnet's communication channel will likely end up in lockdown. In order to solve this problem, a Botmaster will have to set up his own full node and use it to join the Bitcoin network which entails the risk of exposing his identity to the adversary since a full node in the Bitcoin network is supposed to be discovered.

VI. Security Analysis

A. Thread Model

Unanimously, law enforcement agencies, usually backed by bureaucracy and national departments, are viewed as the thread to Botmasters. Thus, we take these entities as adversaries from the point of view of Botmasters and analyze the abilities they possess. They are able to:

1) Reclaim DNS Domain and IP address space. With the influence exerted on ISPs and other network service providers, the requisition launched by these entities,

from the point of view of Botmasters, adversaries, for the digital assets, such as DNS and IP address involved in a particular cyber incident, used for forensic postmortem is usually compulsory. When the situation accentuates, the adversaries can simply ask ISPs to reclaim the DNS domains or IP possessed by C&C servers of a botnet to take it down.

2) Require nodes, precisely, all kinds of digital devices involved in the botnet including those benign devices, for example, public digital services, such as Twitter, GitHub, etc., leveraged by Botmaster to help transfer traffic for the botnet, to shutdown to sabotage the botnet. This is a kill switch that may incur collateral damage.
3) Scrutinize the domestic network traffic. All the ISP and urban's mainframe and backbone networks, even Metropolitan Area Network, are under their oversight. That is to say, the data within is at hand for them.
4) Obtain the bot executable binary files readily in which the details and mechanism are contained. Due to the fact that one of the goals of a botnet is to infect as many devices as possible and proliferate rampantly, the program used for infection is easily obtained.
5) Fully comprehend and grasp the mechanism that a bot joins the botnet and communicates with its neighbors and Botmaster via dissecting the bot executable binary files and reverse-engineering them. That is to say, any existing crypto keys and hardcoded DNS domains and IPs will be harvested, neutralizing any symmetric encryption-based communication. Furthermore, they can artifact and customize the executable as in part the functionality as the original version has into sensor or crawler nodes that deployed to gather intelligence of botnets such as to evaluate the size of the target botnet and attempt to pollute it [20].
6) Infiltrate and lurk in the botnet to gather intelligence such as NLs of bots in the vicinity and commands of Botmaster relayed.
7) Sinkhole traffic of C&C server. Sinkholing is a tactic used by security professionals to redirect malicious botnet traffic into a reservoir under control where it is analyzed and weaponized against the malicious bot or botnet activity.
8) Identify botnets and distinguish between those network traffic generated by bots and other benign devices with miscellaneous algorithms vary from data feature mining and analyzing to various machine learning models, such as data packet periodic features demonstrated by anomaly degree [57], combination of unsupervised classification and clustering algorithms [58], multilayer neural networks [15], [59], [60], [61], [62], and even voting system [2]. This ability of differentiation between the moderate and the malicious nodes they possessed is not omnipotent, though, mighty.
9) Attempt to set up sorties, including index poisoning [63], peer list pollution, and Sybil attacks [64], into a botnet in order to undermine it as severely as possible.
10) Spam any channel writeable. If any communication channel was confirmed to be writable, for instance, those tunnels for bots' upstream and phoning home, then the adversary has the ability to tram as large size of trash data as the capacity of these tunnels has to jam the communication of the botnet. However, the messages of other bots that are not controlled by the adversary will not be corrupted or forged.

B. Spam and DDOS Protection

Due to the elimination of transaction fees, sending transactions in IOTA's Tangle is totally free of charge, meaning that the adversary might attempt to curtail the bot's phoning home process by spamming it with a huge ream of transactions holding garbage data within, confounding the Botmaster by depleting his network resources on receiving invalid transactions, also outnumbering the normal benign bot's phoning home transactions. But IOTA's Tangle has its rate control mechanism scotching spams inherently. Although there are no miners in Tangle, participants still need to solve hash puzzles when they issue transactions. However, compared with bitcoin, the PoW in Tangle is much easier and the power consumed for computing the hash puzzle is far less small. Besides, the difficulty of the hash puzzle is adjustable. This work can have different degrees of difficulty where the actual required computation time is exponential with the difficulty level: it triples with every step in the IOTA protocol [65] as the formula below shows

$$d_{n(t)} = d_0 + \lfloor \gamma \cdot r_n(t) \rfloor. \tag{2}$$

d_0 represents the base difficulty of the PoW and $r_n(t)$ represents the number of transactions issued by node n in a time interval and γ is the adaption rate parameter having a range of zero to one. At time t, for the purpose of sending transactions node n must perform PoW with difficulty $d_n(t)$ such that if it tries to emit a large number of transactions in a short period of time to jam the network, the difficulty of PoW will increase exponentially, thus making the spam attack contained.

C. Collateral Damage Binding

The notion of CDB we proposed helps prove why a parasite botnet is more secure and robust than an autarky one. Also, it serves as a protection mechanism improving the resilience of the OICL when it faces some countermeasures adversaries resort to taking to cause damage such as sinkholing, conspiring with ISP to reclaim the DNS domain, and censorship.

The concept of CDB is derived from the parasite kind of botnet such as Storm [18], which was attached to such existing network services or protocols like ED2K, to obfuscate their communication traffic with the normal traffic flowing on the premises they attached to. Although this strategy may introduce obstacles to muddy the waters when an adversary analyses the botnet's data stream, it contributes no promotion to the stamina of the botnet. That is to say, after the adversary finds out the C&C server, the whole framework will be at risk of being torn apart. Therefore, we have extended the concept of parasitizing, tapped the potential for it, and endowed it with mighty resistance to hold the adversary

at bay. Formally, when taking action to destroy a botnet can also cause damage to the benign digital service (the host) the botnet parasitizing, the CDB can be considered accomplished.

In our botnet architecture, as leveraged on IOTA's Tangle (the host), the resilience of OICL is mostly delegated by the DLT's full nodes, which not only relay and pack our raw transactions into formal ones but also do so with other common transactions, making them the mainstay of the DLT. Due to the presence of these full nodes, the adversary is almost impossible to either shut the DLT down or require these full nodes to stop working. What is more, any kind of act leading to centralization such as content review and censorship is not allowed by the DLT inherently as wiping full nodes out will impair the resilience of DLT per se. Besides, as the last defense, the broad, or virtually worldwide distribution of these full nodes also makes the adversary powerless to expropriate them.

Furthermore, we believe that it is the CDB that keeps the Bitcoin network everlasting since great amounts of electricity are consumed by the hashing PoW process to generate the correct nonce value, making its owners unwill to see the Bitcoin network being torn down. Thus, it is the prize in the real world that the DLT participators take forming one of the main factors that make the DLT so unbreakable. This prize, we name it the CDB.

The damage in CDB means the loss the DLT bears, which is undertaken by the DLT itself and the full nodes' operators (the DLT's participators). So it can be quantified in the views of the two. Via combining loss with the host, the resilience of our parasite botnet will be greatly elevated.

1) Damage Bound to Tangle's Full Node Operators: For participants in the IOTA's Tangle, there are many benefits for keeping a full node online, and they can be measured by a particular unit named mana [66]. A full node can gain mana by receiving and validating transactions sent from users. Generally speaking, mana is an incentive mechanism built to reward those participants who follow the DLT's rule and slash those who do not. Mana decays with time so it is easy to lose but hard to get. Thereby, a full node has to stay online and keep contributing to the DLT to get mana. If an operator hands over the full node to the adversary, and ceases contributing to the DLT, he or she will lose mana rapidly. The mana a full node Z has at time t, $M_Z(t)$ can be rendered by the following formula:

$$M_z(t) = M_z(0)e^{-\gamma t}. \tag{3}$$

In this formula, $M_Z(0)$ is the original mana, proportional to the specific value in the transaction, generated from processing transactions, therefore considered as a constant, and represents the decay rate for mana. As the formula illustrated, the longer the full node is offline, the more mana the operator will lose. Apparently, those full nodes that have been operated for a long time are more unwilling to be shut down, especially the IOTA's community-supported nodes being online ever since the Tangle project began, which can be our binding anchors hardcoded in the bot's program.

2) Damage Bound to the Public DLT: Theoretically, in our design, the botnet would perish only if all the full nodes crumbled. That is to say, in this situation, the odds are against, that the bulk of the DLT has been eradicated. This would barely happen since not just the decentralization and anti-SPoF trait of the DLT but also the market capitalization thwarts this repercussion from happening. As of the writing of this article, the market cap of IOTA's Tangle surpassed 2.42 billion dollars. As a result, the invulnerability of our proposal is endorsed by IOTA's Tangle itself. Assuming that v presents the incident that the IOTA's Tangle is eradicated, and $D(v)$ represents the damage bore by the DLT. Thus, in (4), $D(v)$ is equivalent to the market cap [67] of the DLT itself (which is \$3 564 683 675 by the time of writing this article). In our use case, DLT is IOTA's Tangle

$$D(v) = \text{cap}(\text{DLT}_{\text{iota}}). \tag{4}$$

VII. Discussion

In this section, spectrums to enforce OICL from the view of Botmaster as well as countermeasures by which law enforcement can conduct against OICL are discussed.

A. Enhancements

In addition to the current components of our proposal, there are some enhancements we consider as augmentations.

1) *IFF:* How to further enforce the capability of identifying the adversary's sensor in the botnet is a critical problem that once solved the resilience of the botnet will improve. Sensor detection in a botnet is such a complicated problem that most botnet implementations aforementioned in Section II do not have consideration to achieve it. However, solving it will profit Botmaster since sensors deployed by the adversary play as the vanguard in neutralizing the botnet as they gather the very first-hand intelligence of the targeted botnet via penetrating it. The key to identifying these cunning espionage is that their patterns or you can say, behaviors, are discrepant with normal bots because they are designed to bring the target down rather than becoming its accomplice. For instance, they will neither follow Botmaster's commands nor upload valuable data back. In our earlier design, we conceptualized some gravely complex reputation mechanisms to evaluate the credibility of the bot and assert whether a bot is benign or malicious. For instance, if a bot's software environment includes multiple cybersecurity analysis tools, such as reverse engineering software, network traffic sniffers, vulnerability scanners, etc., its credibility rank will be degraded and considered to be a honeypot once its credibility is lower than a certain threshold. Later, we realized that the fundamental essence a sensor or honeypot has is that it will not participate in the botnet's activities and contribute nothing to the Botmaster. Honeypots or sensors usually block egress data traffic, making them to some extent like a blackhole among other normal bots. What is more, some are smart enough to dodge this stereotype pattern by providing forged data for Botmaster.

Therefore, we decided to abandon the reputation scheme and design a new scheme, namely, the PoH, to detect these sensors and examine the authenticity of their data transferred back. As for those bots that do not upload any data back, they will be alleged to sensors or honeypots from the very outset. In addition, how to let bots detect and kick out the malfunctioning nodes instead of the Botmaster himself doing so is another interesting issue that if it is solved the automation of the botnet will be promoted and resources of the Botmaster saved. Given the idea of binding botnets to Blockchain, in the future, there might be implementations that let their bots vote on Blockchain to decide which bot among them is suspected of being espionage and the voting process does not need Botmaster to participate.

2) *Partition:* Botmasters commonly monetize their activities by partitioning botnets and leasing them as "botnets for hire." Partitioning botnets also enables multitasking and is a good damage control strategy in the case where part of the network is compromised [30]. The P2P Zeus botnet had over 200 000 bots, divided into several sub-botnets, by hardcoding bots with sub-botnet identifiers prior to deployment [68]. The Storm botnet assigned unique encryption keys to bots to distribute them into sub-botnets [69]. In our proposal, all the bots participating are separated into two groups inherently due to the communication protocol. One is those having tasks of DDoS or others that do not need to feed anything back. The other consisted of those bots that need to send digital trophies back to Botmaster, such as files, credentials, etc. Apparently, the partition for two groups is coarse-grained. In order to give the botnet more flexibility, a Bloom Filter can be introduced to separate bots into arbitrary groups to satisfy the need for fine-grained control [30]
3) *Stealth:* It is noteworthy that there is a taboo Botmasters are highly recommended not to do: operating a full node of IOTA's Tangle and then connecting themselves to it. Under no circumstances do we encourage Botmaster to operate his own full nodes on IOTA's Tangle since full nodes in most DLTs are public, for they are maintainers of them and they usually have fixed domains or IP addresses on the Internet. Thus, they can be discovered by others to join the DLT's network. In other words, a full node wants to expose itself to others. Thereby, the risk of running a full node is self-explanatory. Also, we argue that the existing covert communication methods like [70] are not fitted for botnet's C&C, since for the purpose of recruiting as many smart devices as possible, the binary execution used to turn smart equipment into a bot tends to circulate among the Internet. Thereby, there is no doubt that it will be readily obtained by such adversaries like researchers of anti-virus software vendors or digital forensic agencies, governments' security sectors, and other defenders of cybersecurity. As a result, the mechanisms for covert communication within it are wide open to these researchers, and once it has been reverse-engineered completely, any subliminal channels, no matter how deep they lurk, will be no longer covert. The most important trait of covert communication is stealth, but in this worst situation (execution binary is captured by the adversary), there will be no secret at all in the first place. But trust execution environment (TEE) implementation, such as software guard extension (SGX), may shed light on protecting the secrets within the execution binary. SGX is an isolation mechanism, aiming at protecting code and data from modification or disclosure even if all privileged software is malicious [71]. This protection uses special execution environments, so-called enclaves, which work on memory areas that are isolated from the operating system by the hardware. It is very difficult to debug, reverse engineer, or analyze the executed malware inside the enclave in any way [71]. Thus, by leveraging SGX, the covert channel within the binary will remain confidential even if the binary falls into the adversary's hands.

B. Mitigations

Although DLT parasitizing botnet derived all the attributes of its host, there are still some measures [72], [73] defenders (government and security agencies, we call them the defender instead of the adversary) can take to undermine it.

1) Collecting information as more as possible of the botnet is the prerequisite to uproot it. Although PoH can authenticate the validity of the CPU information sent back, it is powerless to find out whether other data is true or treacherous. Thus, the defender could even exploit it as camouflage by sending truthful CPU information while uploading other forged data. In this scenario, PoH itself may act like a shield for the defender's nodes.
2) Law enforcements or governments (defender) can operate their own full nodes on Tangle's network, and lure Botmaster to connect to it. Once Botmaster has linked to it, all the network connecting data including IP addresses are available to defenders. This would contribute a lot in capturing the Botmaster in the real world. But the difficulty is how to trick Botmaster to choose the controlled full nodes to connect.
3) On most DLTs, the address for sending and receiving tokens (or messages) are public, thus, once the Botmaster has sent or received messages his address will be unveiled, and anyone can browse its history of transferring data or tokens via Blockchain explorer. Thereby, defenders can keep an eye on certain suspicious addresses and find clues and patterns of their relationship with other addresses with the aid of Bigdata analysis then further achieve traceability of Botmaster's identification in the real world.
4) Another countermeasure defenders can take is to invoke operators of full nodes to exclude the transactions made by those addresses that belong to Botmaster and admonish the operators to stop relaying these transactions.

Although it sounds plausible in theory, it can only be considered as the last resort since Botmaster can change its addresses to nullify it. And the community of the DLT's operators is not so easily persuaded.

VIII. Conclusion

To summarize, we have described OICL, a new botnet model leverages public DLT to achieve botnet communication fully on-chain, and a novel sensor/honeypot detecting scheme, called PoH, that identifies spy nodes infiltrating into our botnet and promotes its resilience by reducing bots' agglomeration. Also, in security analysis, we introduce the concept of CDB, CDB in short, a botnet resilience enforcing mechanism that illustrates why the autarky botnets are weaker than parasite ones against taking down. Eliminating the overheads of cost and latency, the OICL can freely use all key strengths of the DLT it leveraging, such as decentralization, low latency, and anonymity, and it would be hard to distinguish OICL traffic between normal traffic on Tangle since OICL uses the same formatting of transactions for communication as normal Tangle users do. Besides, without depending on NL that the traditional P2P botnet relies on, common take-down techniques such as confiscating suspect Web domains, sinkholing, seizing C&C servers, or poisoning P2P networks, would not be effective. Furthermore, the three channels (the BC, UC, and SCDC) we design enable the Botmaster to readily send and receive data without worrying about sabotage by the adversary. Performance and efficiency experiments show that OICL is far more cost-saving and quick-responsive than the baseline. Also, in the view of botnet taking down, we discuss possible defense methods against this new kind of botnet. Although there is no evidence that this new kind of botnet is being put into use in the wild, we believe it will be added to the botnet phylogeny sooner or later and we hope that our work will prompt further discussions.

Conflict of Interest

The authors declare no conflicts of interest.

References

[1] J. Á. Cid-Fuentes, C. Szabo, and K. Falkner, "An adaptive framework for the detection of novel botnets," *Comput. Security*, vol. 79, pp. 148–161, Nov. 2018.

[2] M. Asadi, M. A. J. Jamali, S. Parsa, and V. Majidnezhad, "Detecting botnet by using particle swarm optimization algorithm based on voting system," *Future Gener. Comput. Syst.*, vol. 107, pp. 95–111, Jun. 2020.

[3] Y. Fu et al., "Stealthy domain generation algorithms," *IEEE Trans. Inf. Forensics Security*, vol. 12, pp. 1430–1443, 2017.

[4] J. Moubarak, M. Chamoun, and E. Filiol, "Comparative study of recent MEA malware phylogeny," in *Proc. 2nd Int. Conf. Comput. Commun. Syst. (ICCCS)*, 2017, pp. 16–20.

[5] N. Koroniotis, N. Moustafa, and E. Sitnikova, "Forensics and deep learning mechanisms for botnets in Internet of Things: A survey of challenges and solutions," *IEEE Access*, vol. 7, pp. 61764–61785, 2019.

[6] M. Prajapati and D. Dave, "Host-based forensic artefacts of botnet infection," in *Proc. Int. Carnahan Conf. Security Technol. (ICCST)*, 2019, pp. 1–4.

[7] M. Mahmoud, M. Nir, and A. Matrawy, "A survey on botnet architectures, detection and defences," *Int. J. Netw. Security*, vol. 17, no. 3, pp. 264–281, 2015.

[8] C. Cimpanu. "Avast and French police take over malware Botnet and disinfect 850,000 computers." Aug. 2019. [Online]. Available: https://www.zdnet.com/article/avast-and-french-police-take-over-malware-botnet-and-disinfect-850000-computers/

[9] A. Blaise, M. Bouet, V. Conan, and S. Secci, "Botnet fingerprinting: A frequency distributions scheme for lightweight bot detection," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 3, pp. 1701–1714, Sep. 2020.

[10] "Galileo RCS—Running an espionage operation." Jul. 2015. [Online]. Available: https://www.4armed.com/blog/galileo-rcs-running-espionage-operation/

[11] I. Makhdoom, M. Abolhasan, H. Abbas, and W. Ni, "Blockchain's adoption in IoT: The challenges, and a way forward," *J. Netw. Comput. Appl.*, vol. 125, pp. 251–279, Jan. 2019.

[12] U. Majeed, L. U. Khan, I. Yaqoob, S. A. Kazmi, K. Salah, and C. S. Hong, "Blockchain for IoT-based smart cities: Recent advances, requirements, and future challenges," *J. Netw. Comput. Appl.*, vol. 181, May 2021, Art. no. 103007.

[13] X. Zheng, Z. Cai, J. Yu, C. Wang, and Y. Li, "Follow but no track: Privacy preserved profile publishing in cyber-physical social systems," *IEEE Internet Things J.*, vol. 4, no. 6, pp. 1868–1878, Dec. 2017.

[14] H. Xia, L. Li, X. Cheng, X. Cheng, and T. Qiu, "Modeling and analysis botnet propagation in social Internet of Things," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 7470–7481, Aug. 2020.

[15] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, "Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-IoT dataset," *Future Gener. Comput. Syst.*, vol. 100, pp. 779–796, Nov. 2019.

[16] G. Vormayr, T. Zseby, and J. Fabini, "Botnet communication patterns," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2768–2796, 4th Quart., 2017.

[17] S. K. Tetarave, S. Tripathy, E. Kalaimannan, C. John, and A. Srivastava, "A routing table poisoning model for peer-to-peer (P2P) botnets," *IEEE Access*, vol. 7, pp. 67983–67995, 2019.

[18] T. Holz, M. Steiner, F. Dahl, E. W. Biersack, and F. C. Freiling, "Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm," in *Proc. LEET*, vol. 8, 2008, pp. 1–9.

[19] A. Singh, T.-W. Ngan, P. Druschel, and D. S. Wallach, "Eclipse attacks on overlay networks: Threats and defenses," in *Proc. IEEE INFOCOM*, 2006, pp. 1–12.

[20] C. R. Davis, J. M. Fernandez, S. Neville, and J. McHugh, "Sybil attacks as a mitigation strategy against the storm botnet," in *Proc. 3rd Int. Conf. Malicious Unwanted Softw. (MALWARE)*, 2008, pp. 32–40.

[21] A. Nappa, A. Fattori, M. Balduzzi, M. Dell'Amico, and L. Cavallaro, "Take a deep breath: A stealthy, resilient and cost-effective botnet using Skype," in *Proc. Int. Conf. Detect. Intrusions Malware, Vulnerability Assess.*, 2010, pp. 81–100.

[22] A. Dorri, M. Abadi, and M. Dadfarnia, "SocialBotHunter: Botnet detection in twitter-like social networking services using semi-supervised collective classification," in *Proc. IEEE 16th Intl Conf Dependable, Auton. Secure Comput. 16th Int. Conf Pervasive Intell. Comput. 4th Int. Conf Big Data Intell. Comput. Cyber Sci. Technol. Congr. (DASC/PiCom/DataCom/CyberSciTech)*, 2018, pp. 496–503.

[23] D. Mónica and C. Ribeiro, "Leveraging honest users: Stealth command-and-control of botnets," in *Proc. 7th USENIX Workshop Offensive Technol. ({WOOT}),* 2013.

[24] K. Gai, J. Guo, L. Zhu, and S. Yu, "Blockchain meets cloud computing: A survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 2009–2030, 3rd Quart., 2020.

[25] H. Gao et al., "ZombieCoin3.0: On the looming of a novel botnet fortified by distributed ledger technology and Internet of Things," in *Proc. IEEE 23rd Int. Conf. High Perform. Comput. Commun. 7th Int. Conf. Data Sci. Syst. 19th Int. Conf. Smart City 7th Int. Conf. Dependability Sens., Cloud Big Data Syst. Appl. (HPCC/DSS/SmartCity/DependSys)*, 2021, pp. 1625–1634.

[26] H. Gao et al., "BlockchainBot: A novel botnet infrastructure enhanced by blockchain technology and IoT," *Electronics*, vol. 11, no. 7, p. 1065, 2022.

[27] "The botnet chronicles: A journey to infamy," Trend Micro, Tokyo, Japan, White Paper, 2010.

[28] "What does global threat bot mean?" 2012. [Online]. Available: https://www.techopedia.com/definition/59/global-threat-bot-gtbot

[29] I. Arce and E. Levy, "An analysis of the slapper worm," *IEEE Security Privacy*, vol. 1, no. 1, pp. 82–87, Jan./Feb. 2003.

[30] S. T. Ali, P. McCorry, P. H.-J. Lee, and F. Hao, "ZombieCoin 2.0: Managing next-generation botnets using Bitcoin," *Int. J. Inf. Security*, vol. 17, no. 4, pp. 411–422, 2018.

[31] D. Frkat, R. Annessi, and T. Zseby, "ChainChannels: Private botnet communication over public blockchains," in *Proc. IEEE Int. Conf. Internet Things (iThings) IEEE Green Comput. Commun. (GreenCom) IEEE Cyber, Phys. Social Comput. (CPSCom) IEEE Smart Data (SmartData)*, 2018, pp. 1244–1252.

[32] G. Falco, C. Li, P. Fedorov, C. Caldera, R. Arora, and K. Jackson, "Neuromesh: IoT security enabled by a blockchain powered botnet vaccine," in *Proc. Int. Conf. Omni-Layer Intell. Syst.*, 2019, pp. 1–6.
[33] O. Alibrahim and M. Malaika, "Botract: Abusing smart contracts and blockchain for botnet command and control," *Int. J. Inf. Comput. Security*, vol. 17, nos. 1–2, pp. 147–163, 2022.
[34] T. Curran and D. Geist, *Using the Bitcoin Blockchain as a Botnet Resilience Mechanism*, Univ. Amsterdam, Amsterdam, The Netherlands, 2016.
[35] J. Sweeny, *Botnet Resiliency via Private Blockchains*, SANS Inst. Inf. Security Reading Group, North Bethesda, MD, USA, 2017.
[36] S. Pletinckx, C. Trap, and C. Doerr, "Malware coordination using the blockchain: An analysis of the cerber ransomware," in *Proc. IEEE Conf. Commun. Netw. Security (CNS)*, 2018, pp. 1–9.
[37] K. Eisenkraft and A. Olshtein, "Pony's C&C servers hidden inside the Bitcoin blockchain," 2019. [Online]. Available: https://research.checkpoint.com/2019/ponys-cc-servers-hidden-inside-the-bitcoin-blockchain/
[38] L. Böck, N. Alexopoulos, E. Saracoglu, M. Mühlhäuser, and E. Vasilomanolakis, "Assessing the threat of blockchain-based botnets," in *Proc. APWG Symp. Electron. Crime Res. (eCrime)*, 2019, pp. 1–11.
[39] S. Popov. "The Coordicide." 2020. [Online]. Available: https://files.iota.org/papers/20200120_Coordicide_WP.pdf
[40] "IOTA." 2023. [Online]. Available: https://www.iota.org/
[41] S. Popov, "The tangle," IOTA, Berlin, Germany, White paper, 2018.
[42] "In-depth explanation of how IOTA making a transaction (with picture)." 2018. [Online]. Available: https://medium.com/@louielu/in-depth-explanation-of-how-iota-making-a-transaction-with-picture-8a638805f905
[43] Seed. "Create a seed." Accessed: Feb. 2022. [Online]. Available: https://legacy.docs.iota.org/docs/getting-started/1.1/transfer-tokens/create-a-seed?q=seed&highlights=seed
[44] "HORNET." 2023. [Online]. Available: https://hornet.docs.iota.org/
[45] Bee. "A Framework for building IoTA nodes, clients, and applications in rust." Accessed: Oct. 2022. [Online]. Available: https://bee.docs.iota.org/
[46] "SWARM." 2021. [Online]. Available: https://www.ethswarm.org/swarm-whitepaper.pdf
[47] "IPFS—Content addressed, versioned, P2P file system." 2014. [Online]. Available: https://ipfs.io/ipfs/QmV9tSDx9UiPeWExXEeH6aoDvmihvx6jD5eLb4jbTaKGps
[48] L. Böck, S. Karuppayah, T. Grube, M. Mühlhäuser, and M. Fischer, "Hide and seek: Detecting sensors in P2P botnets," in *Proc. IEEE Conf. Commun. Netw. Security (CNS)*, 2015, pp. 731–732.
[49] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," in *Proc. Annu. Int. Cryptol. Conf.* 2018, pp. 757–788. [Online]. Available: https://eprint.iacr.org/2018/601
[50] A. K. Lenstra and B. Wesolowski, "Trustworthy public randomness with sloth, unicorn, and TRX," *Int. J. Appl. Cryptogr.*, vol. 3, no. 4, pp. 330–343, 2017.
[51] D. Boneh, B. Bünz, and B. Fisch, "A survey of two verifiable delay functions," IACR, Bellevue, WA, USA, Rep. 2018/712, 2018.
[52] B. Wesolowski, "Efficient verifiable delay functions," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.*, 2019, pp. 379–407.
[53] K. Pietrzak, "Simple verifiable delay functions," in *Proc. 10th Innov. Theor. Comput. Sci. Conf. (ITCS)*, 2018, pp. 1–20.
[54] "Poanetwork." 2021. [Online]. Available: https://github.com/poanetwork/vdf
[55] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in *Proc. 40th Annu. Symp. Found. Comput. Sci.*, 1999, pp. 120–130.
[56] K. Eisenkraft and A. Olshtein. "Servers hidden inside the Bitcoin blockchain." 2019. [Online]. Available: https://research.checkpoint.com/2019/ponys-cc-servers-hidden-inside-the-bitcoin-blockchain
[57] C.-M. Chen and H.-C. Lin, "Detecting botnet by anomalous traffic," *J. Inf. Security Appl.*, vol. 21, pp. 42–51, Apr. 2015.
[58] O. Y. Al-Jarrah, O. Alhussein, P. D. Yoo, S. Muhaidat, K. Taha, and K. Kim, "Data randomization and cluster-based partitioning for botnet intrusion detection," *IEEE Trans. Cybern.*, vol. 46, no. 8, pp. 1796–1806, Aug. 2016.
[59] A. A. Obeidat, "Hybrid approach for botnet detection using K-means and K-medoids with hopfield neural network," *Int. J. Commun. Netw. Inf. Security*, vol. 9, no. 3, pp. 305–313, 2017.
[60] M. Alauthaman, N. Aslam, L. Zhang, R. Alasem, and M. A. Hossain, "A P2P botnet detection scheme based on decision tree and adaptive multilayer neural networks," *Neural Comput. Appl.*, vol. 29, no. 11, pp. 991–1004, 2018.
[61] L. Mai and D. K. Noh, "Cluster ensemble with link-based approach for botnet detection," *J. Netw. Syst. Manag.*, vol. 26, no. 3, pp. 616–639, 2018.
[62] R. McKay, B. Pendleton, J. Britt, and B. Nakhavanit, "Machine learning algorithms on botnet traffic: Ensemble and simple algorithms," in *Proc. Proc. 3rd Int. Conf. Comput. Data Anal.*, 2019, pp. 31–35.
[63] J. Liang, N. Naoumov, and K. W. Ross, "The index poisoning attack in P2P file sharing systems," in *Proc. INFOCOM*, 2006, pp. 1–12.
[64] J. R. Douceur, "The Sybil attack," in *Proc. Int. Workshop Peer-to-Peer Syst.*, 2002, pp. 251–260.
[65] L. Vigneri and W. Welz, "On the fairness of distributed ledger technologies for the Internet of Things," in *Proc. IEEE Int. Conf. Blockchain Cryptocurrency (ICBC)*, 2020, pp. 1–3.
[66] "Explaining mana in IOTA." 2020. [Online]. Available: https://blog.iota.org/explaining-mana-in-iota-6f636690b916/
[67] "Mainnet." 2023. [Online]. Available: https://explorer.iota.org/mainnet
[68] D. Andriesse, C. Rossow, B. Stone-Gross, D. Plohmann, and H. Bos, "Highly resilient peer-to-peer botnets are here: An analysis of gameover Zeus," in *Proc. 8th Int. Conf. Malicious Unwanted Softw. Americas (MALWARE)*, 2013, pp. 116–123.
[69] R. Naraine. "Storm worm botnet partitions for sale." 2007. [Online]. Available: http://www.zdnet.com/blog/security/storm-worm-botnet-partitions-for-sale/592
[70] H. Cao et al., "Chain-based covert data embedding schemes in blockchain," *IEEE Internet Things J.*, vol. 9, no. 16, pp. 14699–14707, Aug. 2022.
[71] M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard, "Malware guard extension: Using SGX to conceal cache attacks," in *Proc. Int. Conf. Detect. Intrusions Malware, Vulnerability Assessment*, 2017, pp. 3–24.
[72] M. Alauthman, N. Aslam, M. Al-Kasassbeh, S. Khan, A. Al-Qerem, and K.-K. R. Choo, "An efficient reinforcement learning-based botnet detection approach," *J. Netw. Comput. Appl.*, vol. 150, Jan. 2020, Art. no. 102479.
[73] G. De La Torre Parra, P. Rad, K.-K. R. Choo, and N. Beebe, "Detecting Internet of Things attacks using distributed deep learning," *J. Netw. Comput. Appl.*, vol. 163, Aug. 2020, Art. no. 102662.

Haoyu Gao was born in Shanxi, China, in 1994. He received the B.Sc. degree in software engineering from Nanjing University Jinling College, Nanjing, China, in 2018, and the M.Sc. degree in software engineering from Inner Mongolia University of Technology, Hohhot, China, in 2022. He is currently pursuing the Ph.D. degree in cyber-security with Hainan University, Haikou, China.

He also works as an Intern-Researcher with Oxford-Hainan Blockchain Research Institute, Chengmai, China. His research interests include decentralized computing and learning, and blockchain consensus.

Leixiao Li was born in Shandong, China, in 1978. He received the M.A. degree in engineering from Inner Mongolia University of Technology, Hohhot, China, in 2007, and the Ph.D. degree in engineering from Inner Mongolia Agricultural University, Hohhot, in 2019.

He is with Inner Mongolia University of Technology, where he is currently a Professor with the School of Data Science and Application and also with the Research Center of Large-Scale Energy Storage Technologies, Ministry of Education of the People's Republic of China, Beijing, China. His research interests include cloud computing, data mining, and big data processing.

Hong Lei received the bachelor's and master's degrees from Beihang University, Beijing, China, in 2006 and 2009, respectively, and the Ph.D. degree from Michigan State University (MSU), East Lansing, MI, USA, in May 2015.

Then, he was a Postdoctoral Fellow with the Smart Microsystems Laboratory, MSU. He joined Schweitzer Engineering Laboratory in 2016 and then joined the Department of Electrical and Computer Engineering as a Tenure-Track Assistant Professor with Portland State University, Portland, OR, USA, in July 2018. He was appointed as the Associate Dean of Oxford-Hainan Blockchain Research Institute, Chengmai, China, in June 2019. He is currently a Professor with Hainan University, Haikou, China, and doing researches on TEE and blockchain.

Ning Tian received the bachelor's degree from Jinan University, Guangzhou, China, in 2015, and the master's degree from the University of Warwick, Coventry, U.K., in 2016. She is currently pursuing the joint Ph.D. degree with the University of Warwick and Hainan University, Haikou, China.

Her research interests include blockchain technology and digital identity.

Hao Lin was born in Tianjin, China, in 1995. He received the B.A. degree in engineering from Tianjin University of Technology and Education, Tianjin, in 2018, and the M.A. degree in engineering from Inner Mongolia University of Technology, Hohhot, China, in 2021. He is currently pursuing the Ph.D. degree in engineering with Tianjin University of Technology, Tianjin.

His research interests include cyberspace security, social engineering attack, and data mining.

Jianxiong Wan received the B.Sc. degree in computer science from Shanxi Normal University, Xi'an, China, in 2004, the M.Sc. degree in management science from Beijing Information Technology Institute, Beijing, China, in 2009, and the Ph.D. degree in computer science from the University of Science and Technology Beijing, Beijing, in 2013.

He was a Postdoctoral Research Fellow with Massey University, Palmerston North, New Zealand, from 2016 to 2017, and also a Visiting Researcher with Nara Institute of Science and Technology, Ikoma, Japan, from 2017 to 2018. He joined Inner Mongolia University of Technology, Hohhot, China, in 2013, where he is currently a Professor with the School of Data Science and Application. He is also with the Research Center of Large-Scale Energy Storage Technologies, Ministry of Education of the People's Republic of China, Beijing. His research interests include dynamic optimization, intelligent control, and performance modeling of distributed systems, with special focuses on cloud-scale systems.