IoT Security Function Distribution via DLT

Le Su*, Dinil Mon Divakaran†, Sze Ling Yeo‡, Jiqiang Lu§ and Vrizlynn Thing¶
*Nanyang Technological University, Singapore † Trustwave, ‡Institute for Infocomm Research, A*STAR, Singapore,
§School of Cyber Science and Technology, Beihang University, China, ¶National University of Singapore
lsu1@e.ntu.edu.sg, dinil.divakaran@trustwave.com, slyeo@i2r.a-star.edu.sg, lvjiqiang@buaa.edu.cn, vrizlynn.thing@gmail.com

***Abstract*—With the rapid deployment of IoT devices, there is an increasing concern on the security and privacy of the devices. We are now witnessing newer and higher intensity attacks that exploit IoT devices. Although security-by-design is important and necessary, the effectiveness and sustainability of the build-in security defense may still be questionable. This has created new opportunities for third-party security service providers to enter the market. In this work, we leverage the distributed ledger technology (DLT) to propose a solution for distributing IoT security functions. We design the system architecture and describe the different types of operations to be executed. Our system also allows for reputation scoring that further adds credibility to the security functions distributed in the network.**

I. Introduction

In a few years, a house will have a number of IoT devices, with most of them connected to the Internet via a router or a gateway, and unattended by humans. Many devices are being manufactured with little consideration for security, and patches might simply be not available even if vulnerabilities have been known and exploited. Even today, when there is most often a human behind a computer in use, vulnerabilities are discovered frequently and regularly. Yet, the patches are late and many times not applied, thus leaving these systems vulnerable to exploits. When it comes to IoT devices, the security-related problems are expected to increase significantly, for the above-mentioned reasons [1], [2].

As IoT devices are manufactured by a number of vendors, the devices could become obsolete, with no official continuous support for security from the vendors. Besides, the security functions (e.g., security patches, IDS, etc.) may be developed by or outsourced to third parties. In such a vibrant, emerging and crucial market, where the security of the devices is not guaranteed, we argue that it is necessary to build platforms that will help the fast development and deployment of reliable security solutions. We envision a future where anyone can develop security functions (also henceforth referred to as SFs) for IoT devices, offering security-as-a-service. This can potentially lead to the budding of a large number of security solution providers (SSPs) competing to develop SFs for vast numbers and types of IoT devices out in the market. The challenge to this vision is this: how is it possible to verify and validate SFs developed by any arbitrary provider (some of whom would be start-ups and inexperienced) in such a way that the time-to-market is as short as possible?

All authors were with Institute for Infocomm Research, A*STAR, when this work was carried out.

Workshop on Decentralized IoT Systems and Security (DISS) 2020
23 February 2020, San Diego, CA, USA
ISBN 1-891562-64-9
https://dx.doi.org/10.14722/diss.2020.23009
www.ndss-symposium.org

We consider the problem of fast and efficient development of security functions for IoT devices, particularly in the context of smart homes. In this context, we envision the home Internet gateway of IoT devices to be the last mile of defense [3], [4]. That is, the responsibility of protecting the devices rests on the gateway that connects the IoT devices to the Internet. We expect an IoT gateway to be “intelligent” in the sense that it can deploy security functions to and for the devices it protects. The gateway will carry out functions such as patching of the devices, inspection of device traffic (to check for botnet patterns), manipulating the packets (say, to use a more secure protocol than that used by the devices), etc.

In view of the above motivation, we develop and present a system design that aims at distributing legitimate and correct security functions through a network of IoT gateways, quickly and without a central authority leveraging the distributed ledger technology (DLT). At the same time, the SSPs that develop efficient and correct security functions tend to gain reputation in the system, over others that develop buggy and ineffective functions. The key features of our system are:

(i) It is a decentralized network for efficient distribution and deployment of security functions.

(ii) All transactions achieve integrity, authenticity and non-repudiation.

(iii) Our solution brings forth important practices that exist today for software purchases, e.g., allows the gateways to test a trial version of the function before deciding whether to purchase it or not. The system further motivates gateways to provide user feedback, and circulate the effectiveness of the security function in the network.

After an overview of our proposed system, we present the system design in Sec. III, with transactions being defined in Sec. IV. In Sec. V, we explain how the system could be implemented. Sec. VI touches on the reputation system, and Sec. VII discusses potential attacks and their counter measures.

II. System Overview

Our system assumptions are the following:

- Gateways are resourceful machines; they have large storage and they can also perform heavy computations (similar to other recent proposals, e.g., [4]). They are also connected to the Internet with the state-of-the-art bandwidth capacity of the day (say, 1 Gbps).
- Each gateway has its own public-private key pair, which could be used for generating digital signatures.

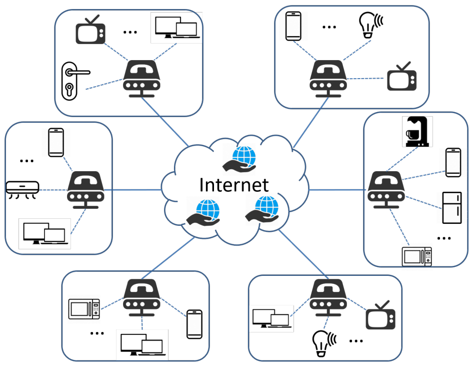

Fig. 1: System Architecture

- Each device in an IoT network trusts its gateway, such that the gateway could act on behalf of its devices.

Our system consists of a network, which we refer to as the distributed ledger network, and contains two types of nodes (or users)—IoT gateways and SSPs. We expect every smart home to have a gateway; obviously the gateways outnumber the SSPs in orders of magnitude. As in a traditional DLT network, all nodes connect to form a P2P network; the P2P network of nodes stores the decentralized database of information. Each node has a cryptographic key pair (public and private key) used to digitally sign and verify messages. Fig. 1 depicts our system architecture from a high level: in the center it is the distributed network, where the SSPs reside. Further, the gateways are connected to the Internet (and subsequently to the SSPs). Each gateway is also connected to multiple IoT devices, such as smart cameras, smart TVs, etc. We assume the system is owned by an alliance of Internet Service Providers (ISPs), such as the Global Telco Security Alliance [5].

We give a summary of our system, before proceeding to present the details. An SSP can develop and release an SF for any product type. A release of an SF always includes a trial version. If a gateway decides to verify and apply a particular solution (e.g., to fix a security flaw of one of its devices), it downloads the trial version of the SF from the SSP. The gateway applies and evaluates the SF, and subsequently reports the outcome to the network. These review reports of the gateways inform the network of the effectiveness of the SFs that are being released into the market (the distributed ledger network), consequently leading to build a reputation for the various SFs. This eventually also helps build a reputation for the corresponding SSP that developed the function(s). Furthermore, the gateway, after testing the trial version, has the right to decide whether to purchase the SF or not.

The above process is realized through a series of transactions, which further trigger the corresponding smart contracts to execute subsequent actions. These transactions are recorded in the ledger, and verifiers could validate the authenticity and the integrity of these transactions.

III. System Design

As previously mentioned, a security function (SF) is an independent software module that can be installed either at a gateway or an IoT device, for improving the security resilience of the device. This could be a firewall for one or more devices, an IDS/IPS system, a DPI (deep packet inspection) module, an anomaly detection solution to check for suspicious patterns on network traffic [6], a forensic analyser [7], a security patch, a firmware update, etc. A gateway can run multiple such SFs as VMs or containers, as is the case with network functions today.

Our system is an interaction between two types of entities, namely SSP and gateways. Their interactions are achieved through a set of transactions that invoke pre-defined smart contracts. We now provide a detailed description for each of the above components:

A. Entities

There are two types of entities in our system:

Gateway: Gateway is a machine connecting (authorized) IoT devices in a premise (smart home) to the Internet, and could be identified by the IP address assigned by its ISP. It also acts as a controller that analyzes, processes and manipulates IoT traffic to protect the devices. In essence, the gateway plays the role of a security guard with high computational and storage capabilities.

Security Solution Provider: The SSP is an entity that develops and provides security function(s) related to one or more IoT device types. It could be the device manufacturer/vendor, or could be any third-party service provider.

B. Roles

There are two roles in our system:

Transaction participant: A participant is either an SSP, or a gateway. To fulfill their requests, they need to initiate transactions with the smart contracts to invoke the corresponding actions.

Verifiers: The verifiers in our system depend on the DLT used for implementation, and is discussed in Sec. V.

C. Transactions and Smart Contracts

In our system, we define five types of transactions: *register*, *release*, *interest*, *review* and *purchase*. Each transaction will trigger an associated smart contract, to complete the corresponding tasks. A smart contract can be triggered by a participant or another smart contract.

Below, we first describe the general format of a transaction and then specify the details. Furthermore, we describe, at a high level, each transaction-accompanying smart contract. In our system, a transaction has the following format:

Txn Type	Gateway Info	SSP Info	Smart Contract Info	Amount	PreTxnLink	Digital Signature

Txn Type: The transaction (Txn) type field is filled with one of the five possible types defined above.

Gateway Info: This field is further divided into two sub-fields:

Gateway Info	
Gateway ID	Data

The "Gateway ID" is an identifier that uniquely represents this particular gateway, such as the unique public key associated with it. The "Data" field is used for storing transaction-specific data, such as the digital signature of the device during registration, or an SF evaluation report for the review transaction. For different transactions, one or more sub-fields of the "Gateway Info" could be left empty.

SSP Info: The SSP information field is similar to the "Gateway Info" field, and is divided into two sub-fields:

SSP Info	
SSP ID	Data

Each sub-field is filled with similar information as described for the "Gateway Info". One or more sub-fields could be left empty according to transaction types.

Smart Contract Info: Each transaction interacts with the associated smart contract to trigger a set of actions. In other words, each transaction will have to include the smart contract address, to identify the corresponding contract from the pool of stored contracts. This field is therefore filled with the address of the particular smart contract (e.g., its public key).

Amount: This field stores a monetary value associated with a particular transaction. It could be left empty for certain transactions if they do not involve any monetary transfer.

PreTxnLink: The "Previous Transaction Link" field is filled with a link to the previous transaction that is related to this current one. The definition of the "previous transaction" will be described later. The purpose of this field is to chain the transactions together, to prevent possible forgery or modification of transactions once they have been recorded.

Digital Sig: This field is used by each transaction initiator to digitally sign this particular transaction, to provide authenticity, integrity, and non-repudiation for this transaction. A standard signature scheme such as RSA could be employed.

Any transaction also includes a timestamp field, to record the time it is created. As it is a standard field, we omit it here.

IV. Transactions in the Distributed Ledger

A. *Register*

Every first-time participant needs to register in the network to avail the services. The registration process is similar to IoT gateways and SSPs.

Gateway registration: a gateway needs to register itself; and the type field in the corresponding transaction would be *register*. As this is a gateway self-registration, the "Gateway ID" is filled with the gateway address. This would be, for example, the public key of this particular gateway. To prevent potential Sybil attack, the system further requires the gateway to provide a valid certificate of ownership obtained from its governing ISP, and embed it into the "Data" sub-field. As an additional measure against the fake gateway registration, the gateways need to pledge a small amount of deposit during registration; thus the "Amount" field is filled with the corresponding system-defined monetary value.

Since there is no SSP involved in this transaction, the "SSP Info" field is left empty. For self-registration, the transaction needs to trigger the corresponding *register* smart contract. This smart contract is pre-written and stored on the distributed ledger, and thus the gateway needs to specify the smart contract address under the "Smart Contract Info" field to execute the corresponding code. For the "PreTxnLink" field, as there is no transaction earlier than the self-registration for this gateway, by default we let this field be left empty. Finally, the gateway produces a signature based on all the previous information and stores that in the "Digital Sig" field.

When the gateway initiates this transaction, the smart contract associated with the *register* is triggered. Taking the gateway address (e.g., its public key) as input, the smart contract will:

1) Check whether this gateway has been registered;
2) Verify the digital signature is valid (as the signature verification is mandatory for all subsequent actions, we will omit to describe it in the subsequent ones).

If the above two conditions are met, the smart contract outputs 1, and 0 otherwise. For all subsequent transactions, the digital signature field is generated similarly, using all the previous fields as the underlying message to be signed.

SSP registration: The SSP registration is very similar to the gateway registration. The "SSP Info" field is filled with its address (such as its public key). The transaction designates to the corresponding *register* smart contract and appends with an SSP digital signature. Furthermore, we require the SSP to pledge relatively large collateral during registration (using the "Amount" field). Thus, we increase the cost of a malicious entity to register legitimately into the system and distribute flawed programs/binaries as SFs.

B. *Release*

The *release* transaction is created only by the SSPs, whenever they have a new SF to release to the network, to make it available to the gateways. There is a smart contract associated with this transaction, in which a deposit amount from the SSP is stored. In this transaction, the sub-field "Data" under "SSP Info" is no longer filled with the digital signature; instead it is filled with a link value (for example, a short URL, or address in a repo stored at an SSP) that directs to the SF offered by the SSP. Such a link is assumed to have limited access (say, using authentication), the scope of which is up to the SSP to decide. The "Amount" field specifies the deposit amount the SSP has to pay. The reason behind mandating the deposit is to discourage and disincentivize the SSPs from offering low-quality solutions. A portion of the deposit will be deducted, if and when, under the *review* transaction, a gateway has reported failure feedback for this particular solution. The solution will no longer be offered to gateways (as it will be automatically ceased by smart contract) if the deposit amount has fallen below a threshold value defined by the community/alliance maintaining the network.

The corresponding *release* smart contract, upon receiving the transaction, will check whether the SSP has registered itself, and the correctness of the "PreTxnLink" field and signatures. Upon the successful verification, the smart contract will store the deposit value for future use.

C. Interest

This transaction is created by gateways. With this transaction, the gateway and the SSP enter into a contract, wherein the SSP passes a trial version of the SF to the gateway. There is an expiry time for the trial version, beyond which the SF is expected not to work. The "Data" sub-field of "Gateway Info" is left empty, as it is not required to append the gateway signature in this transaction. The "Data" sub-field of "SSP Info" should have the link of the SF, which the gateway is interested to purchase for securing its device. The transaction is linked with the *interest* smart contract, with its address filled in the field.

For this transaction, the gateway also deposits money into the smart contract, which will be transferred back to it after submitting a review report of the SF. A review of the SF explicitly informs the network whether the solution worked for the IoT device or not. By having a gateway deposit an amount for the trial, the system incentivizes a gateway to submit a review report of the solution it has tested. The deposited amount may be split between the SSP, the gateway and the system owner (e.g., the ISP alliance) according to system pre-defined rules, if the gateway does not perform the *review* transaction (described below) within a specified time. We also define a *review deadline*: a time by which gateway has to provide a review report of the corresponding solution. The review deadline should give gateway sufficient time to try out the trial solution; that is, it should be longer than the expiry time of the trial version.

Upon receiving this transaction, the associated *interest* smart contract performs the following checks:

1) Verify that the same gateway has not submitted an interest transaction for the same SF previously;
2) Check for sufficient deposit balance;
3) On the expiry of the *review deadline*, check whether there exists a *review* transaction initiated by the gateway for this solution. If there exists no such transaction, the deposit will be forfeited and divided according to pre-defined rules; otherwise the deposit will be returned to the gateway.

The first check is to prevent the gateway from behaving maliciously, by testing an SF multiple times. If the gateway has previously initiated an *interest* transaction, it means it has already tested the same SF. Our system prevents a malicious gateway in attempting to influence the reputation of an SF or SSP. Second, by checking if there is sufficient deposit, the smart contract makes sure that the SF is still valid for gateways to try out. If the deposit falls below a threshold, it means there are many failure reports for this particular SF.

D. Review

Once a gateway tests a security function obtained from an SSP, it may choose to review the SF, via a *review* transaction. As mentioned above, there is an incentive to do so. In a review transaction, the gateway specifies the outcome of the evaluation. Strictly speaking, from a security point of view, there are only two outcomes — success or failure; i.e., the SF either succeeded in removing/mitigating the vulnerability or it failed. Therefore, in the review transaction, the gateway has to specify either success or failure, corresponding to the outcome of the evaluation. (An enhancement to this approach would be to consider multiple scores for a rating.) The system further allows the gateway to submit another *review* transaction after it has purchased the full version, as some of the trial versions do not provide full functionalities. The transaction is similar to the *interest* transaction, except 1) the "Data" sub-field of the gateway info is filled with the outcome of testing the SF (i.e., either success or failure); 2) the "Amount" field is zero, as there is no money transfer in this transaction.

When a gateway wants to report that the SF fails, it specifies failure in the *review* transaction. However, there is a possibility that a gateway may lie and report a negative review for an SF. Therefore, once a gateway reports an SSP offered SF as failed, the gateway thereafter also becomes ineligible to purchase the particular solution. This scheme discourages the gateway from lying when the SF works. If the SF works successfully, the gateway specifies success in its transaction. The previous transaction would be the *interest* transaction for this particular SF. While it is possible that an SSP incentivizes a set of gateways to report positive reviews of its SF even if it fails, in a large network, an SSP will have to provide incentives to a high percentage of gateways to beat the system successfully. This might be too expensive for an SSP to make eventual gains from the market.

Upon receiving the transaction, the associated smart contract performs the following actions:

1) Check the gateway has initiated an *interest* or *purchase* transaction earlier (the "previous transaction");
2) Check the past *review* transactions of this gateway for this particular SF;
3) Based on the test outcome, recompute the reputation of the SF, and deduct money from the deposit if it is set to failure;
4) Trigger the *interest* smart contract to refund the deposit to the gateway.

With the second action, we limit a gateway to submit a review only two times for a given SF: one after the trial, and one after the actual purchase (i.e., after the full version is tested for longer duration). For the third action, as we also propose to have a reputation system for the solutions and SSP, this particular smart contract will re-compute the reputation score whenever there is a review transaction. Once the gateway has submitted a review, the deposit will be returned to it, as described in the last action.

E. Purchase

Created by a gateway, the *purchase* transaction is to purchase an SF from an SSP. The "Amount" field is now filled with the actual amount to be paid to the SSP. In our system, we allow a gateway to purchase an SF even if it has not tested it before (i.e., there is no *interest* and *review* transaction initiated by the gateway for the particular device). This is reasonable, as a gateway may have purchased other SFs from the same SSP before and trusts it, or simply decides not to spend the effort to test the trial version based on the reputation score. In this case, the previous transaction used in the "PreTxnLink" field varies: it could be either a *register* or *review* transaction.

Upon receiving the transaction, the associated *purchase* smart contract performs the following:

1) If there exists a previous *review* transaction and the outcome is set to success, compare the “PreTxnLink” field using this transaction. If the outcome is a failure, discard the transaction (the gateway is not allowed to purchase the SF if it has reported a failure earlier). If no such *review* transaction exists, search and use the gateway registration transaction as the previous transaction, and verify the “PreTxnLink” field.
2) Re-compute the reputation score for the SF.
3) Transfer the payment to the SSP.

V. System Implementation

A. Naïve approach based on blockchain

Different from the traditional blockchain (e.g., the Bitcoin), in our system, a block consists of transactions related to a particular SF. This is useful in computing the reputation score of SFs efficiently. Each block should have the latest reputation score computed and stored for *one and only one* SF. To achieve this computation efficiently, in addition to the hash address that links the current block to the previous block, a new block also contains the hash address of the previous block that has transactions for the *same SF*. With this additional link, a verifier (equivalent to a miner in traditional blockchain) can now compute the reputation of an SF by simply accessing the last block related to the SF, and adding the scores due to transactions in the current block. For *register* transactions, as they are not related to any particular SF, they will be added to the latest block together with other related SF transactions, based on the transaction timestamp.

As in all blockchain networks, verifiers are responsible for creating blocks of transactions to be added to the ledger. We mandate that only gateways can be verifiers, to prevent SSPs from gaining any advantage in the process. Besides the task of gathering the transactions and creating a block, the verifiers are also required to verify each transaction. For such a transaction verification, a verifier repeats the same actions as the smart contracts do for each transaction: verifies that the conditions listed are satisfied, as well as ascertains that the smart contracts have faithfully executed the stipulated actions. We envision our framework to be a *permissioned* blockchain system that is controlled by a group of entities such as the ISPs as mentioned earlier. With this setting, one could use Byzantine Fault Tolerance (BFT) or its variant [8] as the consensus mechanism to achieve a much faster community agreement process, compare to the permissionless blockchain that uses Proof-of-Work or Proof-of-Stake consensus.

B. Corda

The distributed ledger Corda [9] is better suited for implementing our system. Corda is designed for a permissioned platform, where consensus requires the participation of only the concerned parties involved in the transaction (gateways and SSPs in our case) and a notary cluster. The ISP alliance is a suitable notary in our system—while the ISPs may be interested in forming an alliance to tackle threats and attacks more effectively, they need not necessarily trust each other. In Corda, each participating entity (gateway/SSP) needs to be uniquely mapped to a public key and IP address. This requirement works perfectly with our smart-home setting; a gateway (a home router today) already obtains IP address from the ISP providing the broadband connection, and can additionally generate public-private key pairs. Recall, as mentioned in Sec. IV-A, a gateway in our system also needs to obtain a certificate of ownership from its governing ISP, thus having a legal binding. Similarly, with Corda, we can have legally binding identity for SSPs too (in addition to IP addresses and public keys). Note that, each of the transactions defined in our system already has an associated smart contract (with clearly defined output) that is triggered for the transaction, and this is well-aligned with the concept of transaction validity in Corda. Further, Corda does not restrict a system to any specific consensus protocol; among others BFT can be used as a pluggable consensus algorithm by the notary cluster [9]. With simple modifications to our transactions and the recording of information and data, our system can be implemented using Corda. Due to space limitation, we omit the details.

VI. Reputation System

The building of reputation (or rating) systems is a well-researched topic (see [10], for example), and we do not intend to go into the details here. Nevertheless, we briefly discuss how our solution could help in building reputations for the SSPs and the SFs they develop. The reputation of SFs is computed by the verifiers during the process of verification of transactions. The reputation score for a new untested solution is initialized to zero and is updated and stored in each block. Since a block has only transactions related to one particular SF, there is only one such reputation score in each block. For every positive (success) review report submitted by a gateway, a “reputation increment” function is executed, and for every negative (failure) review report submitted, a corresponding “decrement” function is executed. Finally, for all successful purchases, a “reputation increment via purchase” function is executed. All three functions modify the reputation score, and the exact definition of these functions is dependent on the reputation model selected for deployment.

VII. Security Analysis

In this section, we provide a security analysis of our proposed system, describing several potential attacks and their corresponding countermeasures.

Sybil attack — malicious SSPs registering multiple gateways to game the reputation system: An SSP can always register a gateway, but recall that the gateway needs to present the certificate of ownership from an ISP for the *register* transaction. Therefore, if malicious SSP purchases multiple gateways, then this can be easily detected since the certificates used for registering the gateways will indicate the same name of the client/user. Alternatively, if the malicious SSP uses different names for purchasing from one or more ISPs, this forms an illegal act, and when detected, it would be dealt in the same way as it is done today. Since the system is owned by an alliance of ISPs, this kind of illegal registration across ISPs is also easy to detect.

SSP colluding with legit gateways: Nothing in our system stops an SSP from colluding with (users of) legitimate gateways. However, this is not a problem specific to only our system, rather it is common among other similar systems, such as mobile app rating, rating of products on e-commerce platforms, movie reviews, etc., where there are instances of collusion among users, as well as between users and service providers (e.g., in the case of review of restaurants or accommodation) [11]. Solutions that are being developed to detect and mitigate collusion in such systems (e.g., [11], [12]), can be applied for ours as well.

Malicious entity disguising as an SSP to distribute malware as security function: In our system, anyone can register as SSP; therefore, there is a possibility that an attacker registers as SSP to distribute malware. While this is technically possible, observe that our design has increased the cost of doing so, by mandating payment of large collateral during the registration process. There is also a deposit made for every SF released on to the network. Beside and beyond this, the malicious entity would need to obtain a good reputation score to attract a large number of users, and this would require the entity to develop a good SF (that then may embed a back door or malicious code). Finally, making such malware available to our system, also risks the possibility of being easily and quickly detected, as the alliance of ISP can always analyze the SFs by registering authentic gateways for such specific purposes.

VIII. Related Works

Content distribution in a P2P network has been studied for decades. The application spreads across various disciplines, such as privacy-preserving content distribution [13], video-on-demand systems [14], file sharing [15]. There also exist patents on the same topic [16]–[18]. The major difference between the traditional content distributions in a P2P network with our distributed ledger-based system is, all actions in our system (in the format of transactions) are recorded in the ledger. These actions are integrity-checked, authenticity-verified and non-repudiation-achieved through cryptographic primitives, and carried out by smart contracts and verifiers in the system. Our system assumes minimum trust among the participants, and any modification on the transactions is easily detectable. A few surveys discussed about the combination of blockchain with IoT, regarding the security, efficiency, and scalability. Reader may refer to [19], [20] for more information.

A recent work [21] proposed a blockchain-based firmware update system for embedded devices in an IoT environment. The system architecture defined a few transactions and their specifications, as well as the procedure to achieve the firmware update. Despite its merit, however, our system has major differences and advantages compare to it. We utilize smart contracts to achieve much richer functionalities throughout the distribution of security functions, while in [21] the traditional blockchain network might not be able to achieve the proposed functionalities. Our system incentivizes the participants to behave faithfully and penalizes them otherwise, besides inherently incorporating easy computation of reputation scores.

IX. Conclusion

Motivated by the significant increase in the deployment of IoT devices and the associated security breaches and attacks, in this work we propose a distributed ledger-based IoT device security solution distribution system. We described the overall system architecture, detailed the different types of actions that may occur, and provided concrete design by specifying the transaction format and smart contract rules. One possible future direction is to implement and test out the proposed system in a small scale IoT environment.

Acknowledgement

The authors thank the anonymous reviewers and Prof. Pekka Nikander, for their constructive comments and feedback that helped improve this paper.

References

[1] Manos Antonakakis *et al*. Understanding the Mirai Botnet. In *Proc. 26th USENIX Security Symposium*, pages 1093–1110, 2017.

[2] Symantec. ISTR 2019: Internet of Things Cyber Attacks Grow More Diverse. https://www.symantec.com/blogs/expert-perspectives/istr-2019-internet-things-cyber-attacks-grow-more-diverse; accessed: Feb. 2020.

[3] D. M. Divakaran, R. P. Singh, K. S. K. Liyanage, M. Gurusamy, and V Sachidananda. ADROIT: Detecting Spatio-Temporal Correlated Attack-Stages in IoT Networks. In *NDSS DISS Workshop*, 2020.

[4] V. Thangavelu, D. M. Divakaran, R. Sairam, S. S. Bhunia, and M. Gurusamy. DEFT: A Distributed IoT Fingerprinting Technique. *IEEE Internet of Things Journal*, 6(1):940–952, Feb 2019.

[5] Global Telco Security Alliance, 2019. https://www.singtel.com/about-Us/news-releases/global-cyber-security-aliance-formed-by-etisalat-singtel-softbank-and-telefni, accessed: Feb. 2020.

[6] I. Nevat, D. M. Divakaran, S. G. Nagarajan, P. Zhang, L. Su, L. L. Ko, and V. L. L. Thing. Anomaly Detection and Attribution in Networks With Temporally Correlated Traffic. *IEEE/ACM Transactions on Networking*, 26(1):131–144, Feb 2018.

[7] Dinil Mon Divakaran, Kar Wai Fok, Ido Nevat, and Vrizlynn L.L. Thing. Evidence gathering for network security and forensics. *Digital Investigation*, 20:S56 – S65, 2017.

[8] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI*, pages 173–186, 1999.

[9] Mike Hearn. Corda: A distributed ledger. *Corda Technical White Paper*, 2016.

[10] Arash Molavi Kakhki, Chloe Kliman-Silver, and Alan Mislove. Iolaus: Securing Online Content Rating Systems. In *Proc. WWW*, 2013.

[11] M. Allahbakhsh *et al*. Collusion detection in online rating systems. In *Web Technologies and Applications*, pages 196–207, 2013.

[12] M. Allahbakhsh and A. Ignjatovic. An Iterative Method for Calculating Robust Rating Scores. *IEEE TPDS*, 26(2):340–350, Feb 2015.

[13] Amna Qureshi, David Megías, and Helena Rifà-Pous. Framework for preserving security and privacy in peer-to-peer content distribution systems. *Expert Systems with Applications*, 42(3):1391–1408, 2015.

[14] Bo Tan and Laurent Massoulié. Optimal content placement for peer-to-peer video-on-demand systems. *IEEE/ACM Transactions on Networking (TON)*, 21(2):566–579, 2013.

[15] Spyridon Mastorakis, Alexander Afanasyev, Yingdi Yu, and Lixia Zhang. ntorrent: Peer-to-peer file sharing in named data networking. In *Proc. ICCCN*, pages 1–10, 2017.

[16] Joseph Boyd and Peter Marcotte. Peer-to-peer content distribution, May 26 2011. US Patent App. 13/055,641.

[17] Yang Guo, Saurabh Mathur, and Kumar Ramaswamy. Performance aware peer-to-peer content-on-demand, September 16 2014. US Patent 8,838,823.

[18] Todd R Manion, Ravi T Rao, and Michael Shappell. Method for efficient content distribution using a peer-to-peer networking infrastructure, April 1 2014. US Patent 8,688,803.

[19] M. A. Khan and K. Salah. IoT security: Review, blockchain solutions, and open challenges. *FGCS*, 82:395–411, 2018.

[20] Alfonso Panarello, Nachiket Tapas, Giovanni Merlino, Francesco Longo, and Antonio Puliafito. Blockchain and IoT integration: A systematic survey. *Sensors*, 18(8):2575, 2018.

[21] Boohyung Lee and Jong-Hyouk Lee. Blockchain-based secure firmware update for embedded devices in an internet of things environment. *The Journal of Supercomputing*, 73(3):1152–1167, 2017.