Secure IoT Update Using Blockchain

Melike Kaptan
Computer Engineering
Izmir Institute of Technology
Izmir Turkey
melike.kaptan91@gmail.com
ORCID:0000-0003-0110-9035

Emrah Tomur
Research Area Security
Ericsson Research
Istanbul Turkey
emrah.tomur@ericsson.com
ORCID:0000-0001-8985-4974

Tolga Ayav
Computer Engineering
Izmir Institute of Technology
Izmir Turkey
tolgaayav@iyte.edu.tr
ORCID:0000-0002-5339-5507

Yusuf M. Erten
Computer Engineering
Izmir University of Economics
Izmir Turkey
yusuf.erten@ieu.edu.tr
ORCID:0000-0001-9537-7414

2021 2nd International Informatics and Software Engineering Conference (IISEC) | 978-1-6654-0759-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/IISEC54230.2021.9672424

***Abstract*—In this study a platform is devised to send automatic remote updates for embedded devices. In this scenario there are Original Equipment Manufacturers (OEMs), Software suppliers, blockchain nodes, Gateways and embedded devices. OEMs and software suppliers are there to keep their software on Inter Planetary File System (IPFS) and send the meta-data and hashes of their software to the blockchain nodes in order to keep this information distributed and ready to be requested and used. There are also gateways which are the members of the blockchain and the IPFS network. Gateways are responsible for asking for a specific update for specific devices from IPFS database using the meta-data kept on the blockchain, and they will send those hashed secure updates to the devices. In order to provide a traceable data keeping platform, gateway update operations are handled as transactions in a second blockchain network which is the clockchain of the gateways. The system was implemented as of the two separate blockchain networks and it has been shown that, despite the calculation overhead of the member devices, by separating the functions between the two blockchain networks a more reliable and secure platform can be achieved.**

Keywords*—*IoT, remote update, blockchain

I. Introduction

Advances in the connected world requires secure communication between connected entities and robust servers serving those connected devices. This increasing communication introduces security and privacy threats. The existing solutions to update this software rely mostly on centralized servers which introduce a single point of failure. The decentralized nature and the proof-of-work calculations which is an integral part of the blockchains offer a solution to solve these problems for connected world and its applications, because, the increasing number of devices and their needs for connection requires more available servers in decentralized manner. The proof-of-work mechanism is another key element which makes the data corruption harder with the calculation overhead trade off. IoT might benefit from the blockchain networks incorporating smart contracts or keeping fingerprints of their data in blockchains which will beat the data compromise. Another key benefit is having liable history of records when investigations are required for life threatening scenarios.

In this study we propose a software update architecture for the connected devices. The proposed work covers most simple blockchain applications to distribute the updates.

The proposed study has the functionalities listed below:

- We propose to use a blockchain network to distribute software to the devices either to update them or to make installations at the production phase.
- A block chain, a consortium blockchain which is only accessible by authorized users, is used for keeping immutable records of all software producers in an authorized manner. Hence after releasing it to the consortium blockchain network and storing it in the Interplanetary File System (IPFS), producers can not deny the ownership of the software.
- There are also gateway servers which are part of the consortium blockchain, which also form a separate blockchain network themselves. Whenever there is an update for a device that they communicate with, gateway servers will check the software updates in the consortium blockchain to decide if their devices need updates, perform the update operation if necessary and keep a record of these activities.

II. Background

A. Traditional Update

Updates for embedded devices are often done with physical connections on the site or with cloud-based methods. In these techniques software image files are uploaded to the devices through a cable or air interface. In the former approach a technician must execute the work and in the latter remote server performs this update via internet connection. The first method relies on human factor and the second introduces a possible single point of failure. Therefore, both methods have deficiencies. Also in a field like automotive or road side unit software update, if there is a bug in an automotive software and needs to be corrected urgently, relying upon a service technician or one cloud service may have adverse effects on human lives.

B. Blockchain Based Update in IoT

The most important property of blockchain is being secure without requiring any centralized medium to build trust. Blockchain and IoT have a good match here. Data which is needed to be sent from one device to another must have its integrity, confidentiality and availability protected. When we think about data shared between two IoT applications based blockchain environments, it is definitely beneficial for preserving its integrity, since one cannot change data inside the blocks except under the one condition that the intruder has control over most of the nodes of the blockchain network. Integrity also come to effect as the data is signed by the owner and beneficial for availability because data resides on a distributed environment and even if one node is down it is still accessible. Every node in the blockchain network have an ID, address, name and corresponding public and private key pairs. And the network has defined members, and each transaction broadcast is signed by the private key of the sender.

C. Blockchain Based Update in Automotive

Connected vehicle of near future is expected to be carrying a greater number of software applications than ever. These applications are installed onto a special kind of embedded devices called ECU: Electronic Control Unit. Approximately 30-40 ECUs are present in a mid range auto today where this number will be increasing much more as vehicles connected to the Internet will be carrying various applications in the future. When updates are delivered through the Internet, potential security problems will require researchers and manufacturers to apply proper protection mechanisms. One such precaution proposed in the literature against Denial of Service attacks is the use of blockchain infrastructure where the code update sent by the manufacturers is distributed over various nodes of the blockchain.

Vehicle firmware updates are a special subject as far as both the academia and the industry are concerned. Since automotive is a safety critical field, sending remote updates to the vehicles requires careful attention. Most common update methods include physical update done by a service technician as mentioned above. As existing vehicles do not require to be updated very often in their lifetime, this method has served for years. With the advances in the connectivity and increasing importance of the connected vehicles, physical updates are becoming inefficient and producers are seeking new efficient methods.

For over-the-air updates, solutions must be designed to overcome the liability problem. Connections must be traceable in case of any unwanted situations. Whenever a software is distributed to the vehicles, updates which may have bugs must be traceable especially whenever an incident happens like the case of fatal self-driving car accident [1].

III. Related Work

In the literature, there are several studies proposing the use of blockchain technology for IoT and automotive-related tasks. Since the scope of our study is software update mechanisms, we include only such studies in this section.

The starting point of our proposed design is that there may be a universal way to update connected devices to overcome the shortcomings occurring due to lack of confidentiality, integrity and availability as clearly stated in [1]. Their design is one of the first examples which is proposing to use blockchain to send updates to resource constrained devices. In this work the authors also designed an additional mechanism for innocuousness check, assigning some of the nodes to some agencies which are responsible for checking integrity and removing bugs.

In [2] authors presented a secure platform for smart home use-case. In their study, authors proposed has a local blockchain, an immutable ledger in the smart home which is responsible for collecting all the transactions between sensors, actuators or other smart devices. Rather than having a proof-of-work concept, the local blockchain keeps all the transactions handled by the devices. In their following works [3] authors measured the time, packet overhead and energy consumption of the design they previously proposed. In their study they analyse the design and deduce that network and energy consumption of the blockchain based smart home application need to be improved. In order to improve the application, the same authors designed a further study [4] and they changed the way they applied overlay network outside of the smart homes, while they kept the same private blockchain in smart home miners. Rather than having a peer-to- peer and forked blockchain in the overlay network they adopted a public blockchain and a trust mechanism. In [5] another design for the usage of blockchain in smart homes is presented. In this work the transactions serve for the transfer of the firmware update processes of the IoT devices.

In [6] authors use a wireless update methodology and distribute the wireless software updates using blockchain. Their blockchain-based update distribution model is the same model as described in [5]. Software created by the suppliers and manufacturers is propagated through an overlay network in order to be sent to the vehicles. In their study authors compare the results of their mechanism to distribute the new updates and installing them to the vehicles with the wireless updates given in [7]. Their results show that software distribution process requires remarkably less time than the local installation to the vehicles. The same authors in [6] designed another blockchain-based model in [8]. In their study authors draw attention to the liability attribution for an environment with autonomous cars and a connected world.

Like the case of Uber, there is a need to find a liability model for such cases. Authors of [8] emphasize that increased connectivity requires more complex and untampered liability attribution model. In order to realize this model, blockchain based recording mechanism is proposed and improved to provide a decision making mechanism for insurance companies.

Baza et.al. proposes to use a blockhain network to send and get updates in [9]. In their study authors think Autonomous Vehicles (AV) get updates directly from the producer and they also act as gateway nodes which distribute updates for other AVs. Two mechanisms provide secure distribution of the updates, zero knowledge proof and attribute-based encryption.

Our aim here is to propose a model which provides integrity, confidentiality, availability at the same time supports liability for the data transferred. Compared with the surveyed literature

[1]https://www.reuters.com/article/us-uber-crash-autonomous/ uber-not-criminally-liable-in-fatal-2018-arizona-self-/driving- crash-prosecutors-idUSKCN1QM2O8

our model provides a whole picture with a traceable approach in every key entity of the model.

IV. Proposed Model

In this study we introduce a secure and anonymous environment to distribute software to devices. As described before, there will be one blockchain which is used by the software providers of the embedded electronic units and another which is used by the remote software distribution units, namely Gateways, for embedded devices. Both chains will be storing transactions. The roles and the detailed principles of these components will be presented in the proceeding sections but before doing so, we will summarize the overall mechanism.

Any software provider or producer who wishes to send his/her update or first installation to the devices, loads its image file to any platform which returns a hash value which can be used to access the image. With this returned hash, the developer in the company creates a signed transaction with company's private key including this hash value and the additional metadata field and sends it to the blockchain. Additional field indicates target device types that software will be installed. This transaction is broadcast to all miners of the software provider blockchain. Gateways will be informed whenever a new update transaction is broadcast to the producers' blockchain. Then gateway is authorised to read the mined version of the transaction. This means that gateways have rights to read the blockchain data but cannot write. Having read the transactions inside the blocks of the chain, a gateway now has the responsibility to distribute the image file to the corresponding devices. There will be another block chain among gateways. Blockchain of gateways is storing all the information regarding the management of the update transactions. Therefore, there will be signatures of sender gateways and exact timing of the update operation. In the following paragraphs we share some definitions for the sake of completeness.

1) Formal Model: Definitions of some of the terminology is given below:

Original Equipment Manufacturer (OEMs): Original Equipment Manufacturers are the producer companies of the embedded electronic units and their software.

Software Providers(SP): Software Providers are the stakeholder companies providing software for some OEMs.

Devices (D): Embedded devices waiting to be updated.

Gateways (GW): Gateways are the server devices responsible for checking announced updates for devices and sending them to the devices

$$deviceNo_i = < (ProducerNo, Model, Number) >$$

Transaction (Tr): Transactions are the information that are sent to the miners via broadcast messages in order to be put inside blocks. This way the sender and authorized information is shared.

Blocks (B): Blocks are like the storage containers to keep possible number of transaction with the suitable nonce and timestamp.

Mining: The valid block comprises of a suitable nonce, a timestamp, bunch of transactions, authorized information and id. This suitable nonce is formed by a hash calculation using brute force trials. These trials are performed for ensuring the validity of the block.

2) Update File Storage: Before creating a transaction, software image files are kept in a file system. Every file has an encrypted version pointing to the address of those files which will be distributed to the blockchain network to be stacked in a mined block. IPFS is a peer-to-peer distributed file sharing system [10] which is suitable for this purpose because it is secure and easy to share files in a censorship-resistant web like structure. It is like a content based web instead of traditional location based web and its p2p nature makes it fast and secure. It uses an overlay network which has members which do not need to trust each other. IPFS has version control mechanism supported by the Merkle trees. These functionalities make IPFS a favourable versioned-file-system like Git.

3) Transaction Handling: In the producer blockchain network each transaction sender has one public and one private key. Transaction data is sent by the OEMs or Software providers indirectly. Whenever transaction forming data is completed and decided to be released through the provider blockchain, other information is added to the transaction such as operation time and signature of the sender and public key of the sender. Then,

TransactionInformation = $< (SenderID, Imghash, deviceNo_i) >$

turns into a transaction,

Tr = $< SenderID, Imghash, modelNo, timestamp, PublicK, signature, TransactionId >$

By the time the transaction is produced, it is ready to be broadcast to all the producers in the blockchain network for mining operation. For the Gateway Blockchain all the fields will be used for the identification of the Gateways. The sender will be the Gateway and the public key will be the gateways' public key and the hash of the data will refer to the update image of the sent update.

4) Mining Servers: Following the broadcasting of the transactions the members of the producer blockchain put the transaction into a queue. The mining mechanism will be collecting the transactions in this queue and place them into a block. This time block identification fields will be identified. Each block will be keeping hash of its previous block, its ID and number of transactions selected to be placed into that block.

In the blockChain of the gateways, blocks will not be keeping transactions data. Gateways are the servers responsible for updating the devices. They need to be aligned with intended devices all the time when it is needed to send an update, hence gateway mining operations must not be taking resources of the gateway servers. So the block chain of gateways will be keeping only one hash value related to multiple transactions. This hash values, generated by the gateways, are the leaves and the root of the Merkle tree of specified number of transactions waiting in the queue. In [11] Merkle tree operations are summarized.

5) Update: In the update operation, gateways communicate with the devices on site. Today, most of the studies use wi-fi and mobile networks adding extra mechanisms to have secure sender and receiver in order to realize this. An approach where distributors and responders have the roles to verify the operation and repudiate the other side may also be used. But the distribution of the software to the devices or vehicles are out of the scope of our study.

6) Scenario: The sequence diagram of proposed model is given in Figure 1, which shows the general idea of the execution sequence. Very first operation starts with the OEM company who is responsible for producing the software for its devices. The producer wants to keep the newly produced software file in a secure place as well as conveying it to the devices which it is produced for, while keeping its integrity. Then the company puts those in a secure platform like IPFS and forms a transaction which keeps the hash of the file and respective timestamp. The software is now authorized to be used since it is signed with the private key of the producer. The algorithm for this process is shown in Algorithm 1.

Algorithm 1 Algorithm for Sender

```
$INPUT < SenderId, ImageHash, DeviceData >$
**if** $(SenderId) valid$
$(ImageHash) valid$
$(DeviceData) valid$ **then**
    $build\_transaction$
    $TransactionSenderId, ImageHash,$
    $DeviceData, timestamp,$
    $public_key, signature, id$
**else**
    $discardtransaction$
**end if**
```

The transaction goes through a network of computers which are responsible for keeping immutable records of the product features. And this network provides a robust mechanism towards availability attacks since it is a decentralized network of computers, each one is keeping the same chain of information. This chain is formed using Proof Of Work calculations.

After sending updates for devices which need them, gateways must be keeping the records of the updates. This time, these records must be protected under integrity requirement to provide proof of liability. Again, these network of devices, gateways, will be keeping their update transactions in chain of blocks as well as sending the update transactions whenever they performed an update for a device. However, since the number of necessary updates is quite high compared to the software records, it is not logical to keep all transaction information in the blocks but keeping a Merkle tree like structure to preserve integrity is a more viable approach. So a number of transactions will form a Merkle tree now and the hashes of the tree will be kept in the blocks.

V. Implementation

The software is implemented in Eclipse IDE and with Python Socket programming API and run in Python3.6 interpreter. All data is shared in Json format between the members of the blockchain network.

Block Chain Client: Whenever a producer joins the network this producer will have RSA public/private key pairs and respective network address. We already noted that whenever sender sends an update this will be turned into transaction which is signed with the private key of the sender. The transaction will have the respective timestamp and will be signed as depicted below.

S = $TransacitonDetails \times SenderPrivateKey$

S = $< (SenderID, Imghash, modelNO) > \times SenderPrivateKey$

Tr = $< (S, Timestamp, T_{\text{id}}) >$

where T_{id}: Transaction id

Consortium Block Chain:

In algorithm 1 it is shown that every transaction, a sample of which is given in Figure 2, will be broadcast to the network after it is created. After this broadcast operation the transaction will be known by its sender and it is verifiable when added inside a block. Consortium blockchain will have the following operations:

- Transaction Handler : Transaction Handler is responsible for listening all the transactions broadcast to the network.
- Network Broadcaster : Network Broadcaster is responsible for broadcasting the chain length to the block chain network.
- Status Handler : Status Handler is responsible for requesting new blocks mined and chained one after another in any other miner's chain data. Status Handler is managing to gather the blocks which are not in this miner's chain. Then it starts adding the missing ones.
- Chain Synchronization : Chain Synchronization stands for synchronizing any miner's chain with another miner's which has shorter blockchain than this one.
- Network Listener : Network Listener service will be listening to the broadcast messages from other nodes. After listening for the updates from other nodes Chain Synchronization service is called.
- Miner : Miner will be responsible for mining the blocks.

These operations will be running in parallel when a node is up and running. Transaction Handler listens to the transaction port and whenever a transaction is received it receives all data with UDP data packets. Then server puts it inside a queue which will have transactions waiting to be mined. Miner will get one item from the queue and do the necessary checks. After checking the transaction, it will continue to put others in order to reach the necessary transaction count that must be in a block for this blockchain network. After it reaches the defined block size for this network, the miner will be able to start the mining process. Implementation of the services

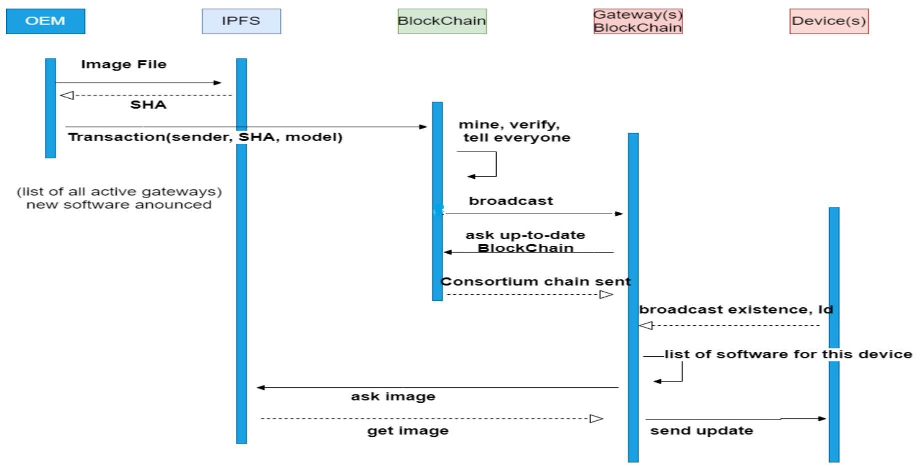
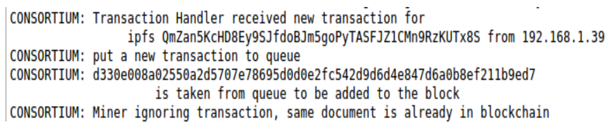

OEM
IPFS
BlockChain
Gateway(s) BlockChain
Device(s)
Image File
SHA
Transaction(sender, SHA, model)
mine, verify, tell everyone
(list of all active gateways) new software anounced
broadcast
ask up-to-date BlockChain
Consortium chain sent
broadcast existence, Id
list of software for this device
ask image
get image
send update

Fig. 1. Sequence Diagram which shows all exchanged messages among different parties in the system



```
CONSORTIUM: Transaction Handler received new transaction for
                ipfs QmZan5KcHD8Ey9SJfdoBJm5goPyTASFJZ1CMn9RzKUTx8S from 192.168.1.39
CONSORTIUM: put a new transaction to queue
CONSORTIUM: d330e008a02550a2d5707e78695d0d0e2fc542d9d6d4e847d6a0b8ef211b9ed7
                    is taken from queue to be added to the block
CONSORTIUM: Miner ignoring transaction, same document is already in blockchain
```

Fig. 2. A sample transaction in the Consortium Network -the transaction is placed in a queue

are adapted from [2] which is a minimum viable blockchain implementation for a wallet of cryptocurrency.

The necessary checks before selecting a particular transaction includes:

- Checking for signature: Since the public key of the sender and the signed version of the transaction is inside the transaction data, server first inspects the signature performing signing operation itself.
- Checking if the transaction already exists: Since the same transaction must not exist more than once, a block miner will check the block that is prepared to be mined at that time. Miner will also check its existence in the block chain.
- Checking the existence of the same update document for the same kind of devices: Miner will be checking if the same upload for the same device family already exists in both the block to be mined and the whole chain of records.

Mining operation will continue as long as there is not a valid chain listened by Status Handler and asked from the Chain Synchronization service. If a node in the network have a chain length which is greater than the length of this chain of blocks then the node will request the longer chain from the respective node. It will be asking for that block it does not own. In order to have a valid chain, each block will be keeping the hash of the block before the block itself. Hence requested block must be in the correct order before they are added to the requester's chain.

[2]https://github.com/codebox/blockchain

Gateway Network:

Similar to the Consortium Network, Gateway Network is also a consortium network. Gateway Network will have both similar and different kind of operations with the Consortium network. The operations which are different are listed below.

- Gateway Listener : Gateway Listener will be listening the consortium block chain status information. Gateway Listener will be the trigger for the chain synchronization operation with the block chain network of the producers. Receiving a new update from the producers will be the prime cause of the update operation.
- Device Listener : Device Listener is responsible for listening the status information coming from the devices. Devices will be sending their device information given by their producers and the software version they carry at that time.
- Miner : Miner will be responsible for mining the blocks.

Gateway blockchain network will behave in different ways than the producer consortium network. Gateway will be listening to the broadcast status information from consortium network. Since Gateway network will be reading the consortium chain, nodes will be updating their records of chain length according to the consortium network. This data is used for just identifying that there are updates for some devices and reading those updates.

The idea is to remain updated all the time on the updates in the consortium network. The process is given in Algorithm 2 for Gateway Listener. The actual gateway operations start listening to the device information on site and checking for the new updates for related devices from all the immutable ledger of updates. Devices will be identified by their device_no parameter. This number will cover respective brand and model or any other necessary information as well as the number specific to that device. Whenever a gateway communicates with a device it will list all updates for this device type, regardless of the exact device number. This is for inspecting the last update for this device type.

Algorithm 2 Gateway Listener

```
Require: Listen Consortium network broadcast information.
  if nodes from consortium chain have block chain lenght longer than we read before then
    $ask\ for\ the\ newly\ mined\ blocks$
  end if
```

After the gateway sends an update to the device it will send a transaction for this update operation in order to be mined as shown in Algorithm 3. But this time, gateway mining will be different than the consortium network.

Algorithm 3 Gateway Operation

```
Require: have consortum chain readible
Require: have status info messages from devices m(¡device_ no, software_ version¿)
  $list\ all\ updates\ historically\ for\ this\ device\ type$
  if device software version is not the has update for this device type then
    $Connect\ ipfs\ to\ get\ last\ update\ file$
    $for\ this\ device$
    $Send\ last\ update\ to\ this\ device$
    $Send\_Transaction(sender\_gateway\_address,$
    $hash\_of\_the\_update\_file, complete\_device\_id)$
  end if
```

Gateway mining pre-checks and mining operation will be as follows:Before adding a respective transaction to the unmined block, miner will be assured that the current unmined block does not have the same update file for the same device and the gateway blockchain also does not have the same update file for the same device. It performs the necessary checks like in the consortium network.

- Checking for signature: Server first inspect the signature performing signing operation by itself. But this time the sender will be a gateway and the transaction to be signed by the private key of that gateway.
- Checking if the transaction exists : Miner will check its existence in the block chain and the current block.
- Checking if the same update document for the same device exists: Miner will be checking if the same upload for the same device already exists in both the block to be mined and the whole chain of records. Also, we do not want the same device to be updated with the same version of software more than once. But our study does not cover this condition since there may be more than one reason for such a case.

VI. Conclusion

The main aim of this study is to propose a model to send remote software updates for embedded devices which we are becoming more and more dependent everyday. If there is not a suitable update mechanism to send updates to the devices, with the enhanced functionalities of these machines, it will create serious problems for the users and manufacturers. That is, if there is a future with connected and more capable vehicles, taking the vehicles to the service shops in order to load new software will not be practical. Or with the growing production areas, installing software by hand will not be efficient enough in the production process.

Beside this, in the future connected world, inspections for liability will be more complex because there will be various producers, large number of users and interactions. Case that need to be verified must be leaning on trust based interaction records. Here integrity and authentication are necessary in a trustless environment. Another concern is availability. In order to serve great number of users, servers must be available all the time. Single point of failures show us that there may be life threatening situations.

What we proposed and designed in this study is a sample model which is suitable for the future use in terms of the requirements we mentioned above. It is a system which involves to blockchain networks which cooperate and perform an update process which helps reduce security issues such as availability and integrity. The proposed solution has been implemented and a feasibility study has been performed. This prototype has to be improved before being applied in a professional production environment since it does not cover the confidentiality area especially in the communication from the gateways to the devices.

References

[1] Boudguiga, A., N. Bouzerna, L. Granboulan, A. Olivereau, F. Quesnel, A. Roger, and R. Sirdey (2017). Towards better availability and accountability for iot updates by means of a blockchain. In 2017 IEEE European Symposium on Security and Privacy Workshops (EuroS& PW), pp. 50–58. IEEE.

[2] Dorri, A., S. S. Kanhere, and R. Jurdak (2016). Blockchain in internet of things: challenges and solutions. arXiv preprint arXiv:1608.05187.

[3] Dorri, A., S. S. Kanhere, and R. Jurdak (2017). Towards an optimized blockchain for iot. In Proceedings of the Second International Conference on Internet-of-Things Design and Implementation, pp. 173–178. ACM.

[4] Dorri, A., S. S. Kanhere, R. Jurdak, and P. Gauravaram (2017). Blockchain for iot security and privacy: The case study of a smart home. In 2017 IEEE international conference on pervasive computing and communications workshops (PerCom workshops), pp. 618–623. IEEE.

[5] Lee, B. and J.-H. Lee (2017). Blockchain-based secure firmware update for embedded devices in an internet of things environment. The Journal of Supercomputing 73(3), 1152–1167.

[6] Steger, M., A. Dorri, S. S. Kanhere, K. Römer, R. Jurdak, and M. Karner (2018). Secure wireless automotive software updates using blockchains: A proof of concept. In Advanced Microsystems for Automotive Applications 2017, pp. 137–149. Springer.

[7] Steger, M., C. Boano, M. Karner, J. Hillebrand, W. Rom, and K. Römer (2016). Secup: secure and efficient wireless software updates for vehicles. In 2016 Euromicro Conference on Digital System Design (DSD), pp. 628–636. IEEE.

[8] Oham, C., S. S. Kanhere, R. Jurdak, and S. Jha (2018). A blockchain based liability attribution framework for autonomous vehicles. arXiv preprint arXiv:1802.05050.

[9] Baza, M., M. Nabil, N. Lasla, K. Fidan, M. Mahmoud, and M. Abdallah (2018),Blockchain-based firmware update scheme tailored for autonomous vehicles. arXiv preprint arXiv:1811.05905.

[10] Benet, J. (2014). Ipfs-content addressed, versioned, p2p file system. arXiv preprint arXiv:1407.3561.

[11] Nakamoto, S. et al. (2008). Bitcoin: A peer-to-peer electronic cash system.