Secure Code Distribution based on BlockChain

Alexandre Ortigosa, *Student, UPC,* Miquel Ferriol, *Student, UPC,* and Hamid Latif, *Student, UPC*

Index Terms—Computer Society, Blockchain, Software distribution, secure, LaTeX, paper.

1 Introduction

NOWADAYS the Internet of Things (IoT) [1] paradigm promises that we will be surrounded by computers that will help in our daily needs, boost industrial efficiency and monitor and actuate over critical infraestructures. In short, IoT represents the bridge between the physical and the virtual worlds.

IoT deployments operate with a set of nodes that cooperate to achieve the desired functionality, typically such nodes are tiny computers equipped with communication capabilities (e.g., wireless) and their own operating system and applicatons.

In this context, it is expected that IoT deployments are a target of cyber-attack, since successful attacks can disrupt physical infraestructure and provide important benefits to the attackers, either economically or politically. As such, protecting IoT deployments and the code running in its nodes is an important challenge.

In this context, secure code distribution of IoT deployments is a well-established topic that typically is achieved by means of Public Key Infraestructure, for instance [7], [8] and [9] are notable examples.

In this paper we propose a software architecture to secure code distribution in IoT deployments taking advantage of the recent Distributed Ledger Technology [2] to provide secure distribution. In addition we prototype the proposed solution and provide a set of performance analysis to evaluate its feasibity.

2 Background

2.1 Blockchain

BlockChain [3] is a of the technologies candiate to revolutionize our economy. It is a DLT (Distributed Ledger Technology) technology that, as its name suggests it is a chain of consecutive blocks of data interlaced between them, as a distributed data base in continuous growth that is controlled by all its participants.

Each one of its blocks is preceded by another block, except the GENESIS block (the first block of the blockchain, generated in its inception). Blocks are associated by means of hash identifiers where each block has the hash of its previous block in its header, as shown in Fig. 1.

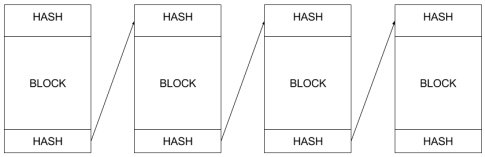

Fig. 1: Chain of blocks.

Each block stores the different transactions and data that represents the state of the chain. All the participants of blockchain can create new blocks. In order to add a new block to the chain a computationally costly problem must be solved, this is done by the blockchain miners. In addition to this, new blocks must be validated by a subset of the participants, for this different mechanisms exist. Finally, once the new block has been validated, it is transmitted to all other nodes using a peer-to-peer network.

This way all nodes have an updated copy of all the chain. As a result, this technology provides inmutability, since having all nodes a copy it is impossible to modify a block without every node noticing it. In addition, it is an inherently decentralized technology, since it uses peer-to-peer networks to distribute information.

2.1.1 Smart Contracts

Smart contracts represents blockhain's next evolution. They are pieces of code created by the users and stored in the blockchain. These smart contracts are defined by certain rules set by the user. These rules are what govern the behavior of all the transactions and data related with the smart contract. To define these contracts a specific programming language, very similar to javascript and called solidity, is used.

Smart contracts are executed by blockchain nodes and its correct execution is guaranteed by a consensus protocol. Each smart contract is identified by a hash of 160 bits, similarly to blockchain's wallets.

The flexiblity enabled by smart contracts enable to implement a wide variety of functions, defining any logic or data wharehousing. Smart contracts also provide security by means of the consensus protocol. However, human mistakes may appear during the definition of the rules, and some of them may imply major security failures. A notable example of this problem is the security breach that took place in the

- *UPC BARCELONA TECH*

  E-mail: alexandre.ortigosa@gmail.com see http://www.fib.upc.edu
- *Alexandre Ortigosa, Miquel Ferriol and Hamid Latif están afiliados a la Facultad de Informatica de Barcelona, Universitat Politecnica de Catalunya, Barcelona, Spain*

Manuscript received Decemver 20, 2016.

DAO, a very popular smart contract from which someone managed to steal $50.000.000 due to a bug in its definition. [13]

The execution of smart contracts is done in the blockchain itself when the miners collect transactions sent to the identifier of the contract and process them to add then to blockchain. Once appended to blockchain, the contract is accessible by all nodes. Another remarkable feature of smart contracts is that they are inmutable and persistent. Inmutable because once they a contract is defined, it cannot be modified and the persistent protperty is inherited from the blockchain technology.

One of the most well-known and well financed platforms that implements smart contracts is Ethereum [14], this is the platform used in this paper to prototype our secure code distribution protocol.

2.1.2 IPFS [5]

IPFS (InterPlanetary File System) is a distributed, peer-to-peer filesystem. It can be understood -according to its creator [5]- as a single BitTorrent swarm.

IPFS works with *merkle links*, in which an address is obtained using the hash of the root directory contents instead of an IP address, as would happen typically in HTTP. This hash allows to verify that the hash of the contents is the same as the hash used to access it.

We selected IPFS because of its characteristics, which guarantee integrity and authenticity, among other things.

3 State of the Art

In this section we describe the state-of-the-art in secure code distribution, as we will see, solutions are typically based on PKI [6] for consensus and verification.

3.1 Secure Code Distribution for Wireless Sensors

The first paper,Efficient Code Distribution in Wireless Sensor Networks [7], proposes a methodology of secure code distribution in the wireless sensor scenario. It emphasizes in the processes of update and does not clear out the channel of distirbution, reason why we could say that it uses a standard channel sending direct messages through the net.

3.2 Secure Code Distribution in Dynamically Programmable Wireless Sensor Networks

In Secure Code Distribution in Dynamically Programmable Wireless Sensor Networks [8], the authors present a methodology to secure code distribution in wireless sensors is proposed. Again, a methodology of division of the code to be transmitted is implemented and these divisions are linked in a data structure of chain type, where each block of data informs of the next block through a hash. This way nodes can verify that all the sequence of received packets is the correct one.

In this article, again, the channel of communication stands aside.

3.3 Secure Network Coding [9]

This article defines a new communications protocol oriented to code distirbution, applying methodologies of transformation to encrypt data and limit the types of codes transmitted through the net. Again, all the process is based on PKI methods.

4 Secure Code Distribution

In this section we will describe how to achieve a secure distribute code distribution.

In our approach we take advantage of IPFS and Ethereum, our focus is to provide reliability and guarantee that updates can be authenticated and are inmutable.

4.1 Overview

Fundamentally, our proposed approach uses IPFS as a data repository of updates and Smart Contracts (Ethereum) to index such updates.

IPFS stores the 'diffs' of the code versions (code updates), such updates are identified by a hash, but they do not provide a required chronological order. For this, we use Smart Contracts to provde a secure index that orders them chronologically.

As mentioned before, IPFS allows such updates to be inmnutable and to verify their integrity. On other side, Ethereum provide the right index.

4.2 Architecture

In the proposal there will be repository nodes, which will be in charge of generating the updates, update them to IPFS and create a new entry in the index of the Smart Contract, and normal nodes, which will be in charge of checking the index and download the new version.

The address of the Smart Contract must be known by all nodes. These will be achieved by creating the contract before deploying the normal nodes and "hardcoding" its address in them.

The Smart Contract can guarantee that the repository nodes have privileges and that these decide if a new normal node can join or not the list of nodes that can check the hashes of the new updates. This can be interesting in some cases, but can raise some security issues, as it can result in an attacker gaining some privileges over the Smart Contract if he breaches a repository node. Ethereum allows the decitions of a Smart Contract to be voted, hence it can help to avoid this case, but it can add complexity and still has a security issue if the attacker gains privileges over it if he breaches more half of the repository nodes.

In this paper we have decided to give privileges to the repository nodes. Ethereum allows to send notifications to the users subscribed to a function of a Smart Contract when it is executed. This is called Event [10]. Through these events we can achieve, efficiently, that all normal nodes know when an update is available in order to download it.

4.3 Step-by-step process

The next figure describes in detail the steps followed to update the code:

0) The repository node creates the diff between the current version and the one to be uploaded.
1) The repository node uploades the update to IPFS
2) IPFS gives a hash identifier of the uploaded file
3) The repository node executes the smart contract functions required to indicate that a new version is available and to add the hash to the list and
4) The smart contract sends the event to subscribed nodes indicating that there is a new version
5) Normal nodes ask IPFS for the diff that the smart contract sent them
6) IPFS gives back the file with the diff
7) Normal nodes apply the diff over the firmware version they had...

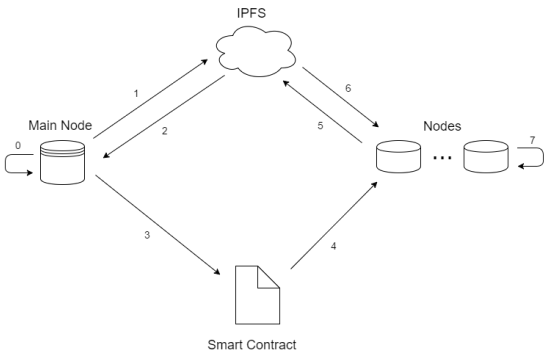

Fig. 2: Secuencia

The process will be sepatated in two parts: updating a new version and downloading a new version.

4.3.1 *Upload a new version*

The process begins with the upload of a new version to IPFS. To make this, a script named "GenerateCommand" is used, which generates the diff version of the patch using the previos and later. To achieve it, the computer from which the script is executed must be running IPFS.

After this, it will log in to Ethereum, unlocking the account and initializing the Smart Contract. After this, the function "uploadNewVersion" of the Smart Contract is executed, which adds the string of the hash of the new version to the list of strings of the Smart Contract. This action generates an event, that causes that normal nodes know that a new version is available.

4.3.2 *Receive a new version*

In order to be able to receive new versions, first the node has to synchronize with the blockchain of Ethereum. To make it, the script "buscarActualizaciones" has to be executed. This script will launch the script "gethBuscar", which will log in to Ethereum, with which will begin the process of synchronization. After this, it will unlock the node's Ethereum account and will initialize the Smart Contract. From this moment, the script will wait to detect a new event that indicates that a new version is available. Once the event reaches the node, it will download the update through IPFS and the hash that it will obtain via the Smart Contract. Once it is downloaded, it will be applied.

5 Performance analysis

5.1 Test Bed

To study the performance of the proposal the time that is needed to upload a version to IPFS, to download it from IPFS and to apply it to the node has been analyzed . In the Fig. 3 the time needed to do those actions can be seen, depending on the size of the update.

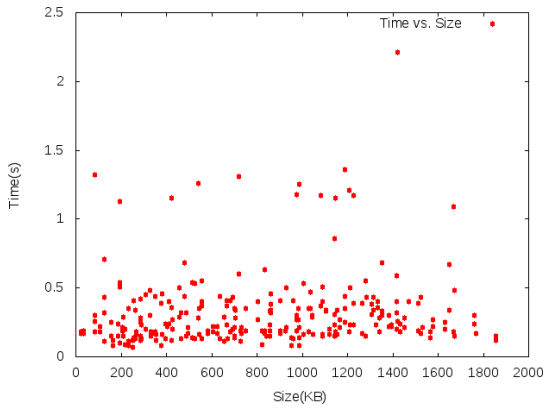

Fig. 3: Size vs time

5.2 Results

Generally, the time needed to do the process does not vary very much for updates whose size is relatively small (between 1 and 2000 KB). To compute the size of the diffs a public repository of git [11] and different versions of it have been used, using the Linux's function diff -ruN.

Next, the performance of events in Ethereum will be studied.

As it has been explained, to start the process of the update an event is required to indicate that there is a new version available, reason why it will be interesing to study the time needed to detect a new event since it the function of the Smart Contract has been executed.

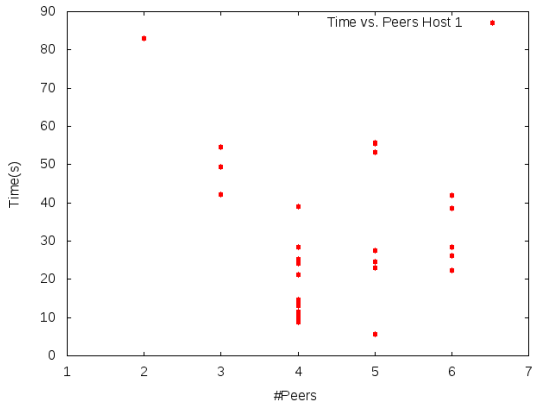

Fig. 4: Time vs peers Host 1

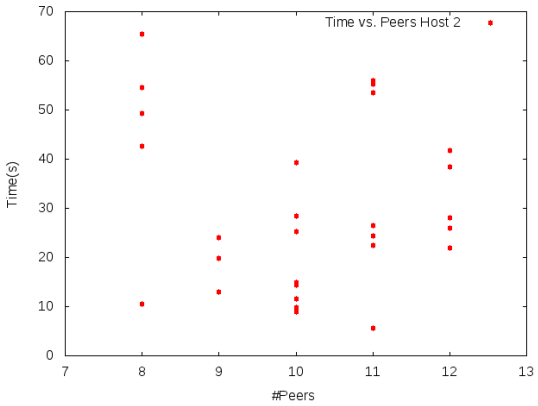

Fig. 5: Time vs peers Host 2

To do this, two hosts synchronized to the blockchain of Etherum haven been used, with a filter established to receive the new events from the Smart Contract.

Figures Fig. 4 and Fig. 5 show the time that passes between the execution of the function and the reception of the event by the nodes, depending on the numbers of peers that each of those have.

To make it, the repository node has sent a notification and the time elapsed between that and the reception of the event has been measured.

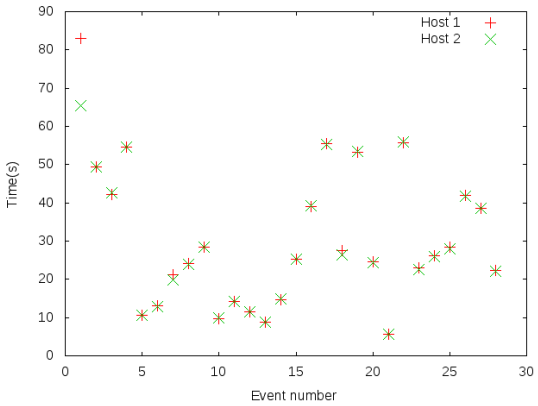

Fig. 6: Comparison between host1 and host2

As it can be seen on Fig. 6, the times needed to same update are very similar, with differences of tenth of seconds. This makes us think that the speed of the update does not depend on the number of peers but the time in which the net is able of mine the transaction and add it to the chain.

6 Conclusions

It has been seen as IPFS and the smartContracts work very well together to safely distribute the code in addition to doing it with a more than reasonable time and in a very synchronized way. This solution is applicable to machines with a high storage capacity since, as being a node in the blockchain, implies having the blockchain itself downloaded. If this solution were to be applied to devices with less storage, a possible solution would be the use of proxies that were the intermediary between the node itself and the blockchain.

References

[1] Xia, F., Yang, L. T., Wang, L., & Vinel, A. (2012). Internet of Things. International Journal of Communication Systems, 25(9), 1101-1102. doi:10.1002/dac.2417
[2] Mclean, S., & Deane-Johns, S. (2016). Demystifying Blockchain and Distributed Ledger Technology Hype or Hero? Computer Law Review International, 17(4). doi:10.9785/cri-2016-0402
[3] Swan, M. (2015). Blockchain: blueprint for a new economy.
[4] Swan, M. (2015). Summary for Policymakers. Climate Change 2013 - The Physical Science Basis. http://doi.org/10.1017/CBO9781107415324.004
[5] IPFS - Content Addressed, Versioned, P2P File System (DRAFT 3)
[6] Tuecke, S., Welch, V., Engert, D., Pearlman, L., & Thompson, M. (2004). Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile. doi:10.17487/rfc3820
[7] Reijers, N., & Langendoen, K. (2003). Efficient code distribution in wireless sensor networks. Proceedings of the 2nd ACM International Conference on Wireless Sensor Networks and Applications (WSNA 03), 6067. http://doi.org/http://doi.acm.org/10.1145/941350.941359
[8] Deng, J., Han, R., & Mishra, S. (2006). *Secure code distribution in dynamically programmable wireless sensor networks*. on Information Processing in Sensor Networks, (December), 292300. http://doi.org/10.1109/IPSN.2006.243786
[9] Ning Cai and Raymond W. Yeung.(2001). Secure Networking Coding.
[10] https://media.consensys.net/technical-introduction-to-events-and-logs-in-ethereum-a074d65dd61e
[11] https://github.com/twitter/twitter-kit-android
[12] Para crear estos diff se ha usado un ordenador con el siguiente procesador: Intel i7-3632QM @ 2,2GHz To create the diffs
[13] https://www.wired.com/2016/06/50-million-hack-just-showed-dao-human/
[14] https://www2.deloitte.com/us/en/pages/finance/articles/cfo-insights-getting-smart-contracts.html

Appendix A
Código desarrollado

https://github.com/alexortigosa/securedistsoft/tree/master/PAE

Acknowledgments

The authors would like to thank Atraura for introducing us to Smart Contracts and Albert Cabellos Aparicio, a UPC teacher whose help made this possible.