accepted to IEEE Workshop on Security & Privacy on the Blockchain, 2018

Incentivized Delivery Network of IoT Software Updates Based on Trustless Proof-of-Distribution

Oded Leiba, Yechiav Yitzchak, Ron Bitton, Asaf Nadler, Asaf Shabtai
Department of Software and Information Systems Engineering
Ben-Gurion University of the Negev
Beer-Sheva, 8410501, Israel
{*odedlei,yitzchak,ronbit,asafnadl*}*@post.bgu.ac.il, shabtaia@bgu.ac.il*

***Abstract*—The Internet of Things (IoT) network of connected devices currently contains more than 11 billion devices and is estimated to double in size within the next four years. The prevalence of these devices makes them an ideal target for attackers. To reduce the risk of attacks vendors routinely deliver security updates (patches) for their devices. The delivery of security updates becomes challenging due to the issue of scalability as the number of devices may grow much quicker than vendors' distribution systems. Previous studies have suggested a permissionless and decentralized blockchain-based network in which nodes can host and deliver security updates, thus the addition of new nodes scales out the network. However, these studies do not provide an incentive for nodes to join the network, making it unlikely for nodes to freely contribute their hosting space, bandwidth, and computation resources.**

In this paper, we propose a novel decentralized IoT software update delivery network in which participating nodes (referred to as *distributors*) are compensated by vendors with digital currency for delivering updates to devices. Upon the release of a new security update, a vendor will make a commitment to provide digital currency to distributors that deliver the update; the commitment will be made with the use of smart contracts, and hence will be public, binding, and irreversible. The smart contract promises compensation to any distributor that provides *proof-of-distribution*, which is unforgeable proof that a single update was delivered to a single device. A distributor acquires the proof-of-distribution by exchanging a security update for a device signature using the Zero-Knowledge Contingent Payment (ZKCP) trustless data exchange protocol. Eliminating the need for trust between the security update distributor and the security consumer (IoT device) by providing fair compensation, can significantly increase the number of distributors, thus facilitating rapid scale out.

1. Introduction

The number of IoT devices is continuously increasing. According to Gartner Inc.[1] 11 billion connected "things" will be in use in 2018, and this figure will reach up to 20 billion by the end of 2020. The massive growth in the number of connected IoT devices, and particularly their essential need for security software and/or firmware upgrading (patching), has resulted in the search for reliable and trusted solutions for distributing software updates on a large scale.

Traditional software update services mainly rely on a client-server architecture, where bandwidth consumption and maintenance issues impose high costs and put security and availability at risk. IoT vendors that wish to provide software update services for their products are required to maintain large-scale data centers to support software distribution to millions of devices[2]. The wide range of IoT devices (and firmware) also poses a challenge to the software update services commonly used for managing updates within organizations. Consequently, vendors delegate the task of updating software to the end users who are not always aware of the security issues and the importance of keeping the devices updated.

For these reasons many IoT devices are not consistently being updated and remain vulnerable to known threats [1], [2], [3]. Previous works have targeted the availability and scalability challenges by using a permissionless blockchain-based decentralized network instead of a private centralized vendor network [4], [5]. However, these solutions provide no incentive for non-vendor nodes to host and deliver security updates, and may face limitations similar to those faced by centralized vendor networks.

In this paper we propose a novel *decentralized* and *incentivized* IoT update delivery network based on trustless *proof-of-distribution*. The proposed framework utilizes blockchain, smart contracts, zero-knowledge contingent payment (ZKCP), and a decentralized storage network. Within this framework participating nodes (i.e., distributors) are compensated for delivering software updates to IoT devices. More specifically, when a new security update is released, a vendor will agree to compensate (with digital currency) distributors who deliver the update. The agreement is bound by a smart contract, making it public and irreversible, which guarantees compensation to any distributor that provides

1. https://www.gartner.com/newsroom/id/3598917

2. http://www.channelpartnersonline.com/2017/12/05/aws-dell-emc-among-vendors-in-rapidly-growing-iot-services-market/

proof-of-distribution, i.e., unforgeable proof that a single update has been delivered to a single IoT device. A distributor acquires the proof-of-distribution by exchanging a security update for a device signature. To ensure trustless data exchange we use the Zero-Knowledge Contingent Payment protocol. Eliminating the need for trust between the software update distributor and the consumer (IoT device) by providing fair compensation, can encourage competition, significantly increase the number of distributors and overall efficiency, thus allowing rapid scale out.

The main contributions of the proposed framework are:
Availability. The proposed framework ensures high network availability compared to the current client-server architecture.
Programmable incentivization. The framework makes it straightforward to support different prioritizations through minor adjustments to the smart contract. Vendors can decide to push forward specific critical security updates, prioritize specific IoT clients, or other relevant settings.
Auditability. The proposed framework allows vendors to monitor and track the software download of its devices.
Integrity. The proposed framework provides high integrity of the software.

The rest of the paper is organized as follows. Section 2 provides a brief introduction to the building blocks of our proposed ecosystem: blockchain, smart contract, and zero-knowledge proofs. Section 3 summarizes previous works in this domain, highlighting the manner in which the proposed ecosystem addresses the limitations of prior work. The proposed ecosystem is described in Section 4. In Section 5 we briefly discuss potential attacks and their handling, and provide a formal claim and proof of the fair exchange. In Section 6 we discuss the limitations of the proposed framework and finally, in Section 7 we conclude with the paper's contributions and introduce possible directions for future work.

2. Background

In this section we introduce the technologies utilized in the proposed framework, providing the necessary background to understand the protocol and its analysis.

2.1. Blockchain

Blockchain was introduced alongside Bitcoin in Satoshi Nakamoto's white paper [6]. A blockchain is a data structure that can be regarded as a public ledger where groups of messages are stacked one on top of the other. These groups of messages are named "blocks" and with the use of digital signatures and a distributed consensus algorithm, users in a non-synchronous configuration (i.e., do not necessarily agree on time and order of messages) can irreversibly agree with high probability on a specific order of blocks.

The agreement on the order of blocks is useful for the prevention of currency "double-spending" (where messages within the blocks correspond to money transactions) as the latter spending can be mutually eliminated by the users, and is therefore the foundation for Bitcoin and various other cryptocurrencies [7], [8], [9].

Concretely, each transaction message within a block contains the coin transfer value, the redeem terms, an input transaction and the signature of the transaction's author for authenticity. The redeeming terms can be referred to as a boolean predicate such that only a true evaluation can make use of the transaction. Because published transactions are a part of a block, they are irreversible and thus redeeming terms can serve to vouch for coin in exchange for a truth assignment, which is the foundation of "smart contracts".

2.2. Smart Contracts

A smart contract [10] is a protocol that enforces the negotiation or performance of an agreement. In the context of blockchain-based cryptocurrencies, it is realized by a traceable and irreversible transaction that can be redeemed only if a set of terms are met. At their essence, smart contracts don't differ from any other kind of transaction and are primarily named likewise when used with redeeming terms that are more complex than verifying the receiver's identity. However, the contextual notion of being addressed to anyone who qualifies to specific terms instead of a specific address makes smart contracts valuable to nodes who wish to satisfy these terms and collect the value.

The features of blockchain-based smart contracts evolved from the Bitcoin scripting language that is a simple, stack-based, and purposefully non-Turning-complete, to a new blockchain paradigm initiated with Ethereum [7] which offers Turing-complete stateful languages for writing smart contracts. The most widely adopted example of such language used for developing smart contracts in Ethereum is Solidity which is a JavaScript-like language. Rootstock [3] is another example of a blockchain platform offering equivalent smart contract capabilities. Hereafter, the use of the term smart contracts in this paper refers to the Ethereum implementation.

2.3. zk-SNARKs

Zero-Knowledge Succinct Non-Interactive ARguments of Knowledge (zk-SNARKs) [11], [12], [13], [14] is an efficient and secure system for proving and verifying zero-knowledge proofs. It allows a *prover* to (efficiently) convince a *verifier* that she possesses knowledge of a secret parameter, called a *witness*, satisfying certain properties, without revealing anything about the secret to the verifier or anyone else. For example, a prover can convince a verifier that it has knowledge of a secret w which is the hash preimage of some value x, without revealing anything about w. Because zk-SNARKs are *succinct*, proofs are very short and easy to verify. More formally, let L be an NP language and C be a decision circuit for L. A trusted party conducts a one-time setup phase that results in two public keys: a

3. http://www.the-blockchain.com/docs/Rootstock-WhitePaper-Overview.pdf

proving key pk and a verification key vk. The proving key pk allows any (untrusted) prover to generate a proof π attesting that $x \in L$ for an instance x of her choice. The non-interactive proof π is both *zero-knowledge* and *proof-of-knowledge*. The proof π has a constant size and can be verified in time that is linear in $|x|$.

A zk-SNARK for circuit satisfiability consists of the following three polynomial-time algorithms:

- $Gen(1^\lambda, C) \rightarrow (pk, vk)$. On security parameter λ and a decision circuit C, *Gen* probabilistically samples pk and vk. Both keys are published as public parameters and can be used to prove/verify membership in L_c.
- $Prove(pk, x, w) \rightarrow \pi$. On input proving key pk, instance x and witness for the NP-statement w, the *prover Prove* outputs a non-interactive proof π for the statement $x \in L_c$.
- $Verify(vk, x, \pi) \rightarrow \{0, 1\}$. On input verifying key vk, an instance x, and a proof π, the *verifier Verify* outputs 1 if $x \in L_c$.

2.4. Zero-Knowledge Contingent Payments (ZKCP)

Zero-Knowledge Contingent Payment (ZKCP) [15] is a two-party protocol allowing the fair exchange of information for payment (e.g., cryptocurrency coins) in a setting where the two parties do not trust each other to deliver. Besides being private and secure in theory, it was practically used in Bitcoin with the use of zk-SNARKs [4].

3. Related Works

Our work falls within the following domains: IoT devices' patching and decentralized storage networks (also known as peer-to-peer file sharing networks).

the two categories.

3.1. General Architectures for IoT Patching

Prior to the Internet of Things (IoT) era, traditional host-centric IT solutions focused on delivering security updates (i.e., patches) by self-hosting the binary updates to make them retrievable and available to clients. The availability and reliability of such solutions may require a complex server infrastructure and experienced IT personnel as the number of clients grow, thus making it very expensive for software providers. Several works targeted the availability and reliability challenges, including Liu et al. [16] who suggested a IaaS solution to outsource infrastructure maintenance, and Zhen-hai and Yong-zhi [17] who suggested a Maven-based solution to increase effectiveness in case of updates' independence. These solutions were not designed towards a large number of deployed devices as in an IoT network.

4. https://bitcoincore.org/en/2016/02/26/zero-knowledge-contingent-payments-announcement/

Following the rapid growth of IoT networks, the work of Yu et al. [18] draws the challenges of securing IoT devices with traditional security, host-centric IT solutions such as anti-virus and software patches. Mainly, the diversity, cyber-physical coupling and scale of IoT devices, forces these security IT solutions to a paradigm shift. Several works focused on the diversity by suggesting solutions for specific scenarios [19], and the updating in a not-trustworthy setting whereas a device may already be infected [20], [21]. However, the abovementioned works do not address the scalability challenge.

Lee and Lee [4] proposed a firmware update scheme in an IoT environment based on custom blockchain with a single-purpose block structure, combined with a peer-to-peer file sharing network as BitTorrent for the distribution of updates, to enhance availability, integrity and versions traceability of updates. This suggestion was improved by Boudguiga et al. [5] with the addition of trusted innocuousness checking nodes that are in charge of verifying the patch before it becomes available to deployed IoT devices.
Notably, both of the latter solutions acknowledged the naiveness in uploading an entire file to the blockchain, hence delegated the distribution effort to off-chain approaches. Nonetheless, they provide no incentive for non-vendor nodes to join the network, and may thus struggle with similar limitations as a centralized vendor network.

Another approach worth mentioning is IOTA [22], a distributed micro-transactions ledger for IoT devices. IOTA is designed for the exchange of services among IoT devices (e.g., electricity for cooling), as opposed to IoT devices patching. Moreover, IOTA's incentive mechanism is based on the need of a device to issue a service request and is therefore unfit for patch delivery.

In our proposed solution we extend the use of a blockchain-based network for IoT update delivery, and specifically address the issue of an incentive for providing the updates (by non-vendor nodes). In addition, the proposed solution eliminates the need for trust between the security update distributor and the consumer (IoT device), thus allowing rapid scale out.

3.2. Decentralized Storage Networks

Peer-to-peer (P2P) file sharing is a technology for the sharing of digital media. Imbalanced use (e.g., peers that avoid uploading) can dramatically reduce the scale and availability of the network and therefore, sharing incentivization is necessary. The first examples of P2P file sharing incentivization mechanisms are the BitTorrent's choking algorithm [23] and Gnutella's free riding prevention [24], in which the download rate for a user that downloads content yet avoids uploading content, is penalized. Though these mechanisms have been proven useful for symmetric users that wish to both download and upload content [25], they are not designed for an asymmetric sharing scheme.

The asymmetric file sharing scheme is composed of two types of end-users: distributors that host files and consumers

that wish to acquire them. Proper incentivization of distributors within this scheme can be more challenging than in the symmetric case. However, with the emergence of the blockchain technology that allows data and currency exchange in the absence of trust among its users, new forms of incentivization are unveiled. The most common incentive is digital currency compensation for either providing storage or bandwidth, which is the case for blockchain-based decentralized storage solutions such as Swarm [5], Filecoin [6], Storj [26], and Siacoin [27].

Regardless of these general incentivization schemes, software patches distribution is a special setting that remains unaddressed, in which there should be no more than a single compensation per patch delivered to a device. This setting, which is the essence of an IoT software updates distribution system, is the main novelty of this P2P file sharing system.

4. Proposed Framework

The proposed framework allows vendors (i.e., manufacturers of IoT devices) a secure and scalable delivery of security updates to their deployed IoT devices. It is referred to as a framework because of its general requirements which allow a variety of concrete implementations. The framework is comprised of two networks. The first is a decentralized storage network (DSN) (see Section 4.1), in which security updates are transferred between network nodes, and the second is a blockchain network (see Section 4.2), in which vendors commit to payment in exchange for delivery in the form of smart contracts, thus incentivizing delivery within the DSN. Both of these networks are facilitated by the three types of nodes participating in the IoT updates delivery network (see Section 4.3): vendors, IoT devices and distributors. These nodes participate in both networks and interact with each other to support the paid-for delivery of updates.

The delivery process (portrayed in Figure 1) is triggered when a vendor releases a new security update that should be delivered to its deployed devices. The vendor delegates the task of serving security updates to its deployed IoT devices, by publishing a smart contract in the blockchain network in which he commits to providing coins in exchange for a *proof-of-distribution*, i.e., providing a proof of delivering the update to a deployed device (see Section 4.4.1). Also, the vendor hosts the new security update, enabling other nodes to consume via the DSN, for a limited and short period of time.

Upon publication of the smart contract, distributors can engage by downloading the update from the DSN, thus making it available for the IoT devices.

The IoT devices become aware of an update release with the publication of the smart contract and can now consume the update from distributors over the DSN (see Section 4.4.2). The update exchange between a deployed device and a distributor relies on a trustless swap in which the deployed device is updated and the distributor acquires a proof-of-distribution (see Section 4.4.3), thus paid by the smart contract (see Section 4.4.4).

5. https://github.com/ethersphere/swarm
6. https://filecoin.io/filecoin.pdf

4.1. Decentralized Storage Network (DSN)

The decentralized storage network (DSN) must be accessible by all nodes (peers) in the network. Accessibility can be achieved with a trackerless peer discovery scheme, using a distributed hash table (DHT). This suggested scheme is similar to that of BitTorrent [23] and IPFS [28].

The connected DSN nodes can both consume and serve files. The set of DSN nodes is denoted as follows:

$$S = \{s_1, \dots, s_{n_S}\}$$

4.2. Blockchain Network

The blockchain network must meet the following properties:

1) Permissionless - any interested party (e.g., an anonymous person, organization, or company) can read and post messages on the blockchain network.
2) Support an intrinsic digital currency i.e., the blockchain can be interpreted as a public ledger in which each party has a currency balance.
3) Support smart contracts (as described in Section 2.2).

At the time of this writing, Ethereum [7] is the most widely adopted example of such an open permissionless blockchain which also supports Turing-complete smart contract languages, and thus is a recommended realization of our framework's required blockchain network.

The set of blockchain network nodes is denoted as:

$$B = \{b_1, \dots, b_{n_B}\}$$

4.3. IoT Updates Delivery Network

4.3.1. Vendor nodes. Vendor nodes are host machines that are owned by manufacturers of IoT devices. All vendor nodes must participate in both the DSN and the blockchain network. Within the blockchain network, vendor nodes must be able to function as both a wallet and a network routing node. The set of vendor nodes is denoted as:

$$V = \{v_1, \dots, v_m\}$$

where

$$V \subset S \wedge V \subset B$$

Every vendor node, v_i, must maintain the following:

1) a master secret/public keys pair (sk^{v_i}, pk^{v_i}).
2) a list of its self-manufactured IoT devices public keys. Each device must have a unique pair of keys. This can be realized by having the vendor "burning" a new secret key into each new manufactured IoT device and adding the key to its list.

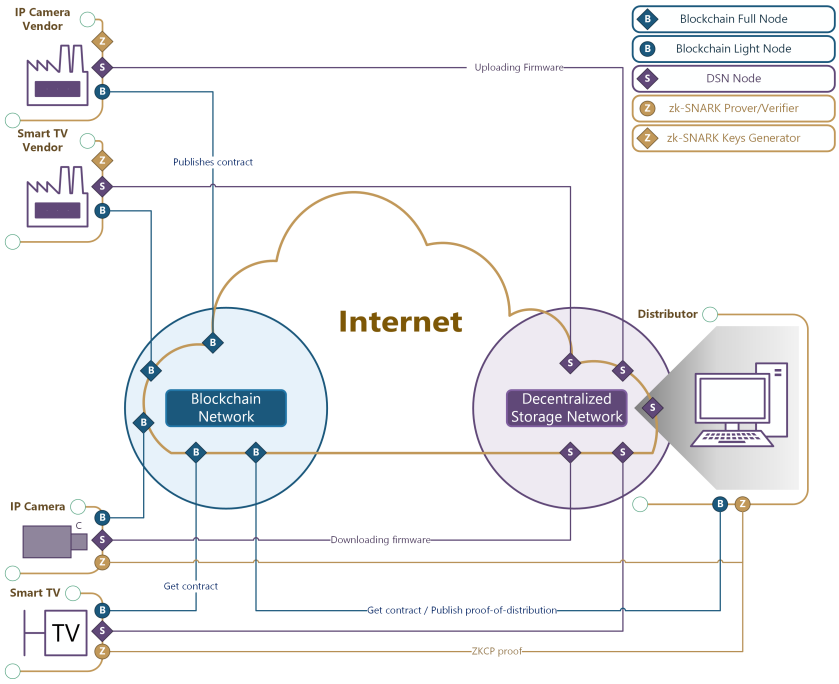

Figure 1. The architecture of the proposed framework.

4.3.2. IoT nodes. IoT nodes are machines that require updates by their manufacturing vendor. All IoT nodes participate in both the DSN and the blockchain network. Within the blockchain network, IoT nodes must be able to function as a network routing node. For every vendor $v_i \in V$, the set of its IoT nodes is denoted:

$$O_i = \{o_{i1}, \ldots, o_{in}\}$$

where

$$\forall i \in [1, m] : O_i \subset B \wedge O_i \subset S$$

Each IoT object, $o_{ik} \in O_i$, maintains the following:

1) a secret/public keys pair $(sk^{o_{ik}}, pk^{o_{ik}})$.
2) public key of its manufacturing vendor (i.e., pk^{v_i}).

The last prerequisite can also be met by having the vendor "burn" its own public key pk^{v_i} into the IoT device (thereby adding to the suggestion made in Section 4.3.1, we can have the vendor burn the pair $(pk^{v_i}, sk^{o_{ik}})$ into the deployed device).
The requirements of IoT nodes can be realized efficiently in disk space and memory, especially given that IoT devices are often limited in these resources.
IoT nodes are not required to store a complete copy of the blockchain. Instead, they can rely on a trusted available blockchain node (e.g., a gateway), or reduce the trust needed by consisting of a light client such as under development in Ethereum [7]. The purpose of the light client protocol is to allow light nodes to download only block headers as they appear, while fetching other parts of the blockchain on-demand, and be sure that the data is correct, provided that the majority of miners are following the protocol correctly, and that at least one honest verifying full node exists and serves the accurate blockchain state.
An alternative memory efficient design could be supported by Non-Interactive Proofs of Proof-of-Work [29] which could require only a soft-fork to existing blockchain protocols, and enable verification of a specific blockchain property, requiring only resources logarithmic in the length of the blockchain.
In addition, having the IoT function as a DSN node also does not impose significant space constraints. The IoT node is not required to share files in the network, and in the cases in which of the distributed hash table is used (e.g., the Kademlia DHT [30] used in BitTorrent) as the peer discovery scheme for DSN, only a small constant amount of memory is required for maintaining internal routing tables.

4.3.3. Distributor nodes. Distributor nodes are host machines that participate in an open bid for proofs-of-distribution. They must participate in both the DSN and the blockchain-based network. Within the blockchain network, distributor nodes must be able to function as both a wallet and a network routing node. Distributor nodes are denoted as:

$$D = \{d_1, \ldots, d_{n_D}\}$$

where

$$D \subset S \wedge D \subset B$$

7. https://github.com/ethereum/wiki/wiki/Light-client-protocol

4.4. Protocol

In this section, we describe the protocol used for a vendor $v_i \in V$ to send an update file U destined to a group of IoT objects manufactured by the vendor, $O_i = \{o_{i1}, \ldots, o_{in}\}$ (see Figure 2 for the protocol sketch).

4.4.1. Contract Publication. Upon releasing a new update U, vendor v_i performs the following phases:

1) Hashes the update file $U_{id} := H(U)$.
2) Generates the zk-SNARKs public proving/verification keys: $(pk^{POD}, vk^{POD}) := Gen(1^{\lambda}, C)$.
   Here λ is a security parameter, and C is a decision circuit.
3) Computes update file wrapping package:
   $P := (U, pk^{POD}, vk^{POD}, sign^{v_i}\{U_{id}, vk^{POD}\})$.
4) Hashes $P_{id} := H(P)$.
5) Sets Δ_{REFUND} to an acceptable time duration for a patch to be effective.
6) Posts a transaction to the underlying blockchain network which deploys a smart contract (in Algorithm 1 we describe pseudo-code for the contract suitable for Solidity), with a deposit f_{v_i} of money, the hash of the update file wrapping package P_{id}, and a program which essentially says:

   > For each object $o_{ik} \in O_i$:
   > Transfer $(f_{v_i} \setminus n)$ coins to the first party who provides proof that o_{ik} have committed to receive the hash preimage of U_{id}, within time Δ_{REFUND}.

   Assigning a time limit Δ_{REFUND} for publishing proofs is important so that the vendor can collect its refund if some of the IoT objects were not served after some set time (e.g., because sufficient time has passed such that a newer update file must be distributed, or because some remote objects are no longer active).
   Note that addressing the specific IoT devices for any security update becomes possible as the vendor maintains the list of their public keys.

4.4.2. Update File Initial Seeding. Routinely, distributor nodes (denoted D) and the manufactured IoT objects (denoted O_i) are watching the blockchain for an indication that a relevant smart contract is deployed by a vendor known by his public key pk^{v_i}. This is achievable, for example, by having the distributors and IoT objects to watch a single "factory contract" in which upon receiving a message with the variable arguments, will create a new instance of the child contract (described in Algorithm 1) and trigger an event with the public key of the bidding vendor and the address of the newly created contract.
Upon receiving this contract, the following process starts:

1) Distributors start to request downloading the update file wrapper package corresponding to the hash P_{id} which was published in the smart contract, using the underlying DSN.
2) The vendor v_i transfers (i.e., seeds) the package P via the DSN (off the blockchain). The file is then propagated until it eventually reaches some distributor $d \in D$ that wishes to participate in the open bid for the proofs-of-distribution.
   It should be made clear that this initial seeding by the vendor is meant to last a limited amount of time until a sufficient mass of distributors has downloaded the file, and from that point the distribution is expected to be performed completely by the competing distributors.
3) After completely downloading the package P, d verifies that:
   $VerifySig(pk^{v_i}, sign^{v_i}\{U_{id}, vk^{POD}\}, U_{id}||vk^{POD}) = 1$.
4) d registers itself in the underlying DSN peer discovery scheme, such as a distributed hash table (DHT), as a holder of the file U using its file hash U_{id}.

4.4.3. Update File Exchange for a Proof-of-Distribution. In this procedure, a distributor and an IoT object are conducting a two-party protocol, in which they perform a fair exchange of the update file for a proof-of-distribution, a digital signature given by the IoT object which can reward the distributor with a payment when sent to the smart-contract.
The fair exchange is achieved in a trustless manner by applying a zero-knowledge contingent payment (ZKCP [15]) which makes use of a zero-knowledge proof generated by the distributor and verified by the IoT object. This part is done in a protocol which is external to the blockchain and therefore does not require any specific modifications to the scripting language of the underlying blockchain nor does it add any burden to the blockchain nodes (so it does not cost any digital currency fees or *gas* in the Ethereum jargon).
In details, this process consists of the following steps:

1) $o_{ik} \in O_i$ performs a lookup for the file U using the publicly available file hash h_U from the smart contract deployed by v_i, and it reaches the distributor d by using the underlying DSN peer-discovery scheme (e.g., DHT).
   It then sends d a request to download the update file which is the hash preimage of U_{id}.
2) d sends a challenge c to o_{ik} to sign on.
3) o_{ik}:
   - Computes $sign^{o_{ik}}\{c\} := Sign(sk^{o_{ik}}, c)$.
   - Sends the tuple $(pk^{o_{ik}}, sign^{o_{ik}}\{c\})$ to d.
4) d:
   - Verifies that $pk^{o_{ik}} \in O_i$.
   - Verifies that $VerifySig(pk^{o_{ik}}, sign^{o_{ik}}\{c\}, c) = 1$.
   - Computes $t := Gen(1^{\lambda})$, where Gen is some secure random key generator for the security parameter λ.
   - Computes $r := H(pk^{d}||t)$.
   - Computes $s := H(r)$.
   - Computes $\hat{U} := Enc(U, r)$ where Enc is some symmetric encryption algorithm (e.g., AES).
   - Computes $x := (\hat{U}, s)$.
   - Computes the zero-knowledge proof:
   $\pi := Prove(pk^{POD}, x, r)$
   for the NP statement:

Algorithm 1: Pseudocode for the *Proofs-of-Distribution* bid contract

```
1  contract ProofsOfDistributionBid
2      function ProofsOfDistributionBid(v, e, h, o)
           // Constructor
3          owner ← v
4          expiration ← e
5          updateHash ← h
6          iot_objects ← o
7          numOfUpdatedObjects ← 0
8          balance ← value // Initial deposit
9      function publishProof(pk_o, t, s, pk_d, signature, r)
10         if block.timestamp ≥ expiration  return;
11         if !iot_objects[pk_o] ∨ iot_objects[pk_o].r ≠ null  return;
12         if r ≠ H(pk_d||t)  return;
13         if s ≠ H(r)  return;
14         if verifySig(pk_o, signature, updateHash||s)  return;
15         iot_objects[pk_o].r ← r
16         transfer(balance ÷ (n - numOfUpdatedObjects), pk_d) // Decrease balance
17         numOfUpdatedObjects ← numOfUpdatedObjects + 1
18         emitEvent("KeyRevealed", pk_o, r)
19     function withdrawFunds()
20         if block.timestamp < expiration  return;
21         if msg.sender ≠ owner  return;
22         transfer(balance, owner)
```

$\exists r$ s.t. $H(r) = s \wedge H(Dec(\hat{U}, r)) = U_{id}$.
- Sends the tuple $(x, \pi, vk^{VOD}, sign^{v_i}\{U_{id}, vk^{POD}\})$ to o_{ik}.

5) o_{ik}:
- Verifies that:
$VerifySig(pk^{v_i}, sign^{v_i}\{U_{id}, vk^{POD}\}, U_{id}||vk^{POD}) = 1$.
- Verifies that $Verify(vk^{POD}, x, \pi) = 1$.
- Sends the tuple $(sign^{o_{ik}}\{U_{id}, s\})$ to d.

4.4.4. Reward Claim for a Proof-of-Distribution. As a distributor d acquires the proof-of-distribution, he collects his reward as follows:

1) d:
- Verifies that $VerifySig(pk^{o_{ik}}, sign^{o_{ik}}\{U_{id}, s\}, U_{id}||s) = 1$.
- Posts a redeem transaction to the smart contract containing: $(pk^{o_{ik}}, t, s, pk^{d}, sign^{o_{ik}}\{U_{id}, s\}, r)$ which would release (f_{v_i}/n) coins to pk^{d} in return, according to the contract description (see function `publishProof` at Algorithm 1).

2) o_{ik}:
- Watches the blockchain nodes for a relevant event which is associated with the device's public key, $pk^{o_{ik}}$, with the associated decrypting key r (see event `"KeyRevealed"` in Algorithm 1).
- Computes $Dec(\hat{U}, r)$ to get U.

5. Security

The proposed framework encourages distributor nodes to participate in the protocol by delivering security updates to IoT devices in exchange for payment compensation. Therefore, a secure implementation of the system would include the guarantee for a fair exchange; i.e., regardless of a trust setting, every distributor node that delivers an update to a device is compensated correctly. This section briefly discusses potential attacks and their handling (Section 5.1) and ends with a formal claim and proof of the fair exchange (Section 5.2), thus guaranteeing that the system is secure.

5.1. Threat Analysis

In this work we consider the Dolev and Yao attacker model [31]. In this case an attacker can *read*, *send* and *drop* any transaction sent to the blockchain or any other network packet. In addition, the attacker can be a passive party eavesdropping on the network packets, or it can be active by injecting, replaying, or filtering any message to anyone of the parties involved. Based on this attack model we consider the following threats to the system:

Denial-of-service (DoS). An attacker can target a vendor by preventing legitimate transactions from appearing on the ledger, thus preventing the vendor from posting a new software update. That kind of an attack is typically referred to as a censorship attack. By construction, the attacker will need to control the majority of the mining power of the network, which is presumably difficult.

An alternative DoS attack is preventing a vendor from uploading new software update to the distributors network, or preventing an IoT device from downloading new software from the distributor. These attacks are mitigated using the multiple independent access points provided by the decentralized storage network. Using a decentralized peer-discovery scheme, such as a distributed hash table (DHT) as

used in BitTorrent [32] and IPFS [28] as examples, further allows fault tolerance and reduces single points of failure. Furthermore, an attacker could prevent the connection from the IoT device to the blockchain network, thus preventing the device from being notified about new software updates. However, the various access points given by the blockchain network assist in mitigating this threat.

Compromising software integrity. An attacker can impersonate a vendor and try to publish a malicious software update to the IoT devices. The integrity of the firmware update is ensured by the ability of the IoT device to verify the received file U by its hash U_{id} specified in the contract. The contract was created by the vendor in a transaction signed by him and verified against his known public key.

Software downgrade attack. An attacker may cause an IoT device to downgrade its software to an outdated version (which may include known vulnerabilities). This threat is mitigated using the consensus regarding the order of the blockchain's transactions, which implies ordering of the updates versions.

5.2. Proof of Fair Exchange

Let U_{ij} be the jth update of the vendor v_i.

Claim: Except a negligible probability in the security parameter λ, a distributer node d_q can receive the payment compensation $C(U_{ij}, o_{ik})$ if and only if:

1) d_q delivered the security update U_{ij} to the IoT device o_{ik}, and
2) No other distributor d_p $(p \neq q)$ received $C(U_{ij}, o_{ik})$

Proof: Let SC_{ij} be the smart contract published by the vendor v_i towards the delivery and compensation of the update U_{ij}. The contract SC_{ij} contains the set of IoT devices O_i, with their proofs-of-distribution (if received). By the contract construction, for every r_x, a successful proof-of-distribution for the update U_{ij} and device o_{ix}, the object o_{ix} is assigned with r_x only if it was not previously assigned. For any distributor d_p $(p \neq q)$ to receive $C(U_{ij}, o_{ik})$ prior to d_q, a published transaction reporting the proof-of-distribution d_p is required for U_{ij}, o_{ik}. If this transaction is indeed published, O_i already has an r assignment for o_{ik}, thus a distributor node d_q cannot receive $C(U_{ij}, o_{ik})$ as required. We therefore, continue by assuming $C(U_{ij}, o_{ik})$ was not yet received by any distributor.

Assuming $C(U_{ij}, o_{ik})$ is still available, in order for d_q to receive it – the smart contract SC_{ij} must induce $verify(\cdot) = 1$. Therefore, if d_q did not deliver the security update U_{ij} to the IoT device o_{ik} he is required to provide the contract with a forged published proof and yet suffice $verify(\cdot) = 1$. Based on the security parameter of the system λ, a distributor d_q can forge this proof only if:

1) The distributor forges o_{ik} digital signature; i.e., d_q forges $sign^{o}_{ik}\{U_{id}, s, pk^{d}\}$.
2) The distributor forges the entire zk-SNARKs' proof-of-knowledge.
3) The distributor uses the proof-of-distribution of another distributor, $d_{q'}$, as his own. i.e., d_q manages to find a hash preimage t' s.t $H(pk^{d_q}||t') = H(pk^{d_{q'}}||t) = r$.

Based on their definitions and the use of the λ parameter in the protocol, digital signatures, zk-SNARKs and hash preimaging are unforgeable except with negligible probability in λ. Therefore, d_q cannot receive $C(U_{ij}, o_{ik})$ without a proof-of-distribution except with a negligible probability in λ.

In the other direction, by the completeness of zk-SNARKs, a distributor d_q that delivers U_{ij} to the IoT device o_{ik} will be able to acquire a proof-of-distribution and can publish it to collect $C(U_{ij}, o_{ik})$, assuming that the IoT object will not disconnect in the middle of their two-party protocol. If no other distributor collected $C(U_{ij}, o_{ik})$, based on the irreversibility of the blockchain, the smart contract is binding, thus d_q will be able to receive the $C(U_{ij}, o_{ik})$ payment as required.

It should be noted that the vendor v_i, which has performed the zk-SNARKs setup to generate the proving and verification keys, can theoretically forge proofs to falsely convince IoT devices that he has delivered them the update file. However, since this will serve him only in getting his own vested funds from the smart contract, and beat his own original purpose to update his manufactured devices, we argue that this is not a real limitation of the protocol.

6. Discussion

Implementation of the proposed framework is based on several assumptions.

Amount of data needed to be stored and sent to the blockchain. Given the need for maximum security from forging signatures on behalf of IoT objects, we are required to actually list the entire set of public keys of the destination IoT objects. In addition, based on the hope that all of the micropayments offered will be redeemed, a similar number of transactions, each containing its corresponding proof-of-distribution along with the public keys of the distributor and the IoT object, would be sent to the blockchain.

Some optimizations can be made:

1) Using batched transactions for sending multiple redeem proofs - constructing the contract so multiple redeem transactions can be aggregated into one. This can be even more useful when combining with a signature aggregation scheme (such as Schnorr signatures) within the smart contract, as is planned to be implemented with Bitcoin [33]. The latter would help reduce the number of signatures needed in cases in which a distributor can serially send aggregated signatures to multiple IoT objects (the multiple messages still must be included in the transaction).
2) Sending an index of the public key in the redeem transaction instead of the whole public key.

At the time of this writing, Ethereum blockchain at its current capacity is able to process 10-15 transactions per second with blocks roughly every 15 seconds, where the median transaction fee is around 25 cents. As the underlying blockchain is currently the bottleneck of the suggested framework's scalability, other consensus layer scalability so-

lutions [34] and platforms can directly benefit our proposed solution.

Some trust assumptions regarding IoT objects. The proposed scheme allows the possibility of an IoT object o to cheat and pretend it is a distributor in order to reduce the need to upload an entire update file U to the blockchain. This can be done by posting a transaction to the smart contract containing the distributor's signature on some string r and its corresponding hash s, where r is not an actual random key that decrypts an encryption version of the update file U but is rather just some arbitrary data used to give o the desired micropayment.
Even though this dishonest behavior is possible in cases in which the IoT object's client software is hacked or compromised, we argue that the IoT software update micropayment cost was actually encompassed in the cost of the device when it was purchased, and we further argue that the act of updating the software is of greater benefit to the IoT buyer than settling for the coins instead.

7. Conclusions and Future Work

We have described a decentralized architecture which utilizes a trustless network to provide software updates to IoT devices. It is based on an economic model where compensation and cost are aligned with the requested service. Service providers (the distributors) have the incentive to perform honestly and quickly, since they know that they will get paid if and only if they will be the first to distribute updates to destination IoT objects, and will earn in proportion to the number of clients served. The presented system allows high availability of software updates with fault tolerance, integrity and the ability for a high quality of service in comparison to traditional architectures. Also, it is compatible but not limited to existing blockchain platforms such as Ethereum. For future work, we are interested in investigating performance benchmarks and scalability improvements which will enable the suggested ecosystem to stretch its ability to support the ever-increasing number of IoT devices.

References

[1] C. Bekara, "Security issues and challenges for the iot-based smart grid," *Procedia Computer Science*, vol. 34, pp. 532–537, 2014.

[2] S. Ray, A. Basak, and S. Bhunia, "Patching the internet of things," *IEEE Spectrum*, vol. 54, no. 11, pp. 30–35, 2017.

[3] S. A. Kumar, T. Vealey, and H. Srivastava, "Security in internet of things: Challenges, solutions and future directions," in *System Sciences (HICSS), 2016 49th Hawaii International Conference on*. IEEE, 2016, pp. 5772–5781.

[4] B. Lee and J.-H. Lee, "Blockchain-based secure firmware update for embedded devices in an internet of things environment," *The Journal of Supercomputing*, vol. 73, no. 3, pp. 1152–1167, 2017.

[5] A. Boudguiga, N. Bouzerna, L. Granboulan, A. Olivereau, F. Quesnel, A. Roger, and R. Sirdey, "Towards better availability and accountability for iot updates by means of a blockchain," in *IEEE Security & Privacy on the Blockchain (IEEE S&B 2017) an IEEE EuroS&P 2017 and Eurocrypt 2017 affiliated workshop*, 2017.

[6] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[7] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *white paper*, 2014.

[8] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "Bitcoin-ng: A scalable blockchain protocol." in *NSDI*, 2016, pp. 45–59.

[9] R. Pass and E. Shi, "Fruitchains: A fair blockchain," in *Proceedings of the ACM Symposium on Principles of Distributed Computing*. ACM, 2017, pp. 315–324.

[10] N. Szabo, "The idea of smart contracts," *Nick Szabos Papers and Concise Tutorials*, 1997.

[11] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct nizks without pcps," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2013, pp. 626–645.

[12] N. Bitansky, A. Chiesa, Y. Ishai, O. Paneth, and R. Ostrovsky, "Succinct non-interactive arguments via linear interactive proofs," in *Theory of Cryptography*. Springer, 2013, pp. 315–333.

[13] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "Snarks for c: Verifying program executions succinctly and in zero knowledge," in *Advances in Cryptology–CRYPTO 2013*. Springer, 2013, pp. 90–108.

[14] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture." in *USENIX Security Symposium*, 2014, pp. 781–796.

[15] B. Wiki, "Zero knowledge contingent payment," 2011.

[16] K. Liu, D. Zou, and H. Jin, "Uaas: software update as a service for the iaas cloud," in *Services Computing (SCC), 2015 IEEE International Conference on*. IEEE, 2015, pp. 483–490.

[17] X. Zhen-hai and Y. Yong-zhi, "Automatic updating method based on maven," in *Computer Science & Education (ICCSE), 2014 9th International Conference on*. IEEE, 2014, pp. 1074–1077.

[18] T. Yu, V. Sekar, S. Seshan, Y. Agarwal, and C. Xu, "Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the internet-of-things," in *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*. ACM, 2015, p. 5.

[19] Y. Onuma, Y. Terashima, and R. Kiyohara, "Ecu software updating in future vehicle networks," in *Advanced Information Networking and Applications Workshops (WAINA), 2017 31st International Conference on*. IEEE, 2017, pp. 35–40.

[20] C. Huth, P. Duplys, and T. Güneysu, "Secure software update and ip protection for untrusted devices in the internet of things via physically unclonable functions," in *Pervasive Computing and Communication Workshops (PerCom Workshops), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1–6.

[21] D.-Y. Kim, S. Kim, and J. H. Park, "Remote software update in trusted connection of long range iot networking integrated with mobile edge cloud," *IEEE Access*, 2017.

[22] S. Popov. (2017) The tangle. [Online]. Available: https://iota.org/IOTA_Whitepaper.pdf

[23] B. Cohen, "Incentives build robustness in bittorrent," in *Workshop on Economics of Peer-to-Peer systems*, vol. 6, 2003, pp. 68–72.

[24] D. Hughes, G. Coulson, and J. Walkerdine, "Free riding on gnutella revisited: the bell tolls?" *IEEE distributed systems online*, vol. 6, no. 6, 2005.

[25] A. Legout, G. Urvoy-Keller, and P. Michiardi, "Rarest first and choke algorithms are enough," in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. ACM, 2006, pp. 203–216.

[26] S. Wilkinson, T. Boshevski, J. Brandoff, and V. Buterin, "Storj a peer-to-peer cloud storage network," 2014.

[27] D. Vorick and L. Champine, "Sia: Simple decentralized storage," 2014.

[28] J. Benet, "Ipfs-content addressed, versioned, p2p file system," *arXiv preprint arXiv:1407.3561*, 2014.

[29] A. Kiayias, A. Miller, and D. Zindros, "Non-interactive proofs of proof-of-work," 2017.

[30] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *International Workshop on Peer-to-Peer Systems*. Springer, 2002, pp. 53–65.

[31] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on information theory*, vol. 29, no. 2, pp. 198–208, 1983.

[32] R. Jiménez, F. Osmani, and B. Knutsson, "Connectivity properties of mainline bittorrent dht nodes," in *Peer-to-Peer Computing, 2009. P2P'09. IEEE Ninth International Conference on*. IEEE, 2009, pp. 262–270.

[33] G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille, "Simple schnorr multi-signatures with applications to bitcoin," Cryptology ePrint Archive, Report 2018/068, 2018, https://eprint.iacr.org/2018/068.

[34] F. Ehrsam, "Scaling ethereum to billions of users," https://medium.com/@FEhrsam/scaling-ethereum-to-billions-of-users-f37d9f487db1.

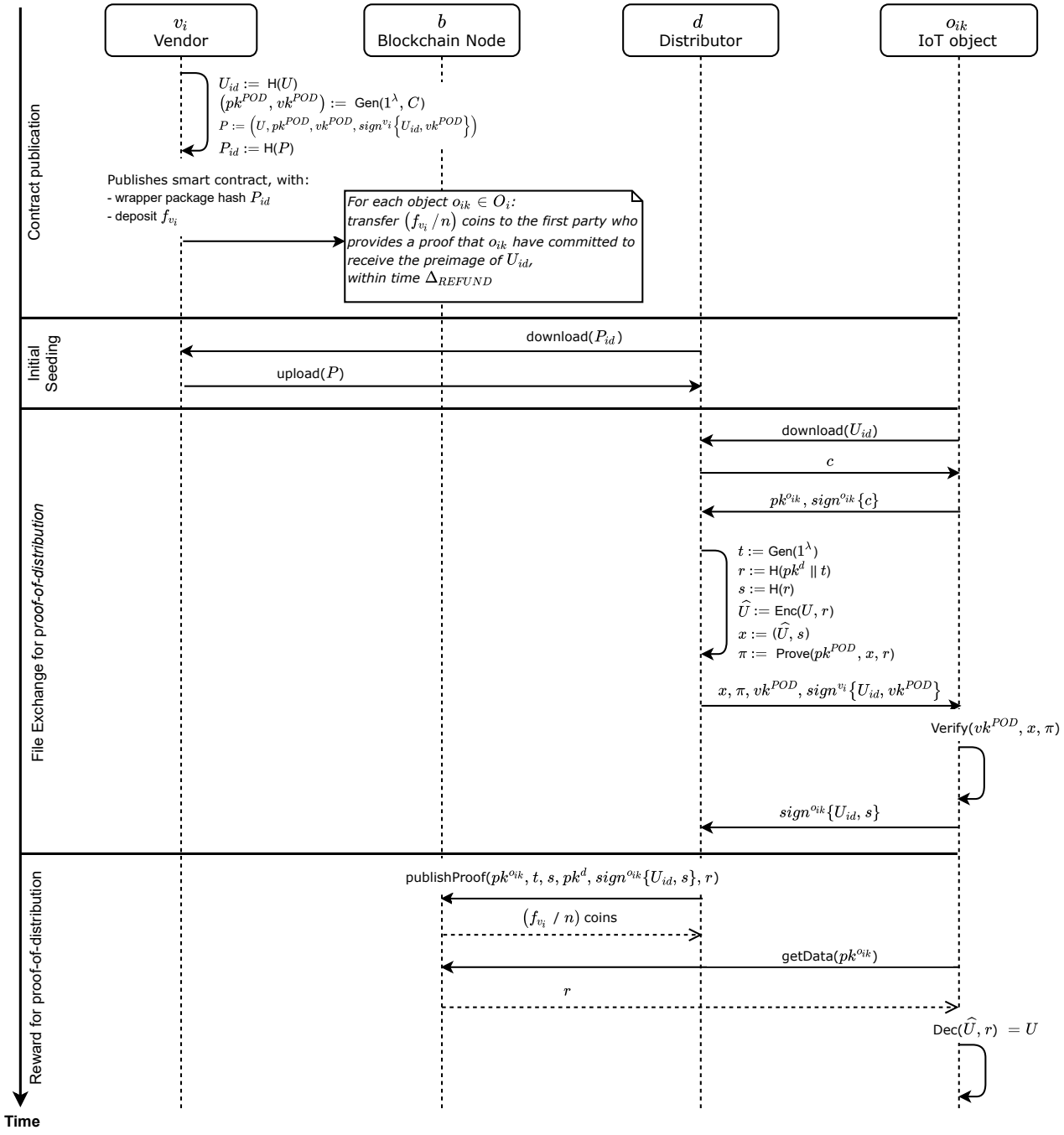

Figure 2. IoT software update protocol sketch. Here C is a decision circuit, x is the instance and r is the secret witness, for the NP statement $\exists r$ s.t. $H(r) = s \wedge H(Dec(\hat{U}, r)) = U_{id}$