Blockchain-enhanced Roots-of-Trust

Vitor Jesus
School of Computing and Digital Technology
Birmingham City University
Birmingham, United Kingdom

***Abstract*—Establishing a root-of-trust is a key early step in establishing trust throughout the lifecycle of a device, notably by attesting the running software. A key technique is to use hardware security in the form of specialised modules or hardware functions such as TPMs. However, even if a device supports such features, other steps exist that can compromise the overall trust model between devices being manufactured until decommissioning. In this paper, we discuss how blockchains, and smart contracts in particular, can be used to harden the overall security management both in the case of existing hardware-enhanced security or when only software attestation is possible.**

I. Introduction

Any aspect in security will in some way and at some point be tied to a chain of trust whereby the trust of the overall process or system is as secure as its weakest link. For example, in any key management scheme, key storage and distribution is typically the most complex step and requires a number of assumptions. Smart devices and the Internet-of-Things (IoT) add the new challenge of location considering that many use-cases include a device on untrusted premises and physically accessible. A smart thermostat at home is a straightforward example: one can only trust the hardware and the software it runs as long there are guarantees nobody had physical access to it in the past and there are no software vulnerabilities that can be remotely exploited. This brings us to hardware-enhanced software security. The only way, arguably, of verifying (or attesting) the current software image it is using is by means of special hardware functions such as processors using Trusted Execution Environments or a dedicated cryptographic module, e.g., Trusted Platform Module (TPM) from TGC [1].

A typical TPM offers a number of basic cryptographic services directly embedded in electronics. For example, it can hold in isolation, directly in the hardware microelectronics, a set of keys and is able to perform de/encryption of messages without the keys ever being exposed. Another service is a persistent memory that can hold, for example, signatures of the software that is used to boot the device while disabling key parts of the hardware until the verification is complete. Combining such features, a strong *root-of-trust* is established in the sense that, if one is able to securely associate such isolated keys to a physical device throughout its lifecycle, one can be sufficiently sure any underlying security process or protocol has not been compromised.

There is however two aspects that weaken this chain of trust. On one hand, devices almost never work in isolation and, in a typical IoT use-case, there will be some gateway nearby the device and further services provided by a Cloud, generally speaking. Further, not all devices will have an on-board TPM or similar hardware-based functions. This could be, for example, due to cost, complexity or constraints on resources. The second fact that weakens the chain of trust is the handling of the device from factory until activation. Commissioning a remote device needs a trusted process in itself which, again, is based on more or less weak assumptions. For example, the device needs to arrive in a trusted state (trusting the supply chain), then provisioned (typically involving creating keys) and finally be attested at least once immediately before being activated for production.

On the other hand, the device also needs to trust what it is receiving from the Cloud counterpart. Striking examples is receiving firmware updates or the device being an actuator and receiving commands. This is a simpler problem as, typically, there is no limitation of resources and hardware security is available. Furthermore, and central to this paper, whereas devices need to be assumed to operate from an untrusted location, one can expect the cloud infrastructure to be physically secure and subject to structured security workflows.

This paper discusses such chains of trust for smart/embedded devices and discusses the use of blockchain technologies, and Smart Contracts, to mitigate such weaknesses. Our strategy is to use Blockchains' immutability and auditing properties and use them in secure attestation, verification and overall management. Specifically, we propose a scheme that offers two features. First, we use smart contracts to provide an emulation of hardware cryptographic services similar to a remote and virtual TPM. In an extreme case, one can verify the device has not been tampered with but at the cost of making the device unusable until manual and comprehensive verification or re-provisioning. The second feature of our scheme is that we use the underlying blockchain to provide a secure message bus over which devices and cloud can communicate sensitive information such as public keys.

In Section II we review key concepts of hardware and firmware security and the concept of Smart Contracts over Blockchains. In Section III we lay out our Device and Threat models and we align with the common lifecycle of a device. In Section IV we discuss our approach to the problem that we evaluate in Section V. Section VI concludes our paper and discusses future work.

II. Background and Related Work

In this section we review related work on Trusted Software

and Smart Contracts in the context of Internet-of-Things.

A. Trusted Software

Hardware-enhanced trusted software execution has seen different proposals [2] that range in complexity, typically categorized by the ability of only performing attestation functions or the ability to run complex software in an isolated environment. TPMs is one of such and are a technology standardized by the Trusted Computing Group [1]. It has been used in many areas but with mixed adoption since whereas virtually all cloud servers, many laptops and mobile phones have one, embedded devices such as those used in Wireless Sensor Networks (WSN) or IoT do not, for cost or implementation complexity reasons. TPMs can be used in any task that relies on handling secret material, such as authentication [3][4]. Whenever there is no TPM available, a root-of-trust needs to be established in other ways [5] even if the trust level may be reduced [6]. Alternatives include software-based attestations such as the one proposed by Seshadri et al [7] that relies on loading trusted software in memory and sending a signed footprint to an external verifier. Software attestation is however challenging to implement in practice and often depends on the specific architecture of the device to be feasible [8].

Note, however, that, when looking at the workflow and the device lifecycle, secure management of keys still shows gaps even with hardware security. A good example is that a manual enrolment phase (such as simply connecting a cable) is typically required where keys and secure boot measurements (cryptographic hashes) are attested and recorded to build a baseline and used in later comparison. This verified binding between cryptographic material, identity and function of the device is part of the problem we address in our paper using Smart Contracts.

B. Blockchains and Smart Contracts in IoT

Blockchains are a recent technology that cleverly enables trustless, distributed and open verification applications by, normally, heavily using computing power. In its original form (seen in Bitcoin), data is stored in blocks, which are linked together using strong cryptography as they are created, thus forming a chain of records whose immutability increases as new blocks are added. Every new set of records, or a block, is verified by many nodes (the consensus layer) and subject to a resource-intensive cryptographic process (mining) that is similar to brute-forcing a hash thus providing strong assurances that, after enough time elapsed (several blocks of data and concatenated hashes), the data cannot be changed. Smart Contracts take this concept further by allowing not only records but also code to be executed. In its original form, data and code is publicly auditable. Since the information is open and tamper-resistant, the system is trustless since any unauthorized or unexpected modifications, at the time of submission, are visible to every participant. Applications to IoT and trusted execution are thus immediate [8] and they include enhanced security, privacy and identity [10], verification of the supply chain [8], distribution of firmware updates (taking advantage of both assurances of integrity and distributed topology of nodes) [11][12] or enhancing trust in Industrial IoT [12] or coordinating business workflows as in electricity co-generation [14].

To the best of our knowledge, this is the first work exploring the use of Smart Contracts in designing *generic* hardware-emulated, security services, joining trusted and untrusted software, and using Blockchains as a trusted communication channel. In other words, we aim at using Smart Contracts to either enhance or create Roots-of-Trust. Boudguiga et al [12] and Novo [15] propose architectures to perform a specific task such as verification of IoT updates or performing access control, typically by designing a blockchain architecture with which nodes will fully integrate instead of using it as service. Further, they do not use smart contracts and use the blockchain as immutable and verifiable storage in a custom blockchain architecture. A similar proposal is the case of Machado and Frohlich [17], but using smart contracts and different consensus algorithms, who propose an integrity verification architecture based on blockchains and fit for IoT, constrained devices and real-time applications. Wu et al [16] discuss a related problem of using a blockchain as an out-of-band communication channel, a concept we also use, but for authentication.

III. Threat Model

For end-to-end and continuous trust, the device must be secured across all stages in its lifecycle. We start by introducing a generic threat model that is aligned with a general lifecycle. We then discuss the requirements our approach expects to meet along with identifying limitations and in the next section we elaborate on our proposals.

A. Device Lifecycle

We start by considering a generic lifecycle of devices. This allows to extract a threat model and design our architecture and smart contracts. We assume a device goes through the following stages, from manufacturing to disposal:

1. *Factory* – the device is manufactured and an early firmware (or bootloader) is installed. This first layer of firmware which, beyond accessory functions (such as power-on tests), is the software component that will load further components up to an Operating Systems and user applications. Being the first software layer, it must support and undergo full verification as all further verifications will depend on it.
2. *Supply Chain Handling* – The device is then physically distributed and may be handled by several parties until it arrives to the last owner (e.g., end user or service manager). There may be the case where other parties install a new firmware that needs to be also verified by the end user.
3. *Commissioning* – The device may be now configured, physically installed, provisioned and integrated with the cloud at its final location. All these steps may be remote or by an end-user. Physical attacks are possible. The device is further registered and integrated with a

Cloud service. This requires, at least, verification of firmware, generation of root keys and (possibly) a unique identifier for the device. Note that, for the verification part, it is desirable to commission a device remotely with little or no local effort and allowing for device/user unlinkability, similar to Intel's SDO approach [18] that uses group keys instead of per device.

4. *In-Service* – The device is now ready to send and receive data or commands. At this point, both hardware and software needs to be verified and protocols used should be secure.
5. *Maintenance* – The device undergoes maintenance stages such as software updates and reconfigurations. Note this stage repeats in time and interleaves the In-Service stage.
6. *Decommissioning* – the device is removed from service and is disposed, reinstalled or repurposed.

Considering these stages, a hardware-based attestation, such as a TPM, is invaluable to effectively create a state of trust over which trust in all other dependencies can be built on. There are however aspects that a TPM cannot address. Furthermore, as discussed, it is not possible to have a TPM on every device so other scenarios and solutions need to be considered, up to accepting service from untrusted hardware. The next subsection presents a list of threats per stage in the lifecycle of the device.

B. Threat Model

The key threats are the following:

1. The device was compromised during manufacturing, which includes hardware trojans or modified firmware.
2. Taking advantage of physical access to the device, it is modified during handling across the supply chain or in later phases which includes the end-user.
3. A different device altogether was delivered.
4. During commissioning or maintenance, the device is installed using compromised software.
5. Illegitimate modification of the software by a legitimate user because of physical access after being installed.
6. Insecure management of secret material during Commissioning phase such as keys and identity of the device were correctly generated by a legitimate party but storage and/or transmission was insecure.
7. Data received may not be from the device or may have been intercepted or modified in-flight.
8. Device receives data from malicious parties that are able to spoof the cloud services which includes compromised firmware and configurations.

Other threats may be considered (e.g., privacy, access control or availability) but the problem we tackle is, simply speaking, about authenticity and integrity. Furthermore, note we're considering the two directions: both device and cloud exchange data that needs to be trusted data.

IV. Scenarios

In this section we describe our approach to use smart contracts to establish roots-of-trust. We start by considering three types of devices: (i) with a TPM, (ii) without a TPM but supporting tamper-detection and (iii) software-based attestation. We then elaborate on how smart contracts can enhance trust using three representative scenarios in the lifecycle of a device: (i) bootstrapping a newly manufactured device, (ii) software-attesting a device with no TPM but supporting tamper-detection after installation at an untrusted location and (iii) a smart meter reading. We then discuss our assumptions and the effectiveness of the approach.

In the diagrams, we use the following notation:

- a dashed line means detection of new state, after the blockchain consensus layer converged
- Solid lines require interaction with the blockchain, thus changing the state of the contract
- Changing the state of contract needs a public and private key pair (where the public key is often the address and identity of the device interfacing the contract) and an address of the contract itself. These are configurations that need to be present at the device from the beginning
- We denote encryption of `data` with key `key` by `key[data]`.

A. Bootstrapping a device

We start by looking at a device that has just been manufactured, in terms of hardware, and is ready to be packaged and sent to either the final customer or to another party that will further configure or install software. We assume the device has, at this early stage, a firmware able to participate in attestation and tracking actions. We further assume the device has been configured with means to interact with a smart contract such as keys, an address, a whitelist of blockchain nodes and perhaps a token or cryptocurrency.

See Figure 1. At this stage, the device actions two different contracts. The first contract `supply_chain`, which must be pre-existing and mutually managed between the manufacturer and any owners of the device, will track its journey and modifications until arriving the final location. The second contract, `device`, is created on-the-fly by the device and will track its authenticity and integrity. Note that, depending on the attestation method, it may not matter whether the device has the right firmware or configuration. If the device fails any verification, it is not accepted into its final use. The two contracts are expected to be linked but only at the beginning so the address of `device` can be found.

The device checks-in for the first time with a factory identifier that we are calling here `sn` (thinking of a serial number). The Cloud counterpart, here representing any server infrastructure on the side of the (future) owner of the device, will detect a new device (denoted by a dashed line) once their blockchain nodes synchronise. Note this is not a directed message. The device will also register a public key (or a full

certificate, simply represented by `pubD`) for future private communications as will show and unique ID, `guid`. All these elements are generated locally by the device, either using a TPM or running normal software and the public elements are then published by executing a method in the smart contract running in the blockchain.

The device is now ready to be verified and steps that require traceability are communicated using the blockchain. The Cloud also publishes its certificate (`cert`). At this stage, both device and Cloud have each other's identity. The Cloud requires now an attestation to which the device confirms by sending a message directly. The firmware location (`fwLoc`) is sent, possibly encrypted with `fwKey` (which is sent encrypted with the Device's public key), along with any other parameters (`params`) the device requires to run the attestation. If a software attestation is done, these parameters may include a checksum function and a prover binary, among others. The device fetches the firmware directly from a cloud server, installs and an attestation process is executed. Both parties record results on the blockchain. The results should be later checked for consistency. Finally, as the device changes hands, similar actions may take place thus recording any modifications using smart contracts.

B. Device with tamper-detection

This scenario has relevance in case of a device that has no means to verify the integrity of its hardware or software but has means to detect tampering to some satisfactory degree of trust. A simple solution is at the cost of physically destroying some functionality of the device. This scenario is challenging if only because once physical access has been breached, and in the absence of a trusted attestation process, it is virtually impossible to fully prevent modification by physical reprogramming.

We nevertheless assume there was at some point a verification that could be, for example, a manual inspection. Further, we assume there is a trusted physical mechanism (such as switch) that, upon activation, will put the device in a lockdown mode disabling all external interfaces, not accepting any new software and only running a specific application that will interface the smart contract methods (Figure 2). As soon as the device detects physical violation, it will update the contract with a report and not allow any further action. This report needs to be acknowledged by the Cloud at which point the device could allow reduced operation depending on the policy object (`policy`) coming from the Cloud. The lockdown will be removed once a confirmation is read in the blockchain coming from the Cloud.

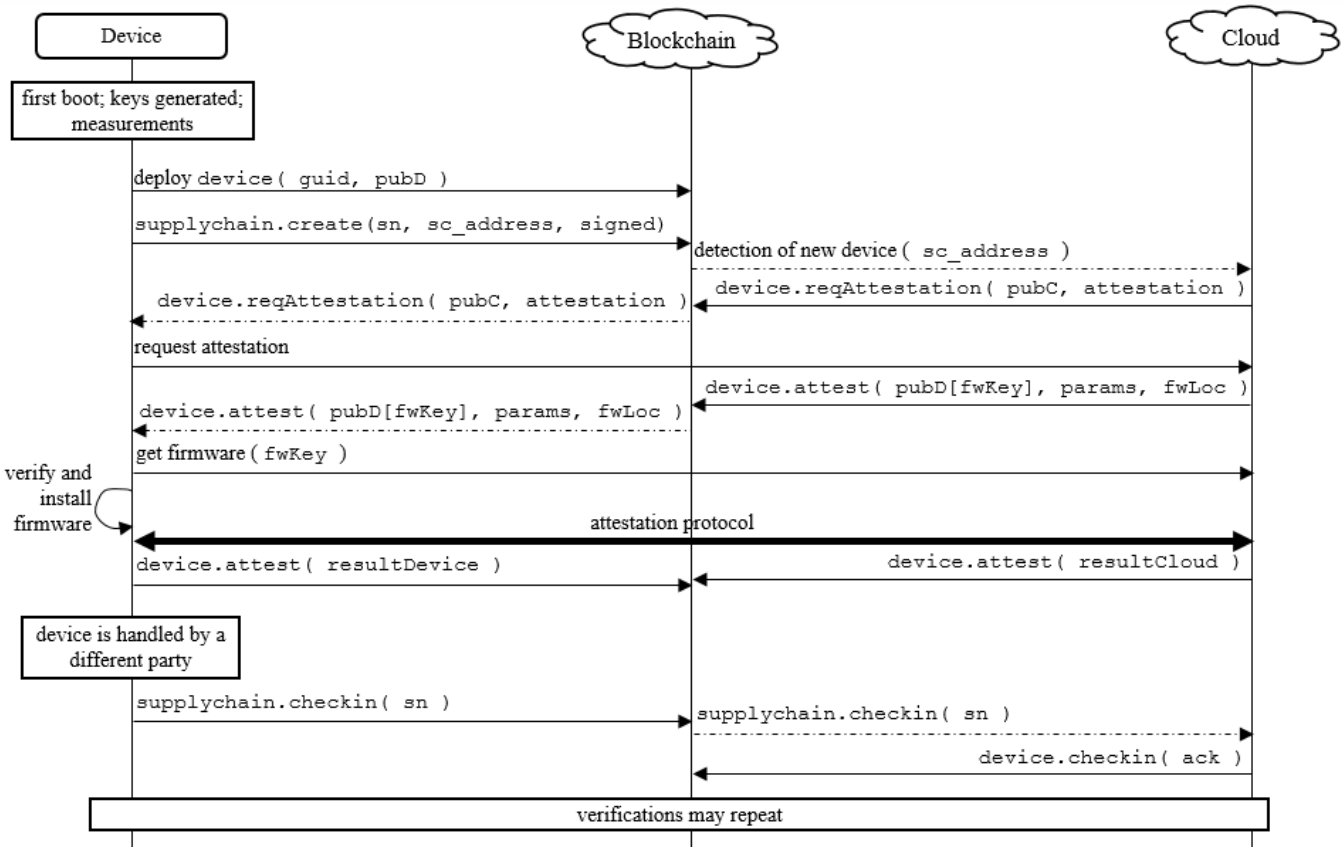

Figure 1. Bootstrapping a device.

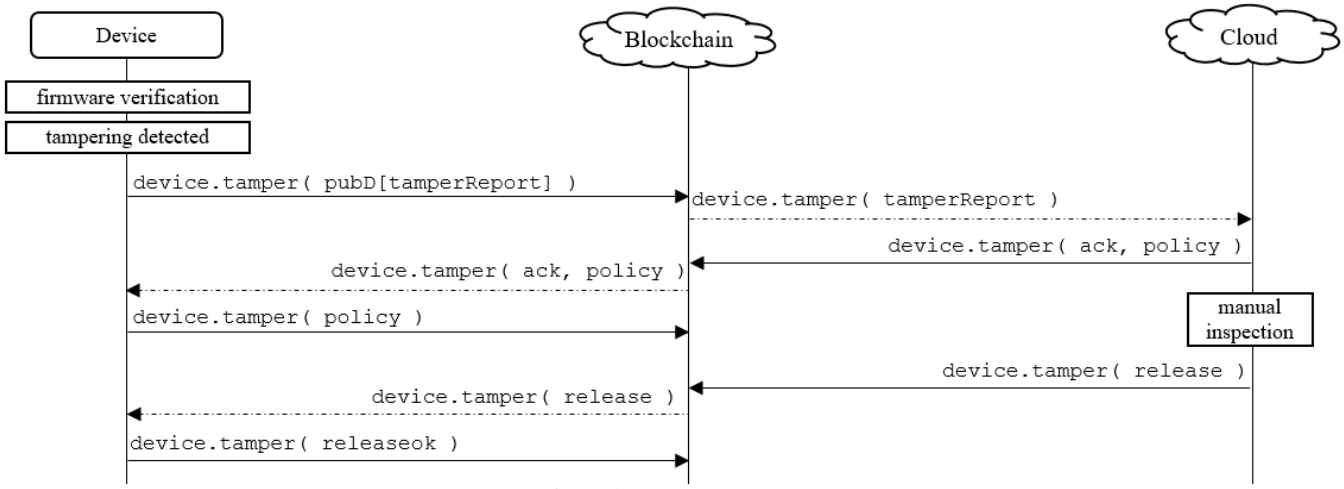

Figure 2. Tamper detection.

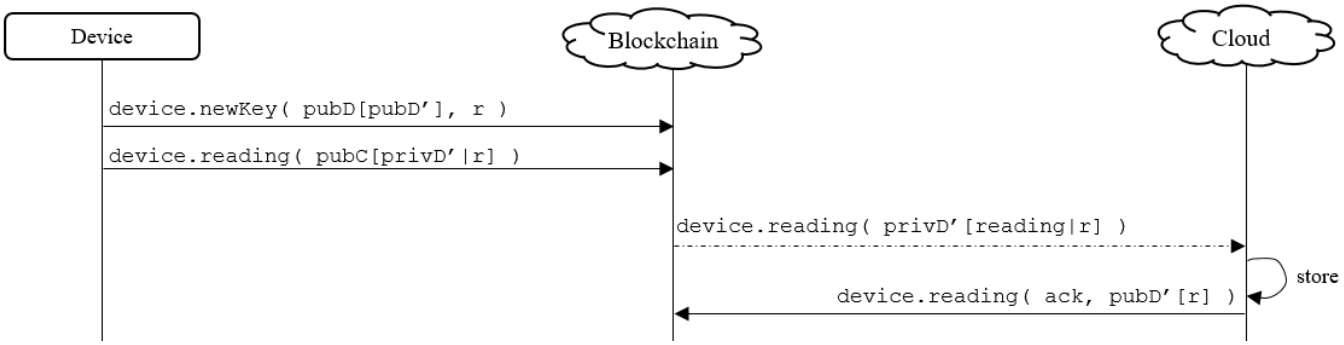

Figure 3. Smart Reader scenario.

C. *Smart Meter reading*

Figure 3 shows a scenario where a device is using a secure protocol to update a value. Given this potentially involves private personal information (e.g., electricity readings), we need to support confidentiality and forward secrecy. The device starts by creating a temporary key: `pubD'` is the public part that is stored in the contract and `privD'` which is kept local to the device. For forward privacy, these keys are only used for this reading, and a random number to prevent replay attacks and confirm the specific reading order. The key is ephemeral and is used to protect the reading until confirmed storage when it is discarded. Note, however, that depending on the blockchain type information cannot be discarded as such but only revoked.

D. *Discussion*

A key feature these three use-cases highlight that can only be achieved with blockchains is that of a secure medium over which messages are passed and, to any arbitrary level, broadcasted to any number of nodes and as many as possible to improve both security and resilience. All three mechanisms can be done in a centralised fashion using pairs of nodes; for example, the location of the firmware can be negotiated over TLS. The problem with peer-to-peer protocols is that it typically needs a previous step to bootstrap trust, such as a list of certification authorities that must be known by both parties engaging in the protocol. A blockchain-enhanced architecture does significantly remove that need and essentially provide (pseudo-) centralised means to store unmodifiable information while being distributed in nature and thus resilient to any arbitrary level. A further advantage is that, for data that needs to be trusted, recorded and auditable at any point in the future, and by parties that do not trust each other, such as a smart-meter reading that may be disputed at some point.

It also provides verifiable means to handle functions that a typical TPM provides such as generation of keys, storage of state (such as hashes of firmware) or generation of random number. Naturally, private keys cannot be stored in the clear so they either need a public key counterpart or, if symmetric, need to be protected by a secret that needs to be stored locally thus vulnerable in the absence of a TPM. In any case, note that updating the smart contract needs a transaction (often paying with a cryptocurrency) which requires a secret key.

A further advantage of using smart contracts is that even with TPMs some previous provisioning information needs to be securely shared prior to the device being provisioned and installed. When attesting a device, the measurements of the device need to be compared to a trusted template that is typically unique to the device when considering local configuration when devices are installed for a particular use or

user. A blockchain elegantly solves the problem from an architectural perspective even if storage of secret material such as keys needs hardware unless software-attestation is acceptable. A combination of both, however, TPM and blockchains, is able to create perfect security from an architectural perspective (i.e., excluding implementation vulnerabilities).

A key challenge is, however, how devices will participate in the blockchain. There are two basic scenarios: passively *reading updates* to the blockchain (such as when the cloud publishes information) and actively executing or *writing state* in the blockchain.

Regarding reading information, the ideal scenario is for the device itself to be a full node in the blockchain thus receiving and validating (but not mining) every new block. The contracts can therefore be inspected and executed locally as a copy exists. first is the device receiving full updates to changes in contracts, and all contracts active in the blockchain, thus being able to verify by itself state and consistency. This may depend on the chosen blockchain implementation, e.g., a public one such as Ethereum or a private and permissioned one with, in principle, weaker security since the security of a blockchain depends on its scale. In both cases, and depending on how constrained the device is, storing the full blockchain, or even just the current state and an integrity metric (similar to a hash of all transactions and block headers), requires vast amounts of storage that is blatantly incompatible to the typical IoT device. In other words, currently, one needs to rely on a trusted gateway which, despite typically existing in a IoT architecture, introduces a vulnerable point. Note however that the device may have a list of different nodes on the wider Internet (so outside the local network) that can be used to as uncorrelated sources (trusting the network links) to verify the state of a contract. Multiple nodes can be configured, along with public keys and well-known identities, and they either agree on the state of the contracts, after mined, or some may have been compromised and the perceived state, from the device's perspective, cannot be trusted. In other words, and this is a strength of blockchains, in order to compromise the interface of the device with the blockchain, either the link is compromised (such as the gateway spoofing network responses) or most nodes in the list of the device need to be compromised. The overall problem is left for future work but current solutions point to an significant increase in the architectural complexity [12][15].

Writing, executing or updating a contracts seems simpler and is more of a problem of managing keys – which can be managed in similar ways as presented before. In the impossibility of a device being a full node, this needs the cooperation of a trusted set of nodes. The transactions that underpin the execution of a method in the smart contract require very little space (say, ~1 kByte). They are signed and authenticated in-band (or otherwise the transaction does not validate) and are sent to any node participating in the blockchain. In the worst case, the transaction needs to be repeated and any pending protocol is stalled. This may not be a critical problem since.

V. IMPLEMENTATION

In this section we discuss simplified code samples of smart contracts. We use Ethereum and the Solidity language. For simplicity of presentation and lack of space, we will omit unnecessary details in the code and Solidity syntax.

The first smart contract is shown in Figure 4 and implements part of the bootstrapping functionality of `Device`. Contracts in Ethereum pay for both storage (roughly the objects at the top and the constructor) and execution cycles (roughly the functions). Information that needs to be stored for the lifetime of the Device is, among other, its public key, a factory identification (such as a serial number) and the addresses of the parties that can interact the contract (the device and a counterpart server in the Cloud).

The device first needs to generate a unique identifier (`guid`), e.g., from its serial number. It also needs to generate locally at least one public and private key pair (ideally a certificate) Keys should be generated from the TPM if existing. When first deploying the contract, the `constructor` will populate these objects.

```
contract Device {
  //...
  Key publicKey;
  string serialnumber, guid;
  Address ownerDevice, ownerCloud;
  AttestationResult attResDevice, attResCloud;

  constructor() public {
    ownerDevice = DEVICE_ADDRESS;
    ownerCloud == CLOUD_SERVER_ADDRESS;
    serialnumber = SERIAL_NUMBER;
    publicKey[0] = PUBLIC_KEY_DEVICE;
    guid = GUID;
  }

  function decommission() public {
    if (msg.sender == ownerDevice
      || msg.sender == ownerCloud)
      selfdestruct( cloudAddress );
  }

  function managePublicKey( party );
  function reqAttestat( Key pubC, Attestation att );
  function attest( Key fwKey, Url fw_url );

  function attestRes ( AttestResult r) {
    if (msg.sender == ownerDevice)
      attResDevice == r;
    if (msg.sender == ownerCloud))
      attResCloud == r;
  }
}
```

Figure 4. Bootstrapping pseudocode for contract `Device`.

The other methods are aligned with the signalling diagrams described before. The bootstrapping contract also implements access control in the form of allowed addresses. For example, when reporting the attestation result, only a (signed) transaction coming from the device's address (`ownerDevice`) or a trusted server (`ownerCloud`) can update that information.

Figure 5 shows a simplified contract to report tampering detection. As explained, as soon as the device detects an

incident, we assume it activates a lockdown mode, record its state in the blockchain and will wait for a policy from the cloud server. The device will not change its state until the cloud counterpart sends a message, in the blockchain, to release the hardware.

```
contract Device {
  Key publicKey;
  TamperReport tr;
  Address ownerDevice, ownerCloud;
  bool devLocked;

  constructor() public {
    ownerDevice = DEVICE_ADDRESS;
    ownerCloud == CLOUD_SERVER_ADDRESS;
    publicKey = PUBLIC_KEY_DEVICE;
    tr.detected = devLocked = false;
  }

  function tamperReport() public {
    if (msg.sender == ownerDevice) {
      devLocked = tr.detected = true;
      tr.nounce = random();
      tr.signed = sign(PUBLIC_KEY_DEVICE, tr);
      //other tr attributes
    }
  }

  function setTamperPolicy( TamperPolicy policy );

  function ackTamperPolicy() {
    if ( msg.sender == ownerDevice )
      tr.ackDevice = true;
  };

  function tamperRelease( bool status ) {
    if ( msg.sender == ownerCloud ) devLocked = 0;
  }
}
```

Figure 5. Tamper-detection smart contract.

As expected, running these contracts is rather slow when compared with point-to-point protocols. We deployed and ran the contracts in a local Ethereum test network where no other contracts were being executed. This assured that every block was predictable given the low load in mining and confirmations occurred after about 15 seconds. Running a simple tamper-detection protocol took several minutes just for exchanging messages and updating the protocol state.

Also, the full contracts themselves, both `Device` (more complex) and `SupplyChain`, were on the order of 1 kB and, hence, manageable. The transactions themselves, that devices can hold templates locally, would range between 150 bytes when recording just a flag (e.g., `tamperRelease()`) and a few kB for the case of recording a certificate.

VI. Conclusions and Outlook

This paper offered an approach to use blockchains and smart contracts to enhance current methods of establishing a root-of-trust. We describe our approach and discuss its theoretical and practical feasibility. We also present a brief implementation for the Ethereum blockchain. Our approach raises further questions that our future work will address, which include implementing our proposal in an actual testbed with heterogeneous devices. A key open direction is how to integrate devices, which are likely to be constrained in some aspect, in a large blockchain such as Ethereum. This may have two approaches: designing a private blockchain architecture and, complementary, to combine the security of a distributed blockchain with conventional point-to-point techniques. This means putting a component of trust in external nodes which, in a IoT system likely involves a gateway.

References

[1] Trusted Computing Group: http://www.trustedcomputinggroup.org/. Accessed 3-Aug-2018

[2] Shepherd et al, *Secure and Trusted Execution: Past, Present, and Future - A Critical Review in the Context of the Internet of Things and Cyber-Physical Systems*, 2016 IEEE Trustcom, Tianjin, China

[3] W. Hu, H. Tan, P. Corke, W. Chan Shih, S. Jha, *Toward Trusted Wireless Sensor Networks*, ACM Transactions on Sensor Networks, Vol. 7, No. 1, Article 5, August 2010

[4] RV Steiner, E Lupu, *Attestation in Wireless Sensor Networks: a Survey*, ACM Journal Computing Surveys, Vol 49, Issue 3, No 51, Dec 2016

[5] SJ Johnston, M Scott, SJ Cox, *Recommendations for securing Internet of Things devices using commodity hardware*, IEEE 3rd World Forum on Internet of Things (WF-IoT), Reston, VA, USA, 2016

[6] Hailun Tan, Gene Tsudik, Sanjay Jha, *MTRA: Multiple-Tier Remote Attestation in IoT Networks*, 2017 IEEE Conf on Communications and Network Security (CNS),

[7] Seshadri et al, *SWATT: SoftWare-based ATTestation for Embedded Devices*, Proc IEEE Symposium on Security and Privacy (ISCC), 2004

[8] Y. Li, Y. Cheng, V. Gligor, A. Perrig, *Establishing software-only root of trust on embedded systems: Facts and fiction*, Security Protocols XXIII, Springer, 2015, pp. 50–68.

[9] K Christidis, M Devetsikiotis, *Blockchains and Smart Contracts for the Internet of Things*, IEEE Access, V. 4, 2016

[10] Kouzinopoulos C.S. et al., *Using Blockchains to Strengthen the Security of Internet of Things*. Security in Computer and Information Sciences. Euro-CYBERSEC 2018, vol 821. Springer, 2018

[11] Boohyung Lee, Jong-Hyouk Lee, *Blockchain-based secure firmware update for embedded devices in an Internet of Things environment*, The Journal of Supercomputing, Volume 73, Issue 3, March 2017

[12] Boudguiga et al, *Towards Better Availability and Accountability for IoT Updates by means of a Blockchain*, 2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), Paris, France

[13] Bahga, A., Madisetti, V.K., *Blockchain platform for industrial Internet of Things*, J. Softw. Eng. Appl. 9(10), 533 (2016)

[14] Zhang, Y., Wen, J.: *The IoT electric business model: using blockchain technology for the Internet of Things*, Peer-to-Peer Netw. Appl. Vol 10, issue 4, 2017

[15] Oscar Novo, *Blockchain Meets IoT: An Architecture for Scalable Access Management in IoT*, IEEE Internet Of Things Journal, v5, n2, Apr 2018

[16] Wu et al, *An Out-of-band Authentication Scheme for Internet of Things Using Blockchain Technology*, 2018 Intl Conf on Computing, Networking and Communications (ICNC), Maui, Hawai

[17] C Machado, AA Frohlich, *IoT Data Integrity Verification for Cyber-Physical Systems using Blockchain*, IEEE 21st International Symposium on Real-Time Distributed Computing, 2018

[18] Intel, *Intel Secure Device Onboard (SDO)*: https://www.intel.com/content/www/us/en/internet-of-things/secure-device-onboard.html, accessed 08-August-2018.