A Blockchain Proxy for Lightweight IoT Devices

Gero Dittmann, Jens Jelitto
IBM Research – Zurich
8803 Rüschlikon, Switzerland
{ged,jje}@zurich.ibm.com

***Abstract*—IoT devices are quickly becoming a critical source of information about the physical world considered in business processes. Blockchains are a promising platform for such processes if they involve multiple parties with no shared, commonly trusted IT infrastructure. Transacting with a blockchain, however, requires software whose footprint overwhelms many lightweight IoT devices.**

In this paper we introduce the concept of a blockchain proxy to which an IoT device can offload a large part of this software footprint. The proxy only requires a slim proxy SDK on the device that holds a regular blockchain identity with its own private key, retaining full control of the transactions in the device. We discuss security implications and present cold-chain monitoring as a use case. Preliminary results show significant savings in CPU time and communication bandwidth for the IoT device.

I. Introduction

Blockchains are an emerging platform for automating business processes across organizations, such as supply-chain management (SCM) or asset life-cycle management. The internet of things (IoT) can deliver critical inputs to such processes, in particular GPS coordinates or sensor readings such as temperature, humidity, pressure, mechanical shock (impact) and vibrations, reporting the status of shipments or meteorological data. Cold-chain monitoring, for instance, can be implemented by having a temperature sensor in a shipping container report its readings to a blockchain-based SCM system in regular intervals.

To trust those readings, the receiver must be able to securely identify the sensor and to verify that the readings have not been manipulated on the way to the blockchain. In the context of a permissioned blockchain this can be achieved by furnishing the sensor with a key pair and having a public-key infrastructure (PKI) issue a certificate for those keys, giving the sensor an identity on the blockchain. The sensor signs its readings with its private key and sends it to the blockchain along with its identity certificate. In this way, a receiver can verify the sensor identity and the integrity of its readings using the certificate and the signature.

IoT sensors, however, are often constrained in their compute power, memory size and communication bandwidth. In this paper we propose to minimize the footprint of connecting an IoT device to a permissioned blockchain by offloading the communication with the blockchain to a proxy service, except signing the transactions which remains with the device. As the blockchain identity rests with the sensor, even the proxy service cannot modify the readings without detection as it cannot forge the sensor device's signature.

The remainder of this paper is structured as follows: Section II discusses related work. Section III introduces our system architecture and Section IV applies it to a cold-chain monitoring use-case. Section V discusses trust assumptions and security threats. Section VI details a prototype implementation and Section VII gives some preliminary experimental results. Section VIII concludes and proposes future work.

II. Related work

In recent years, some vendors have started deploying public-key infrastructures that offer identities to IoT sensors [1], [2]. They issue keys and certificates to be installed in a sensor and then used to sign readings before sending them out. The trustworthiness of sensor readings can be improved by employing trusted execution environments [3]. Others rely on anomaly detection to detect rogue readings [4] or attacks on sensors [5]. There is also work on distributed processing of sensor data [6]. None of these approaches have any specific connection to blockchains. Non-blockchain identities can be used to sign only the payload of blockchain transactions but verifying these signatures requires replicating the infrastructure the blockchain already has.

To connect a sensor to a blockchain one could use the regular blockchain SDK (software development kit, [7]) directly on the sensor. However, this approach has a rather significant compute, memory and communication footprint that cannot be supported by lightweight IoT devices. The other straightforward approach is to run the blockchain SDK on a server with a blockchain identity that receives readings from the sensor and signs them for the blockchain. In this arrangement, however, the blockchain clients need to trust the server as the readings are not protected from modification at the server.

Our approach, in contrast, keeps the blockchain identity on the sensor to prevent such modification but offloads the part of the blockchain communication that is not critical to security. To the best of our knowledge, this is the first method to give a lightweight IoT device its own blockchain identity, establishing end-to-end trust from a sensor to remote blockchain clients.

III. System Architecture

Our approach assumes a blockchain with PKI-based identity management. Figure 1 shows the system view of our approach. A PKI certificate authority (CA) is registered with the blockchain, establishing it as an identity provider trusted by the blockchain peers. The CA issues an identity certificate to an IoT sensor which uses the certified private key to sign its blockchain transactions. The sensor sends the signed transactions along with its certificate to the blockchain proxy which executes the required protocols with the blockchain nodes to commit the transactions to the ledger.

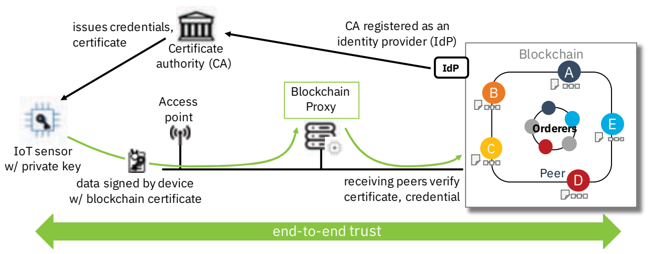
issues credentials, certificate
Certificate authority (CA)
CA registered as an identity provider (IdP)
IdP
Blockchain
IoT sensor w/ private key
Access point
Blockchain Proxy
Orderers
Peer
data signed by device w/ blockchain certificate
receiving peers verify certificate, credential
end-to-end trust

Fig. 1. End-to-end trust from an IoT device to the blockchain.

Reconfiguring an IoT device in the field is expensive and may not always be possible, depending on the device and use case. The IoT identity issued by the CA can be long-lived and re-used for different blockchains. The receiving blockchain may map the data received from one device to different processes over time.

IV. Use case: calibrated cold-chain monitoring

Supply-chain management is a much-discussed application for both blockchains and IoT. The many independent parties involved in SCM make a blockchain an attractive platform for tracking shipments and managing the associated data. IoT sensors can report the location and environmental conditions of a shipment, like temperature and humidity, to an SCM system.

Cold-chain monitoring is a particularly critical aspect of SCM as it must ensure that temperature-sensitive products such as food or vaccines are stored below their temperature thresholds at all times lest they spoil, rendering them worthless or even dangerous to consumers. IoT sensors in a package or shipping container can monitor the temperature from sender to recipient and either attest proper cooling or report a violation as soon as it happens.

Sensor readings can only be considered trustworthy, however, if (1) the sensor can be securely identified, (2) the sensor works reliably and (3) the readings cannot be modified on the way from the sensor to the SCM system. By enabling the sensor to sign its readings, our proxy approach ensures that both (1) the sensor can be securely identified and (3) the readings cannot be modified.

Proper operation of a sensor (2) is traditionally ensured by authorities that calibrate and test sensors (e.g., industrial scales) and issue certificates and seals to reassure the recipients of their readings. In our system, the calibration authority can act as a certificate authority as well by signing up as an identity provider for the blockchain, issuing identity certificates upon calibration that expire when the next calibration is due.

Recipients of a sensor reading can verify that the sensor's certificate has been issued by a trustworthy calibration authority and the calibration has not expired. The authority attests to both the sensor identity and the reliability of its readings. In addition, it can apply seals to verify the physical integrity of the device. The required physical inspection of the seal at a recipient's end is comparable to verifying a container's seal against unauthorized opening. If the recipient finds the seal intact, it vouches for all readings up to that point in time.

In a cold chain this setup enables the sender and recipient of a temperature-sensitive product to monitor the shipment's temperature throughout the supply chain, relying on sensors verifiably calibrated by trustworthy authorities.

V. Security analysis

A. *Adversarial model*

The principals in our system, shown in Figure 1, are the IoT device, the blockchain proxy, the peers that maintain the blockchain, and the CA. The asset to be protected is the data the IoT device is collecting about its environment. This IoT data must be shielded against tampering as it travels from the device to the blockchain. As our approach relies on securely identifying the device, its private key is also an asset to be protected.

Of the principals we trust the blockchain peers to reliably authenticate and immutably commit valid transactions to the shared ledger. We trust the CA to issue credentials strictly to trustworthy IoT devices. By extension, we trust any IoT device with an identity provided by the CA. If the CA is also a calibration authority, as in the cold-chain use-case, it vouches not only for the identity of the sensor but also for the accuracy of its measurements and, if it applies a seal, for the sensor's physical integrity.

Potential adversaries are parties with an interest in making the IoT environment appear different from what it really is. In the cold-chain use-case, a carrier might want to save fuel by skimping on cooling a shipment. To avoid detection, the carrier would try to make the temperature appear below the stipulated threshold. Other adversaries bent on disrupting business might try to arbitrarily modify IoT data. To mount an attack, an adversary needs to control the device environment, the proxy or the network.

An adversary with physical access to the IoT device could tamper with its environment, e.g., by cooling only the temperature sensor but not the actual shipment. Other physical attacks are to swap an IoT device against one controlled by the adversary or to impersonate the IoT device, e.g., by extracting the device's private key.

An adversary controlling the network could tamper with the device's readings in transit anywhere between the device and the blockchain. There is also a risk of denial-of-service (DoS) attacks by interrupting one of the networks between the device, the proxy and the blockchain.

An adversary controlling the blockchain proxy could mount a man-in-the-middle attack. A replay attack could be used to simulate acceptable temperatures when a threshold is actually being violated. Another vector is to make the proxy delay, reorder or drop individual messages, or to shut down the proxy altogether.

For some use cases, confidentiality of the IoT data between device and blockchain may be a concern, either on the network or even with respect to the proxy.

B. Analysis and mitigation

The device identity granted by the CA combined with the required authentication of each blockchain transaction make it difficult for an adversary to swap or impersonate the device. When a use case carries the risk of a physical attack on the device's private key the key must be stored in a secure element.

Deceiving the IoT device by modifying its environment is not prevented by our system. This threat applies to any system relying on an individual device, independent of the back-end to which it reports its readings. It can be addressed by correlating readings from multiple sources to detect incoherence and potentially derive a more trustworthy result by majority [4]. This approach can be combined with our system.

Tampering with the readings in transit would require the adversary to forge the device signature on the modified data. Otherwise, the tampering will be detected by the blockchain peers as the original signature will not validate tampered readings. Replay attacks are prevented by the blockchain protocol employing unique transaction IDs, nonces, and by the fact that only the IoT device has the private key to sign messages.

DoS and reorder attacks can be detected by subscribing to a blockchain event that confirms the submitted transaction has been committed to the shared ledger. If the event does not arrive before a deadline the device can resubmit the transaction. We plan to add this capability to the proxy SDK in the future.

If confidentiality of the IoT data on the network is a requirement it can be addressed by using TLS for each hop between the device, proxy and blockchain, respectively. If even the proxy must not see the data in the clear then the device can encrypt the data before inserting it into a transaction, either to be decrypted by the chaincode or to be stored on the shared ledger encrypted. Any additional encryption, however, increases the software footprint for the IoT device.

VI. Implementation

We have implemented a *Fabric Proxy* for Hyperledger Fabric, a permissioned blockchain with pre-order execution and separate nodes (*peers*) for ordering and committing transactions. Fabric clients create transaction proposals, sign them and send them to endorsing peers, a subset of committing peers. Each endorser simulates the transaction and sends an endorsement, consisting of the simulation result signed by the endorser, back to the client. The client collects enough endorsements to satisfy the required endorsement policy into a transaction, signs it and sends it to the orderers. They sequentialize transactions, group them into blocks and distribute them to the committing peers who validate them and commit valid transactions to their local copy of the distributed ledger.

To support application development, the basic client functionality is available as Fabric SDKs for a variety of languages. We picked the SDK for the Go language [7] and carved out the client identity and transaction signing, resulting in a slim SDK stub for the IoT device and the remaining SDK back end for the Proxy, as shown in Fig. 2. We connect both parts by MQTT [8], a lightweight publish-subscribe messaging protocol on top of TCP/IP that is widely used in IoT for its small footprint. MQTT clients communicate via a broker that manages messages by topics. A client that subscribes to a topic receives all messages other clients publish to that topic.

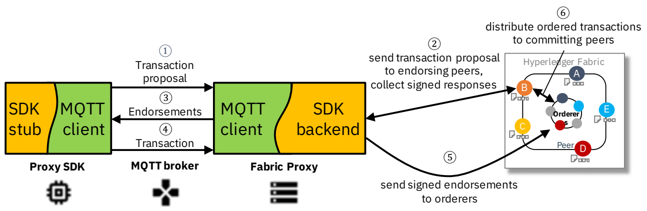

Fig. 2. Fabric Proxy system design.

Topics can be divided into subtopics, separated by a slash (/) by convention. We use a first sub-topic to identify whether the message was published by ("from") or addressed to ("to") an IoT device. The second sub-topic is the device's MQTT client ID. In this way, a device does not need to know which Proxy to talk to—only the Proxy needs to know which devices to subscribe to. This makes load-balancing and fail-over between proxies possible without reconfiguring any devices. A third sub-topic identifies the message type corresponding to the numbered messages in Fig. 2. An example topic for a device7 to send a transaction would be: `from/device7/sendTX`

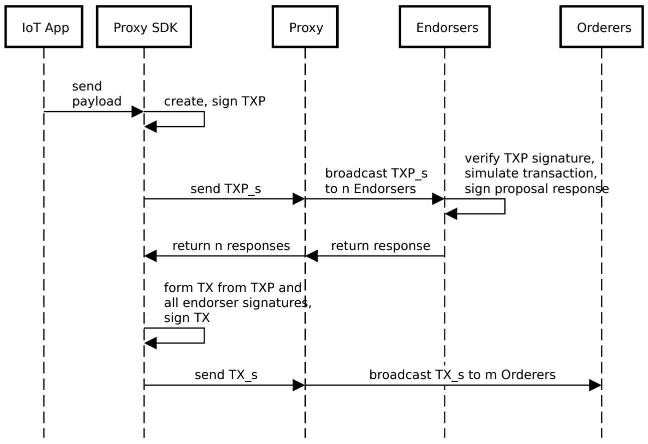

Fig. 3. Flow from an IoT application to the Fabric peers via the Fabric Proxy. The IoT app and the Fabric SDK are both running on the IoT device. *TXP: transaction proposal; TX: transaction; _s: signed.*

Fig. 3 shows the detailed sequence of function calls and messages to commit a transaction to the ledger. An application (*IoT app*) calls the Proxy SDK, both running on an IoT device, to submit a transaction to the blockchain via the Fabric Proxy in the cloud. The IoT app composes the transaction payload and sends it to the Proxy SDK. The SDK creates and signs the transaction proposal (*TXP*) using the device's blockchain identity and sends the result to the Proxy which broadcasts it to the required set of *n* endorsing peers.

Each endorser verifies the device's signature of the TXP and simulates the transaction to generate the read-write set of accessed ledger variables. The endorser signs the read-write set and returns it as a proposal response to the Proxy. The Proxy collects the proposal responses from all endorsers and returns them to the SDK. As all endorsers should produce the same read-write set, the Proxy can strip all but one copy from the responses to save bandwidth to the IoT device.

Without further interaction with the IoT app, the SDK forms the final transaction (TX) from the read-write set and all endorser signatures, signs it and sends it to the Proxy. The Proxy broadcasts the signed transaction to *m* orderers from where the Fabric infrastructure ensures the transaction will be committed to the ledger, provided it is valid.

Other than holding an identity (keys and certificate), the device needs to be configured with the channel ID, chaincode name, version and function to invoke with which parameters. This information is required for transaction forming. The remaining configuration resides with the Proxy, comprising the addresses of peers to contact (endorsers and orderers).

As an example blockchain application we've implemented an SCM system that associates temperature and humidity readings with a shipment. For the sensor we use a Raspberry Pi with a Sense HAT sensor board, running our Proxy SDK. The Fabric Proxy and the MQTT broker are deployed as cloud services.

VII. Preliminary results

The goal of the Proxy is to reduce the compute power and bandwidth consumed by blockchain transactions on an IoT device. To assess the potential savings conservatively we compare individual transactions sent using the regular SDK against the same transactions sent via the Proxy SDK. This means the measurements include the full set-up overhead with no amortization over multiple transactions. The SDK code has not yet been slimmed down at all. The blockchain setup features only a single endorsing peer, which represents the lower bound of the bandwidth saved by offloading the communication with the endorsers to the Proxy (proposal broadcast, endorsement collection).

As the regular SDK is not available for lightweight IoT devices we used a laptop with an Intel Core i7-4770HQ CPU clocked with 2.20 GHz, assuming the results will scale roughly linearly to the weaker processors found in IoT devices. We used Go's `pprof` package to measure CPU time and the `nload` shell tool for the network bandwidth consumed.

TABLE I
Preliminary measurements for an individual transaction.

metric	regular SDK	Proxy SDK	savings
CPU time	130 ms	80 ms	38%
data sent	14 KB	11 KB	21%
data received	90 KB	17 KB	81%

Table I presents the results. In spite of the conservative setup and the lack of optimization there are significant savings for the IoT device in CPU time as well as data sent and received. Further studies will have to corroborate those findings and explain the savings in detail.

VIII. Conclusions and future work

In this paper we have introduced a blockchain proxy as a service for lightweight IoT devices to offload communication with a blockchain while retaining full control of all transactions committed to the shared ledger. We have argued that the approach is robust against tampering with the device data in transit and delivers trustworthy readings to the blockchain. Preliminary results of a proxy for Hyperledger Fabric demonstrate the potential for the IoT device to save significant CPU time and communication bandwidth using the proxy.

We are currently working on a Proxy SDK in C to support the most common programming language for truly lightweight IoT devices such as Arduinos. As a truly minimum implementation this SDK will provide savings in memory size as well. As future work we plan to support subscribing to Fabric events via the Proxy to harden the system against DoS attacks. Events also provide a channel back from the blockchain to the device that can be used to control the device and trigger actions there. Having transactions write sequence numbers to a shared-ledger variable would protect the system against DoS attacks, malicious reordering and selective dropping of transactions.

Acknowledgments

The authors would like to thank Thanos Amoutzias for the performance measurements, Björn Tackmann for conceptual advice, and Alessandro Sorniotti for technical education and feedback.

References

[1] H. Liu, S. Saroiu, A. Wolman, and H. Raj, "Software abstractions for trusted sensors," in *Proceedings of MobiSys '12*. ACM, June 2012, pp. 365–378.
[2] G. Prophet. (2017, March) Certificate authority solution for IoT manufacturers. Smart2Zero. [Online]. Available: https://www.smart2zero.com/news/certificate-authority-solution-iot-manufacturers
[3] H. Janjua, W. Joosen, S. Michiels, and D. Hughes, "Trusted operations on sensor data," *Sensors*, vol. 18, no. 5, May 2018.
[4] T. Ide, "Collaborative anomaly detection on blockchain from noisy sensor data," in *Proceedings of ICDMW; Workshop on Blockchain Systems for Decentralized Mining (BSDM)*. IEEE, November 2018, pp. 120–127.
[5] M. Raciti and S. Nadjm-Tehrani, "Embedded cyber-physical anomaly detection in smart meters," in *Proceedings of Critical Information Infrastructures Security (CRITIS '12)*. Springer, September 2012, pp. 34–45.
[6] K. Parmar and D. C. Jinwala, "Malleable cryptosystems and their applications in wireless sensor networks," in *Computer and Network Security Essentials*. Springer, August 2017, pp. 97–111.
[7] Hyperledger Fabric Client SDK for Go. [Online]. Available: https://github.com/hyperledger/fabric-sdk-go
[8] MQTT. [Online]. Available: https://mqtt.org/