Secure and Effective Implementation of an IOTA Light Node using STM32

Diego Stucchi, Ruggero Susella, Pasqualina Fragneto, Beatrice Rossi
diego.stucchi-ext@st.com,ruggero.susella@st.com,pasqualina.fragneto@st.com,beatrice.rossi@st.com
STMicroelectronics, System Research and Applications
Agrate Brianza, Italy

ABSTRACT

A major challenge in networked sensor systems and other IoT environments is addressing security. Vulnerabilities in those systems arise from poor physical security, unauthenticated devices, insecure firmware updates, insecure communication, and data corruption. In recent times Distributed Ledger Technologies (DLTs), of which Blockchain is an instance, have been identified as a possible solution to some of these issues. The blokchain model genetically ensures decentralized security and privacy, and therefore could provide IoT systems with a trusted infrastructure for securely logging data or exchanging tokens without the necessity, and costs, of central servers. Blockchain is no panacea, either. IoT devices that get connected to a blockchain network must still be secured, in particular they must protect the confidentiality of the keys. This requires the embedded microcontroller to execute only authenticated firmware, with protections against software attacks, such as buffer overflows, and resistance against side-channel attacks. In addition, as confirmed from the scarcity of implementations reported in the literature, it is still not clear whether blockchain protocols can be implemented efficiently on resource-constrained IoT devices. In this work, also supported by a Demo, we show an example of secure IoT device that enables the functionalities of IOTA, a DLT specifically designed for the use in the IoT. In particular, we present a Light Node based on STM32 that implements all the cryptographic functions, IOTA specific operations and communication functions required to successfully publish transactions in the IOTA distributed ledger. Our implementations on microcontrollers (ARM Cortex-M) performs up to 22 times faster in terms of cycles and up to 4 times faster in absolute time with respect to the state-of-the-art implementation on a Raspberry PI 3B. Our Light Node also ensures protection of the stored private data and guarantees secure firmware update thanks to a suitable configuration of some security features provided by STM32 microcontrollers.

KEYWORDS

Networked Systems, Internet of Things, Distributed Ledger Technology, Security, IOTA, STM32

ACM Reference Format:
Diego Stucchi, Ruggero Susella, Pasqualina Fragneto, Beatrice Rossi. 2019. Secure and Effective Implementation of an IOTA Light Node using STM32. In *2nd Workshop on Blockchain-enabled Networked Sensor (BlockSys'19), November 10, 2019, New York, NY, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3362744.3363344

1 BACKGROUND ON IOTA

IOTA is a DLT specifically designed for the use in the IoT. The IOTA distributed ledger, called Tangle, is formed by the transactions issued by the nodes in the IOTA network. IOTA adopts a rather unconventional approach based on trinary representation which uses trits = -1, 0, 1 instead of bits, and trytes of 3 trits instead of bytes. An IOTA transaction is a 2673 tryte-long string that can either withdraw/deposit IOTA tokens or send data. Transactions that refer to the same transfer of tokens are packed together in a structure called bundle. Every transaction must contain a Proof-of-Work (PoW) nonce, while only withdrawing transactions must contain a valid signature. The IOTA network includes Full Nodes and Light Nodes. Full Nodes are connected to their neighbors and store a copy of the Tangle, while Light Nodes are IoT devices identified with a seed used to create addresses and sign transactions. After receiving a valid transaction from a Light Node, a Full Node adds it to its copy of the Tangle, updates the balances of the affected addresses and broadcasts it to its neighbors. A complete description of the IOTA protocol can be found at [1].

2 CONTRIBUTIONS

In this work, also supported by a Demo, we describe an IOTA Light Node implemented using STM32 [3]. Our implementations extend the investigation carried out in [5] performing up to 22 times faster in terms of cycles and up to 4 times faster in absolute time, making us thinking to the applicability of IOTA to IoT under a more optimistic light. We also describe how to protect the private data stored in our Light Node and to guarantee secure firmware update thanks to a suitable configuration of some security features provided by STM32 microcontrollers.

2.1 An IOTA Light Node on STM32

We developed our Light Node on the STM32 boards Nucleo-F746ZG and Nucleo-F429ZI, both equipped with an Ethernet module. The specs of the boards are reported in Table 1 and the complete documentation is available at [3]. Our Light Node implements three main functionalities: cryptographic hash functions (Curl, Keccak, Kerl), IOTA specific operations, (Account Manager, Bundle Manager, Diver) and communication functions. For Curl we used the reference IOTA implementation available at [2], while for Keccak we replaced the IOTA implementation with an assembly one. Kerl is

Platform	SoC	CPU Core	frequency	RAM	Cores	Prepare and sign			PoW
						l = 1	l = 2	l = 3	
STM32	STM32F746ZG	Cortex-M7	216 MHz	up to 320kB	1	80ms	201ms	391ms	307,9s
STM32	STM32F429ZI	Cortex-M4	180 MHz	up to 256kB	1	135ms	349ms	687ms	623,3s
TI Launchpad	CC2650	Cortex-M3	48 MHz	20kB	1	6382ms	7716ms	-	not feasible
Raspberry PI 3B	BCM2837	Cortex-A53	1200 MHz	1Gb	4	328ms	372ms	-	82.9s

Table 1: Performances of our Light Node (STM32) compared those reported in [5]. In prepare and sign our results are averaged over 1000 trials, while in PoW over 125. The letter l indicates the considered security level. Times are computed as number of cycles divided by frequency.

implemented as a conversion from trits to bits, followed by a series of evaluations of Keccak. The Account Manager generates addresses and signatures from the seed according to the IOTA protocol. To save memory, we never store addresses on our Light Node, instead we generate them from the seed whenever required. The Bundle Manager creates a structure containing the Bundle Essences of the transactions belonging to the same bundle (each Bundle Essence includes the amount of tokens transferred in the transaction, a deposit/withdrawal address and a user-defined value named Obsolete Tag). The Bundle Manager hashes the Bundle Essences using Kerl and normalizes the output. If the result contains a specific tryte (the M-tryte), the Bundle Manager increments the Obsolete Tag of the first transaction in the bundle and repeats the computation (M-bug). The Diver is in charge of finding a PoW nonce. Our implementation derives from the original PearlDiver algorithm [4] which we have optimized to be executed on our boards. Our Light Node communicates with a Full Node using LwIP and exchanges JSON objects with data and specific API commands via http POST requests; after receiving an answer, it parses the JSON object and extrapolates the requested information.

2.2 Security Aspects of our Implementation

On top of the decentralized security and privacy ensured by the IOTA DLT, we exploit some features provided by STM32 microcontrollers to get additional security to our Light Node. At the first boot our Light Node securely generates the seed using the True Random Number Generator available on the boards we considered. A sector of the Flash memory is reserved for the seed. We exploit Readout Protection (RDP), a security feature that allows to protect the embedded firmware against copy, reverse engineering or dumping using debug tools or code injection in RAM. In our implementation we set RDP to the maximum level, ensuring full protection of the microcontroller from external attackers (debug interfaces are disabled and therefore access to Flash and RAM memories are forbidden). Note that, however, modifications by the internal application are still possible. This allows to have a secure firmware update functionality to update the internal code when the Light Node is on the field. We also exploit Memory Protection Unit (MPU), a security feature that allows to define specific access rights (Executable, Not executable, Read-Write, Read Only, or No Access) for any memory-mapped resource and two execution modes (Privileged or Unprivileged) for processes. In our implementation we set as privileged the processes implemented in the Account Manager and in the secure firmware update and we define the following access rights for such processes: Read Only for the region storing the seed and that defining MPU configuration, Read-Write for Flash interface and Direct Memory Access (DMA) controller. On the contrary, unprivileged processes have No Access to those memory regions. As for Side-Channel Attacks (SCA), our implementation does not need of particular protections. In fact, one-time hash-based signature schemes, such as the Winternitz scheme used in IOTA, have been shown in [6] to be very resilient to SCA.

3 PERFORMANCE EVALUATION

We assess the performances of our Light Node by evaluating the following processes: 1. Preparing and signing a transaction and 2. Computing a PoW nonce. We compare our implementation with those presented in [5]. Results are reported in Table 1. The most computationally expensive step in process 1 is solving the M-bug, as it requires to repeatedly verify the presence of the M-tryte and, if so, increase the Obsolete Tag and repeat the computation. This introduces a significant variance in the latency, which was also noted, but unexplained, in [5]. In fact, the number of repetitions might vary significantly and unpredictably (on average 120 repetitions per bundle in our experiments). As expected, with both the Nucleo-F746ZG (F7) and and the Nucleo-F429ZI (F4) our Light Node performs far better than the CC2650 but also remarkably good compared to the Raspberry PI 3B (RPI). More precisally, when preparing and signing a bundle of level 1 (resp. 2) our implementation on the F7 is 4 (resp. 1.85) times faster than the RPI one, while that on F4 is 2.43 (resp. 1.06) times faster than the RPI one. If we were to scale our results at the frequency of 1200MHz, the implementation on F7 would be 22.61 (resp. 10.27) times faster than the RPI one, while that on F4 16.19 (resp. 7.10) times faster than the RPI one. As for process 2, it is clear that the boards we considered cannot perform a PoW quickly enough for real time applications. As before, to compare directly with the RPI, we can scale our results at the frequency of 1200MHz: this would lead to an indicative time of 58.3s to perform a PoW on the F7 and of 93.4s on the F4, whereas the RPI takes 82.9s on average. In conclusion, we have shown that our Light Node is able to efficiently participate to the IOTA network. The only limitation is determined by the PoW which is too time consuming to provide instantaneous transaction issuing. Employing a gateway architecture with a PoW proxy server or outsourcing the PoW to a Full Node might be viable solutions to this issue.

REFERENCES

[1] https://docs.iota.org
[2] https://github.com/iotaledger
[3] http://www.st.com/stm32nucleo
[4] https://github.com/Come-from-Beyond/PearlDiver
[5] A. Elsts, E. Mitskas and G. Oikonomou, *Distributed Ledger Technology and the Internet of Things: A Feasibility Study.* BlockSys, 2018.
[6] M.J. Kannwischer, A. Genet, D. Butin, J. Kramer, and J. Buchmann, *Differential power analysis of XMSS and SPHINCS.* COSADE, 2018.