AlkylVM: A Virtual Machine for Smart Contract Blockchain Connected Internet of Things

Joshua Ellul
Department of Computer Science
University of Malta, Malta
joshua.ellul@um.edu.mt

Gordon J. Pace
Department of Computer Science
University of Malta, Malta
gordon.pace@um.edu.mt

***Abstract*—Blockchain technology and the application of smart contracts allow for automation of verifiable digital processes between any number of parties. The Internet of Things (IoT) has seen great potential in the past decade to revolutionise our day-to-day lives with the aim of automating physical processes by incorporating Internet-connected devices into commodities. By integrating the IoT with blockchain systems and smart contracts it is possible to provide verifiable automation of physical processes involving different parties. The challenge lies in that due to resource constraints, many of the computational devices used within the IoT are not capable of directly interacting with blockchain implementations. In this paper, we describe and give a reference design and implementation of a split-virtual machine, AlkylVM, which allows for resource constrained IoT devices to interact with blockchain systems.**

***Index Terms*—Blockchain, Internet of Things, Smart contracts, Virtual Machines, Distributed ledgers.**

I. Introduction

Smart contracts built on top of blockchain technology has piqued the interest of various industries and stakeholders, resulting in the technology being adopted due to the advantages which it brings, by allowing for the automation of verifiable and enforceable digital processes between the parties involved. In the past decade, the Internet of Things (IoT) has seen a similar surge in interest due to the automation benefits that the IoT facilitates. By integrating the IoT with smart contracts and blockchain technology, besides the plethora of applications that would be possible [1], one can enable verifiable and enforceable automation of and arbitration between physical processes. For example, consider the use case of a landlord that would like to: (i) automate the process of providing access to property and other commodities; (ii) monitor tenants' usage of commodities and services; and (iii) automate any errands and tasks for the upkeep of the property. It would be beneficial for the landlord to define smart contracts with the different parties involved to ensure that they guarantee to keep their end of the agreement. For instance, a contract may state that if any rent due is not paid on time, then the tenants are automatically locked out. This requires the smart contract to interact with the IoT devices involved (e.g. a smart lock). Furthermore, the landlord and different tenants may, from time to time, also want to change certain functionality — for example, the landlord may want to implement a property-wide lock-out for a given amount of time on all smart locks if a number of access attempts are invalid. The solution should thus allow for the end IoT logic to be agreed upon to be defined within the smart contract, to reprogram without any manual intervention on the IoT devices when the smart contract comes in to play. Such a use case could become a reality once IoT devices are integrated with blockchain systems and provide reprogrammability based upon the contents of smart contracts.

In this paper, we present a general solution for the combination of these two technologies, and use this property management use case as a running example to illustrate this solution. One of the major challenges is that many devices used within the Internet of Things are resource constrained and are not capable of interacting directly with blockchain implementations (such as Ethereum). Also, given that use cases exist where IoT devices will be required to perform different tasks based upon the smart contracts that they will be required to fulfil, then some form of IoT application updating (based upon the smart contract) must be provided.

We propose a general solution to this challenge, and give a reference design architecture and implementation of a split-virtual machine (where the implementation of more computationally expensive operations are offloaded to resource rich devices), AlkyVM, which enables the integration of IoT devices (both resource-constrained and resource-rich alike) with blockchain technology. The virtual machine, AlkylVM, proposed within this paper specifically enables the integration of IoT devices with the Ethereum blockchain, however the architecture and concepts can be applied on any blockchain infrastructure which supports smart contracts. To the best of our knowledge, this is the first approach proposed to achieve this goal. The architecture and virtual machine naming were inspired by the organic chemical compound structure that makes up Ethers.

II. System Architecture

One of the major challenges in having IoT devices interact with a blockchain system, is one of constrained resources on the devices. In various other IoT contexts, one of the solutions frequently adopted to overcome executing computationally expensive operations is that of using a split-virtual machine architecture [2], in which the virtual machine is split into different components which operate on different devices. We are proposing to use such a solution for connecting IoT devices

978-1-5386-3662-6/18/$31.00 ©2018 IEEE

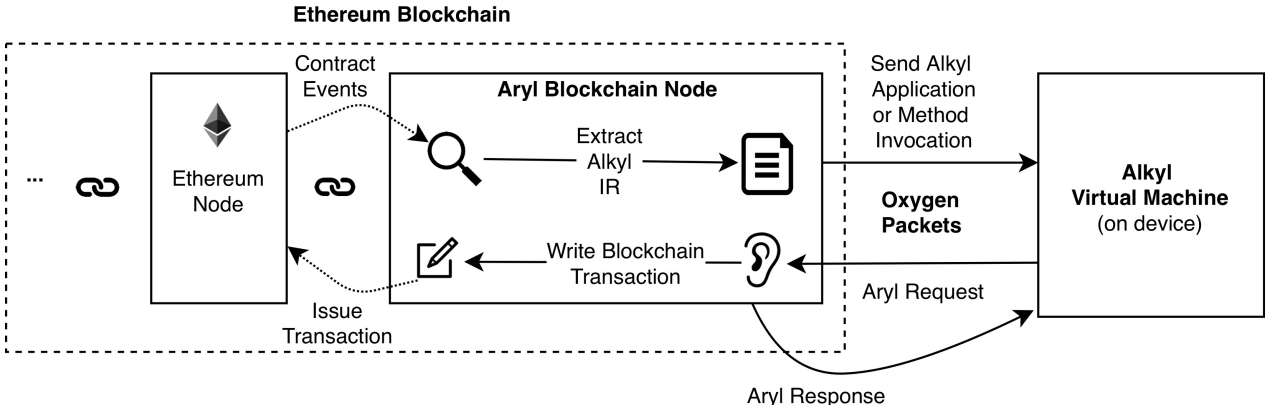

Fig. 1. System overview

with a blockchain. An overview of the proposed system architecture is depicted in Figure 1. The Aryl Blockchain Node acts as a gateway between the blockchain and the connected IoT device/s. Each blockchain connected IoT device would run an instance of the Alkyl Virtual Machine, which allows for blockchain programmability of end IoT devices by communicating with the Aryl blockchain node. Communication between the Aryl blockchain node and end IoT device Alkyl virtual machines is required to be a trusted one. Following is further internal detail of each system component.

A. Aryl Blockchain Node

The Aryl Blockchain Node connects to the Ethereum blockchain just like any other blockchain node (and should be considered to be part of the Ethereum blockchain just like any other Ethereum node). The Aryl node operating on a device that is capable of interacting with the blockchain is responsible for monitoring smart contract transactions and events that would require interaction with the IoT connected device/s or infrastructure (this could be in the form of monitoring a specific smart contract, though not limited to a single smart contract). Upon encountering a change within the blockchain that would require interaction with the IoT system, the Aryl node will extract the application logic required to execute on the end IoT device/s and transmit it to them accordingly.

The Aryl blockchain node is also responsible for listening to the AlkylVM devices that it is servicing for any requests in relation to the blockchain, system-wide state, or execution of computationally expensive operations that the connected device/s may not be able to perform. When a blockchain write request is received from an AlkylVM being serviced (that is a particular IoT device would like to change the state of the blockchain), the Aryl node will translate and write the request into the blockchain (as any other standard blockchain node would). When a read request is required then the Aryl blockchain node will only be required to look at the local state of the blockchain and return the requested result to the IoT device which made the request.

For the landlord property management use case previously described, the Aryl blockchain node would listen to the blockchain for any new smart contracts (i.e. agreements) which concern the property. For example, upon a receiving new tenant agreement, the Aryl Blockchain Node would extract any specific IoT device code for functionality that has been agreed upon and deliver it to the end IoT devices for installation. The Aryl node would also listen to devices for any request that should update either the blockchain smart contract state, or the local IoT device system state. For example, if the agreed functionality was to lock-out all smart locks after a number of invalids attempts, then a smart lock that encounters such activity would alert the Aryl node so that it can propagate the lock-out messages throughout the property's smart locks.

B. Alkyl Virtual Machine

The Alkyl virtual machine and run-time system will await instructions from the Aryl blockchain node. This does not preclude the end IoT device from performing other operations while it is awaiting instructions. Upon receiving an application, the AlkylVM will execute the instructions accordingly. Whenever the executing IoT application requires to: (i) verify or modify any state from the blockchain or IoT system-wide state, for example to check and deduct from a tenant's utility account balance or to check if access to a particular area is still granted; or (ii) to perform computation beyond its capabilities; then a request will be made to the Aryl blockchain node. In this regard, the Aryl blockchain node is treated as a trusted node from the perspective of the connected AlkylVM IoT device/s. The architecture proposed here is not only relevant for resource constrained blockchain connected devices, however the same abstraction could also be used within an IoT device that has the capabilities to act as a blockchain node itself whereby the IoT device will contain both the Aryl blockchain node and the Alkyl virtual machine. The VM has been implemented for MSP430 and AVR microcontrollers, and also for more powerful Raspberry Pi and Intel Edison platforms. Internal VM implementation details follow.

1) AlkylVM Software Stack: Alkyl applications encoded in the Alkyl Intermediate Representation (IR) received from the Aryl Blockchain Node are executed on the Alkyl Virtual Machine (VM). When an application is required to make use of any IoT device specific functionality, the application will make calls via the VM to the run-time libraries by means of the provided Application Programming Interface (API). The VM and run-time libraries will run on top of the available operating system (if one is available) or on the bare-metal, and facilitate interaction with the device drivers or hardware abstraction layer (HAL) where necessary. Low-level driver and internal operating system code may initiate events that are handled within the high-level IoT application code. For example when a key is pressed on a key pad, the low level drivers would raise the event to the run-time library which would then pass on the event to the Alkyl application if it has registered to receive such events. However, there is an endless number of such events and thus the application code must be aware of which events the driver code exposes via the Service Provider Interface (SPI).

2) Intermediate Representation: It would be ideal to encode within the same smart contract deployed on the blockchain not only the in-blockchain smart contract logic, however also the off-blockchain IoT device process that must be executed. This removes the requirement of having to keep track of which off-blockchain processes to execute for different smart contracts that the IoT devices will be interacting with. Therefore, it was decided to use an EVM-like intermediate representation. This could also be useful for an eventual language that provides for both smart contract logic and IoT device execution within the same smart contract sources. The Ethereum Virtual Machine (EVM) is a stack based machine with a 256-bit word size, that is each instruction operates on 256-bit operands and stack items take up 256-bits. The main design decision behind this was *"to facilitate the Keccak-256 hash scheme and elliptic-curve computations"* [3].

Most IoT devices utilise architectures that have an 8, 16 or 32-bit word size. Typical, with most IoT application code consisting of 16 and 32-bit operations. Therefore, the 256-bit operations from Ethereum will incur a heavy execution overhead due to the lower bit-width operations which can be executed on the devices. Supporting 256 bit operations will incur an extensive computational burden, as can be seen from the table below which provides the number of clock cycles required to execute various 8, 16, 32 and 64 bit operations, as well as an implementation that supports big numbers (including 256 bits)[1]:

Operation	Clock Cycles				
	8bit	16bit	32bit	64bit	Big Number[2]
Add	9	16	28	72	1159
Compare	12	21	37	90	298
Multiply	12	24	103	368	2666
Assignment	3	6	12	17	1077

AlkylVM IR provides 8, 16, 32 and 64-bit operations by providing bit mode instructions that specify a code block's word size. Higher bit width word sizes can also be supported. Code blocks that specify their word size must be delineated in the code. This is to ensure ease of static analysis of the IR and also to enable ahead-of-time compilation for efficient execution on the resource constrained devices [4].

AlkylVM supports Ethereum's: stop, arithmetic, comparison, bitwise logic, stack manipulation, memory, storage and flow operations [3] (interaction with the blockchain is provided through the Aryl Blockchain Node). Since AlkylVM supports a variable-sized stack width, and type conversion of stack elements is required, then type information must be preserved within the stack elements. This can be implemented either by associating a type identifier byte with each element, or else could be implemented by using a variable length encoding similar to PrefixVarint[3] or LEB128 (used to encode operands in the Dalvik Executable Format[4] and WebAssembly's binary encoding[5]). The choice on which variable stack size implementation to use is left as an implementation detail as it does not affect the VM IR specification. We have decided to implement a variable length encoding similar to PrefixVarint since the bit size of the type can be deduced based on the least-significant bits of the value, rather than having to check each byte's least-significant bit, allowing for more optimisations.

III. Programming Model

The IoT device VMs where code is expected to execute are off-blockchain, contrary to execution of smart contract code performed on the collective blockchain. The IoT devices can perform actions based on operations initiated from the blockchain and thereafter can follow up by initiating transactions on the blockchain. It would be ideal to allow for the smart contract itself to define what operations should be performed on the IoT devices once a transaction is executed. It would not be sufficient to have the application logic predefined within the IoT systems and therefore the virtual machine approach being presented herein would allow for such smart contract defined IoT device behaviour.

Smart contracts are written as they normally would be. Events that trigger actions on the IoT system should be exposed in the smart contract. The Aryl blockchain node monitors for these events and executes the respective Aryl coordinating code (on the Aryl blockchain node itself). Going back to the property-management use case, consider a smart contract that allows tenants to rent property and automatically get access to all of the premises' entrances using a pin which they have passed (in an encrypted manner) along with their payment (similar to smart lock systems like slock.it[6]). The following Aryl blockchain node script would take care of setting the pin on each connected IoT smart lock within the system, valid for the specified number of minutes:

```
On PaymentEvent(string encryptPin, uint32_t mins) {
 string pin = Decrypt(encryptPin);
```

[1]https://github.com/nickgammon/BigNumber

[3]https://github.com/WebAssembly/design/issues/601

[4]https://source.android.com/devices/tech/dalvik/dex-format

[5]http://webassembly.org/

[6]https://slock.it

```
 for each (device in devices) {
  device.SetPin(pin, block.timestamp + (mins * 60));
 }
}
```

The Aryl blockchain node script is concerned with coordinating of actions between the blockchain and the different connected IoT devices. The Aryl blockchain node script currently supports commands to coordinate each connected IoT device or named IoT devices. That being said, the focus of this paper is on the AlkylVM, that is the virtual machine residing on the IoT connected device.

Applications for the end IoT devices are written in Alkyl, a C-like dialect which compiles down to the platform-independent Alkyl IR which is then executed from within the VM. The language requires that variables are strongly typed (since the IR is also strongly typed). An implementation of the end IoT device smart lock Alkyl application follows:

```
char* pin;
time_t expires;
uint8_t index;
bool valid;

public void SetPin(char* pin, time_t expires) {
  this.pin = pin;
  this.expires = expires;
  this.valid = true;
}

deviceevent void KeyPressed(char key) {
  if (pin[index] != key) {
    valid = false;
  }
  index++;
  if (index == 4) {
    index = 0;
    if (valid) {
      SystemCall(Unlock);
    } else {
      SystemCall(IncorrectBeep);
      valid = true;
    }
  }
}
```

The underlying IoT device firmware would make use of the Alkyl run-time library to call the `KeyPressed` AlkylVM application event any time a key is pressed on the device. This particular implementation allows a user to try to enter the correct pin any number of times, and will unlock the door once the correct pin is entered.

The smart contract, Aryl blockchain node coordinating application, and the end IoT Alkyl application will ultimately be compiled and fused into a single smart contract, whereby the smart contract logic is unaffected by the inclusion of the Aryl and Alkyl applications, and yet the Aryl and Alkyl IR instructions are embedded within the smart contract Ethereum bytecode. By doing so, the involved parties can not only agree to the smart contract logic, but also to what should be executed on the blockchain connected IoT systems. The embedding of the IoT system code within the smart contract coupled with the end IoT system VMs also allows for IoT application code to be developed for the specific smart contract without requiring manual updating of the end IoT devices.

IV. Related Work

The closest related work to enabling automated physical processes based on blockchain smart contract content on resource constrained IoT devices include the following: defining virtual resources within firmware on IoT devices which can thereafter be instructed to download a sequence of function invocations [5]; and using a blockchain as a means of storing data from IoT devices and also to store configuration properties [6]. Other tools exist that allow for the orchestration of resource constrained IoT devices such as Node-RED[7] which allows a master controller to send commands to connected Arduino devices (amongst other devices). Such an approach differs from ours in that Node-RED requires to send each individual command which is there by executed immediately on the device, whereas our approach sends the code to be executed and the code is thereafter executed (without requiring any further orchestration from a master controller).

V. Conclusions

In this paper we have presented AlkylVM, a split-virtual machine architecture to enable the integration of resource constrained (and resource rich) devices with blockchain systems, in which an implementation of the virtual machine for the Ethereum blockchain network has been described. The programming model presented herein provided a decomposition of each of the components within the system. In future work we will be looking into programming models that may be better suited towards the requirements of such a system involving heterogeneous distributed systems and paradigms.

By integrating IoT devices with blockchain systems, it would be possible to automate physical processes initiated by verifiable blockchain transactions. It would be beneficial to also provide guarantees to parties that the end IoT devices have executed their obligation as required. Therefore, we are looking into means of providing such guarantees of physical process execution back to other involved parties.

Acknowledgement: This short paper has been accepted in Blockchains and Smart Contracts workshop (BSC'2018).

References

[1] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[2] D. Simon, C. Cifuentes, D. Cleal, J. Daniels, and D. White, "Java™on the bare metal of wireless sensor devices: The squawk java virtual machine," in *Proceedings of the 2Nd International Conference on Virtual Execution Environments*, ser. VEE '06. New York, NY, USA: ACM, 2006, pp. 78–88. [Online]. Available: http://doi.acm.org/10.1145/1134760.1134773

[3] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, 2014.

[4] J. Ellul and K. Martinez, "Run-time compilation of bytecode in sensor networks," in *Sensor Technologies and Applications (SENSORCOMM), 2010 Fourth International Conference on*. IEEE, 2010, pp. 133–138.

[5] M. Samaniego and R. Deters, "Hosting virtual iot resources on edge-hosts with blockchain," in *2016 IEEE International Conference on Computer and Information Technology (CIT)*, Dec 2016, pp. 116–119.

[6] S. Huh, S. Cho, and S. Kim, "Managing iot devices using blockchain platform," in *2017 19th International Conference on Advanced Communication Technology (ICACT)*, Feb 2017, pp. 464–467.

[7] https://nodered.org/