Deploying Blockchain Smart Contracts Using XQuery Language

Ahmad Mousa Altamimi  
Computer Science Department  
Applied Science Private University, Amman, Jordan  
a_altamimi@asu.edu.jo

Abstract- Blockchain technology provides a new decentralized approach for archiving digital resources. It has huge potential to enhance applications in many industries such as finance, healthcare, media, and education. This is supported by the Blockchain key features like data protection and smart contracts. However, although smart contracts present great opportunities for a range of domains, there are several problems related to their designing and implementation. Specifically, smart contracts assume ideal defined, but this is not always the case. If one step of contract designing is executed incorrectly, the smart contract will be implemented wrongly. Because the content of blocks can be expressed in any format, for example, in XML, smart contracts can be defined and deployed using XQuery Language. XQuery makes data contained in the instance available to other applications. In this paper, we proposed a security model that utilizes the XQuery syntax to define smart contracts. The defined contracts implicitly specify the rules on the Blockchain resources and are integrated into XML schema itself, which can be applied to instance objects without any change. To ground our conceptual idea, a prototype implementation is specifically developed for this research. The developed system consists of three main components while the User Interface Tool allows end-users to design the contracts using the XQuery with SQL-like syntax. The Smart Contracts Repository stores the contracts generated by the interface tool to be applied later by the Contracts Applier to the Blockchain for enforcement. We found that the specified contracts can support complex rules over the resources when combining and ends up with reusable contracts, which efficiently simplify the security administration and achieve access control over the Blockchain blocks.

Keywords – Blockchain, Smart Contract, XML, XQuery language

I. INTRODUCTION

Blockchain technology is a decentralized, shared database that forms a new way of storing data over the internet [1]. The stored data on a blockchain is represented as immutable time-stamped series of records and can take any form, transactions, identities, medical records, or any data type that stores in a block [2-4]. Each block in the chain represents a package data structure and consists of clusters data for incorporation in the Blockchain. The Blockchain is started by the genesis block, which represents the first block in the chain and cannot be modified. It represents the universal secure parent of all the blocks to build a trusted blockchain [5]. The Blockchain is expanded by appending and linking blocks together, forming a chain of blocks through a process called hashing. The hash algorithm (e.g., SHA256 cryptographic) is running to produce a fixed-length size hash code from any arbitrary blockchain content size [6]. The generated code is then placed in the block header and used to link the block with the former one. While producing such a hash should be easy for any arbitrary input, it should be tough to retrieve back the original text based on the output hash. Additionally, any changes in the original input data should result in extensive and uncorrelated modifications to the output hash [5].

No one controls blockchain blocks as it is duplicated (distributed) over a cluster of nodes or users and managed by them. This distributed approach has a significant advantage as a user can view or edit the chain from anywhere if the user has the right cryptography keys. Moreover, it ensures the network security as it is computationally impractical to launch an attack against the blockchain network, where 51% of the systems blocks must be compromised to surpass the hashing power of the target network [7]. To regulate the interaction between Blockchain parties, smart contracts, a self-executed code written in an acceptable language such as Solidity, are used to establish a set of rules on which the parties undergoing agreement agree to interact with each other. Specifically, smart contracts help in developing and managing authorization control as it provides the necessary requirements to be fulfilled before granting access to the recognized personnel data [8]. So, whenever these requirements/rules are met, the underlying agreement is enforced, which forms a means of decentralized automation.
To ensure the authenticity of the contract, Blockchain copies the contact to a large number of nodes on the network. Thus, every node on that network acts as an automatic witness to the contract and tells the other nodes to look for evidence that the contract's terms were fulfilled. Moreover, by decentralizing the verification of contract terms, it almost impossible for contractual partners to cheat one another. So, individuals can transact anything of value, which solves the problem of trust in conditional transactions [9].

Besides on the features mentioned above, it also offers numerous advantages. They capable of tracking performance in real-time, consequently lower the cost of transactions, guarantee more security, and help turn legal obligations into automated processes [10]. Through the use of smart contracts, it makes Blockchain platforms so attractive and brings a revolutionary change across many industries such as finance, healthcare, media, and education [3-4].

According to [11], Blockchain technology will be available for adoption on a large scale that will apply to these industries. Being said that, although smart contracts present great opportunities for a range of domains, there are several problems related to their regulation and implementation. The first one is a legal problem relates to the enforceability of smart contracts. Strictly speaking, Blockchain is a decentralized and permissionless system that can be used to conduct business transactions between multiple regions or countries speedily. However, the current resolution of contracts is differed between countries and is settled in courts. This problem is out of our scope because it relates to legal enforcement nature, which involves notaries, courts, and lawyers [12].

The second problem is the general anxiety related to the smart contracts coding errors or incorrect design that may result in system malfunctions. Precisely, the Smart contract process consists of four main steps: Firstly, the contract Predefining step that concerns the establishment of all parties and the conditions for execution (time or date). Secondly, setting the events step that triggers the implementation of the contract. Thirdly, the contract execution step, which results in executing the contract based on the conditions that have been met. Finally, the Settlement step, which ends in settling the contract content [12].

Smart contracts assume ideal defined; however, this is not always the case. If one or more of the described steps are executed incorrectly, the smart contract will be implemented wrongly. Because the content of blocks can be expressed in any format, for example, in XML or JSON for applications [13], smart contracts can be defined and deployed using XQuery Language. XQuery makes data contained in the instance available to other applications [14].

In this paper, we proposed a security model that utilizes the XQuery syntax to define smart contracts. The defined contracts implicitly specify the rules on the Blockchain resources and are integrated into XML schema itself, which can be applied to instance objects without any change. Moreover, the specified contracts can support complex rules over the resources when combining and ends up with reusable contracts, which efficiently simplify the security administration and achieve access control over the Blockchain blocks. One can summarize the benefits of this approach as follow:

- Support Standardization: This reduces the costs of negotiations and agreements when executed.
- Facilitate automation: which reduces transaction times and unnecessary manual processes.
- Enable Reliance: schemes support well-designed smart contracts that facilitate the automatically execution, thereby settlement risk will be reduced.
- Provide Innovation: by automating the flow of digital assets, new products may be foster and tweak the code to create new versions of the contract for new businesses.
- Achieve security: it allows policy designers to identify security and privacy constructs, taking into account the requirements of Blockchain technology.

To ground our conceptual idea, a prototype implementation is explicitly developed for this research. It consists of three main components: the first component is the User Interface Tool, which allows end-users to design the contracts using the XQuery with SQL-like syntax (it can also be implemented using XPath language). The Smart Contracts Repository, which stores the contracts generated by the interface tool. Finally, the Contracts Applier, which retrieves the contracts from the repository and applies them to the Blockchain for enforcement. More details about the prototype are given in Section IV.

The rest of this paper is organized as follows. Section II discusses the terminology relevant to our approach. A detailed discussion of the proposed model is given in Section III. Finally, conclusions and future work are offered in Section IV.
II. Primarily Material

In this section, the basic terminology relevant to Blockchain technology, Smart Contracts, XML Schema, and XQuery is briefly discussed from a technical, standard, and developer perspective.

2.1. Blockchain –

Blockchain is a series of linked blocks/data containers that are stored data for incorporation in a public ledger [15]. There are two types of Blockchain: The Public Blockchain, where the chain is opened to anyone to participate and see all the stored data. This can be helpful in applications with no sensitive or private data. In contrast, the Private Blockchain has permissions to join the chain, and the stored data can be seen by only the authorized users [16]. Every Blockchain has a secure starting point, a “root” (known as the genesis block) from which to build a trusted blockchain. It is hash, structure, and all the computers that use the Blockchain recognize the created time. Each block within a Blockchain is composed of two parts: the header, which includes metadata, accompanied by a lengthy record that advances its size, and the body that stores the data itself. The block is identified by an identifier, a primitive identifier that identifies a block. It is stored in the block header and calculated using a cryptographic hash algorithm. The cryptographic hash is also known as a digital fingerprint that is built by hashing the block header twice through the SHA256 algorithm resulting in the 32-byte hash [6]. The block hash recognizes a block and can be autonomously determined by any node by directly hashing the block header. Figure 1 depicts a simple representation of a Blockchain-based system.

![Figure 1. Simple Blockchain representation](image)

2.2. Smart Contract –

A smart contract is a self-executing code, publicly viewable, and running on top of the Blockchain to facilitate the verification and enforcement of the negotiation of the agreement between different parties. It ensures that invalid transactions are not added to the Blockchain and is composed of a set of predefined rules that regulate how the involved parties can interact with each other [17]. When these rules are met, the agreement is automatically imposed. This is done automatically in a decentralized fashion. For example, consider two parties who agree to the terms of a transaction and create a smart contract to fulfill that transaction. The authenticity of the contract is ensured in Blockchain by duplicating it to a large number of computers on a network. In turn, every device on that network acts as an automatic witness to the contract like a notary public works [9]. As compared with existing available contracts, a smart contract is faster as well as it also decreases the time for executing and deploying the data. Since it a decentralized system, no one can act as another in the system. Thus, hackers cannot be able to modify or edit the document without getting the right permissions. Moreover, the smart contract is considered as more cost-efficient because it will use fewer resources and eliminate the additional costs. Smart contracts are more trustworthy by having the property of immutable, which means no one can modify/edit or delete the data without meeting the permissions, and this can be automatically encrypted the rules. In short, the smart contract simplifies the transaction happening in the Blockchain and makes it easier to perform [18,19]. For example, assume the Smart contract in healthcare, where managing patients’ health information, records, and data has become a large task for many practitioners. This issue cannot be solved by practitioners alone. This is where smart contracts come in.
2.3. XML Document and XML Schema –

The growth of popularity using XML documents raises the need for employing XML schemas for describing the stored data structure [20]. While the XML is only containing data in a suitable format and structure. The XML schema contains grammars to validate the structure of XML documents. Another great strength about XML Schemas is that they are written in XML. This means that XML Schemas are extensible, which facilitates the reusing of one schema in other schemas, the reacting of own data types derived from the standard types and referencing multiple schemas in the same document [21]. For example, assume the XML document shown in Figure 2. a, which utilizes different data types to build and define the Blockchain depicted in Figure 1. Two main categories of data types have been used: the simple and the complex types. While the simple types are used to define single elements (e.g., string, integer, URL, date, etc.), the complex types are used to define reusable clusters of elements. These elements are defined by their names and types and represented with the parent-child relationship. Off course, all of the XML elements may have an id attribute.

![XML Document and XML Schema](image)

Figure 2. A simple XML document and its schema

Figure 2. b depicts the corresponding XML schema for the XML document shown in Figure 2. a. The schema starts with the XML declaration, followed by the root element. The schema elements may be simple or properly nested and must be closed. The schema can be validated after parsing it and used just like normal XML instance because it is an XML-like document. In addition, it is understandable as the sender describes the data as the receiver will read. Finally, a language like XQuery can be used to make data contained in the instance available to other applications.

2.4. XQuery language –

XQuery is a functional query language built on XPath expressions and used to retrieve information stored in XML format [22]. It was designed to query XML data in the same manner of SQL is for databases. XQuery is powerful and easy language replaced complex Java or C++ programs with a few lines of code. It distinguishes itself to other query languages by many features such as its simple type system with expression-oriented programming language capabilities [23]. So, it is best for XML-based databases and object-based databases. Moreover, the XQuery language can be used to retrieve both hierarchal and tabular data and also query tree and graphical structures. For example, recall the XML document depicted in Figure 2. a. The XQuery code shown in Figure 3 displays the header hash value for all blocks.
Blockchain blocks can be modeled in any format (e.g., using XML documents). Here, we assume that these resources are defined by a set of schemes, and the rules of the contract are defined using XQuery. Specifically, the proposed model defines the rules on schema objects themselves. Then these rules will be implicitly specified directly on Blockchain resources and can be combined together to support complex rules over Blockchain resources. At the same time, because schema elements support rich relationships between them or between schemes themselves, the generated complex rules can be reused. Moreover, rules can be transparently and consistently propagated to all relevant resources before sharing it with different users. Based on these features, this model would employ efficient smart contracts that can, in turn, simplify the administration overhead.

3.1. The Model integration –
External users can exchange resources over the network. This is done by using the Hypertext Transfer Protocol (HTTP), which is the best way to communicate between applications because it is supported by all Internet browsers and servers. Also, one can consider the SOAP (Simple Object Access Protocol) message [24], which is a messaging protocol used to encapsulate data like messages and exchange them via HTTP [25]. That is, a SOAP message is an ordinary XML document contains a differentiating element that identifies the resource, a header, and body elements, which contain the requested information. Our proposed model exploits SOAP messaging for sending Blockchain and their contracts or schemes. These messages were designed so that they can be tunnelled over HTTP. This would and did help in its rapid adoption. Because the infrastructure of HTTP is already in place, users would not have to spend extra money on another kind of implementation. Instead, they can expose and access web services using technology already deployed. In addition, the integration allows us to easily be transferring the Blockchain in secured mode, which provides a way to communicate between different applications with different technologies running on different systems. The success of this integration demonstrates that the general principles behind our model are broadly applicable to any standards resource, regardless of the content of the resource.

3.2. The Model components –
The model consists of three major components. The following subsections discuss these components and provide a brief description of their Functionalities:

1. The User Interface Tool: This tool enables users to design contracts. It aims to hide unnecessary details and facilitates the using of resources elements in a simple way. The tool uses either the XPath or the XQuery with SQL-like syntax. Here, all the contract parties and conditions are defined (e.g., time or date), which perform templates, or blueprints to define the rules for Blockchain resources. An equivalent number to the Smart contracts should be created of schemes, each of which defines the rules for each resource. For example, Figure 4 depicts a simple contract written in Solidity language for banking exchange that is used to send coins to a partner incorporates and has access to the system. The same contract defined using XQuery language and shown in Figure 5.
The code starts with the line `address public minter` declares a state variable of type address that is publicly accessible. It is used for storing addresses of contracts to external persons. The constructor function is `Coin`, which is run during the creation of the contract and permanently stores the address of the person who creates the contract. The rest functions (mint and send) are two functions that actually end up with the contract. They are called by users and contracts alike. The mint function is called by anyone; however, nothing will happen if it calls by the account that created the contract. Ultimately, to send coins to someone else, the send function is used by anyone who already has some of these coins.

2. The Smart Contracts Repository: After defining the contracts, they are stored in the repository to be used later by the Contracts Applier. These contracts collectively represent the security metadata. Ultimately, administrators interact with the repository using front-end tools that are typically a graphical, interactive interface.

3. The Contracts Applier: The last and the most significant component is the Contracts Applier. It is a bidirectional component, which retrieves a specific contract from the repository then sets the events that trigger the implementation of the contract before integrated it in the Blockchain. After outsourcing the Blockchain and if the contract rules are met, the contract is executed, which results in executing the contract based on the conditions that have been met. After that, the contract content is settling in the settlement step.
3.3. Deploying a smart contract —
In fact, Nodes do not understand what a smart contract is. Instead, a smart contract is a piece of code. These smart contracts will have to be compiled, resulting in bytecodes that are understood by EVM (Ethereum Virtual Machine) [26]. Of course, if the smart contract has syntax errors, it will not generate bytecode in the compiler. Or if bytecode contains wrong opcodes unknown to the EVM, the transaction will get failed. In the case of successful compiling, the bytecode is sent to the chain as a transaction, which results in creating a contract instance in the chain. Once the contract is deployed or distributed, the state of the Blockchain will be changed according to the contract variables [27]. For example, recall the contract depicted in Figure 2, after deploying the contract the send function will be invoked resulting in transferring the amount from the sender balance to the receiver balance (modifying the account balances information in the Blockchain) as shown below.

```
function send (address receiver, uint amount) public {
    balances[msg.sender] -= amount;
    balances[receiver] += amount;
}
```

In Ethereum [28], the account balance is maintained by State trie of the Blockchain. So, when we are deploying a smart contract, the bytecode passed as part of the transaction will be saved in the state trie as a part of the account created for the deployed contract. In fact, the contracts will not be found inside blocks but transactions. It can contain some contract bytecode that will be executed by the EVM and changes the state in the Blockchain. So, nodes only execute those bytecodes, and the transaction will be executed successfully, or it will fail.

IV. CONCLUSION AND FUTURE WORK

The big revolution in using Blockchain technology is supported by the facilities that such technology provides, like distribution, consensuses, encryption, and virtualization. On top of the Blockchain, smart contracts are placed to facilitate the verification and enforcement of the negotiation between different parties. However, this sort of "traditional" coding is very error-prone since smart contracts assume ideal defined, which is not always the case. If the smart contract is defined correctly, it will be implemented wrongly and may result in system malfunctions. In this paper, we present a security model that defines and deploys smart contracts on the level of XML schemes using the XQuery language.

The model consists of three main components, which are the User Interface Tool that allows end-users to design the contracts in terms of all parties and the conditions for execution. The Smart Contracts Repository, which stores the contracts generated by the interface tool. The Contracts Applier, which retrieves the contracts from the repository and setting the events that trigger the implementation of the contract before adding it to the Blockchain for enforcement. These contracts are implicitly mapping to the Blockchain resources and can then be combined to support complex rules over Blockchain resources. This will result in reusable contracts, which efficiently simplify the administration overhead and, at the same time, generate complex rules over the Blockchain.

As future work, we aim to extend our model to cover different domains that may require additional processing before outsourcing Blockchain as it gains much attention from both academic researchers and industrial practitioners. One can consider some promising application areas for the deployment of Blockchain like education, government, healthcare, or any digital document exchange domains. The model should satisfy the required security measures for Blockchain resources.

ACKNOWLEDGEMENTS

The authors are grateful to the Applied Science Private University, Amman-Jordan, for the full financial support granted to cover the publication fee of this research article.

REFERENCES