Combining ID's, Attributes, and Policies in Hyperledger Fabric

Daan Gordijn[1], Roland Kromes[1(✉)], Thanassis Giannetsos[2], and Kaitai Liang[1]

[1] Cyber Security Group, Delft University of Technology, Delft, The Netherlands
D.A.Gordijn@student.tudelft.nl, {R.G.Kromes,Kaitai.Liang}@tudelft.nl
[2] Ubitech Ltd., Digital Security and Trusted Computing Group, Athens, Greece
agiannetsos@ubitech.eu

Abstract. This work aims to provide a more secure access control in Hyperledger Fabric blockchain by combining multiple IDs, attributes, and policies with the components that regulate access control. The access control system currently used by Hyperledger Fabric is first completely analyzed. Next, a new implementation is proposed that builds upon the existing solution but provides users and developers with easier ways to make access control decisions based on combinations of multiple ID's, attributes, and policies. Our proposed implementation encapsulates the Fabric CA client to facilitate attribute addition and simplify the process of registering and enrolling a newly created certificate (corresponding to a new user). This research, concludes that it is possible to combine multiple ID's, attributes, and policies with the help of Hyperledger Fabric's smart contract technology. Furthermore, it could be seen that the performance impact for real-world applications is negligible compared to the insecure case of always providing access to a resource without performing access control.

Keywords: Blockchain · IPFS · Privacy · Security

1 Introduction

Ever since the anonymous Satoshi Nakamoto published his Bitcoin white paper [21] in 2008, blockchain has become one of the most disruptive technologies in the computer science industry. In recent years, many other innovative blockchain technologies have been developed [20], which are becoming increasingly more popular.

While Bitcoin was created to provide a digital alternative to traditional, bank-controlled currencies [17], many of these newer blockchain technologies are designed to provide a platform for building and deploying decentralized applications through the use of smart contracts[1]. By implementing their business

[1] A digital contract written into code that is stored and automatically executed on the nodes of a distributed blockchain network [6].

W. Meng and W. Li (Eds.): BlockTEA 2022, LNICST 498, pp. 32–48, 2023.
https://doi.org/10.1007/978-3-031-31420-9_3

logic within these smart contracts, decentralized applications can automatically execute any transaction without human intervention, making them completely independent and decentralized [28]. Due to the many benefits of decentralized applications [2], the adoption of blockchain technologies has recently expanded to many non-financial applications such as "healthcare, supply chain management, market monitoring, smart energy, and copyright protection" [29].

Most of these traditional blockchain technologies, such as Bitcoin, Ethereum, and Cardano, are so-called "permissionless" blockchain technologies. This type of blockchain technology, however, has many privacy issues when it is being used in the context of enterprise-level applications, as described in [23]. Many alternative, so-called "permissioned" blockchain technologies have been proposed to solve the issues, the most promising of which is Hyperledger Fabric [9]. Through the use of innovative concepts such as channels, policies, identities, and Membership Service Providers, Hyperledger Fabric can determine the identity of participants, perform access control based on these identities, and ensure the privacy of transactions and smart contracts.

As with many technologies, the increase in popularity of blockchain technologies also drives an increase in security threats and attacks. One of the major issues that many blockchain technologies, including Hyperledger Fabric, currently have is providing secure access control to the distributed ledger and smart contracts. Hyperledger Fabric partially addresses this issue by only granting network access upon submission of a valid X.509 certificate [22], issued and approved by a trusted Certificate Authority. However, this type of ID-based access control is not scalable for larger organizations. This paper will therefore investigate how secure access control in Hyperledger Fabric can be improved, in particular by looking into solutions that can combine these ID's with attributes and policies.

The study also highlights the access control components that currently interact within Hyperledger Fabric. This work also provides a an implementation for combining multiple ID's, attributes, and policies be combined within Hyperledger Fabric, and analyzes the performance impact of ID-, attribute-, and policy-based access control.

This paper is structured in the following manner. First, Sect. 2 will provide a summary of the most relevant work that currently exists in literature. Next, Sect. 3 will provide an overview of the contributions made by this research. Then, Sect. 4 will provide a background on the current access control system of Hyperledger Fabric, while Sect. 5 will present the proposed system model that has been implemented as part of this research. Subsequently, Sect. 6 will provide an overview and analysis of the results that have been obtained during the research, while Sect. 7 will provide a brief discussion. Finally, Sect. 8 will present the main conclusions of this research.

2 Related Work

Research into secure access control in various blockchain technologies, including Hyperledger Fabric, has been conducted in multiple papers. Many of these stud-

ies are performed in the context of exploring the integration of blockchain technologies with the Internet of Things (IoT), as blockchain is currently seen as the most promising technique for providing secure access control to IoT devices [3].

In [24], a summary of the major problems of modern access control systems is presented, together with an explanation of how these problems can potentially be solved using blockchain technologies. Furthermore, this paper provides an overview of existing access control studies and describes the current challenges of blockchain-based access control.

In [3], an attribute-based access control scheme for Internet of Things devices is proposed by employing blockchain technology to keep track of the distribution of the attributes. Next, [26] proposes a different scheme that is built upon various smart contracts and so-called "functional modules", which are jointly responsible for managing attribute information and making access control decisions. Finally, [31] proposes yet another access control scheme which is implemented and deployed using the smart contract technology of the Ethereum blockchain network.

While the papers discussed so so far describe general blockchain-based access control systems, other papers make specific use of the Hyperledger Fabric blockchain technology. First, [13,30], and [16] explore basic access control scenarios for IoT devices in Hyperledger Fabric. Next, [32] combines the Hyperledger Fabric blockchain technology with the InterPlanetary File System (IPFS) [15], allowing IoT devices to easily store documents on a distributed file system and store the hashes of these documents on the blockchain ledger. Finally, [1] proposes a multi-layered and multi-model access control system in the context of an agricultural supply chain system that runs on Hyperledger Fabric.

While research into secure access control in Hyperledger Fabric and other blockchain technologies certainly exists, no study into combining multiple ID's, attributes, and policies during the decision-making process has been conducted. To fill in this gap, this paper will propose a new access control scheme that combines these ID's, attributes, and policies within a single smart contract deployed to a Hyperledger Fabric network. For consistency, this research paper will consider the scenario where an IoT device wants to store a document on IPFS and subsequently save the returned document hash on the blockchain network, as also used in [32].

3 Contribution

Using the existing literature from the previous phase, a new design was proposed to provide secure access control in Hyperledger Fabric. As stated in the research question, this design had to combine multiple ID's, attributes, and policies in the decision-making process. Subsequently, during the implementation phase, the design was implemented using a smart contract and deployed to a local Hyperledger Fabric test network, which was set up using the official tutorial [11].

Hyperledger Fabric currently supports three programming languages for the development of smart contracts and client applications: Go, Java, and NodeJS [12]. For each language, several SDK's are available [10] that help make the implementation of smart contracts and client applications easier. For this particular research project, NodeJS with TypeScript has been selected as the toolchain for the implementation phase, as this language is very easy to learn and understand.

The complete repository that contains a basic test network together with the smart contracts and sample applications that have been implemented during this research project is available on GitHub[2]. The `README` stored in this repository also includes a small tutorial, as well as a complete overview of the required tools and their recommended versions.

4 Background

This section discusses the current approach to secure access control in Hyperledger Fabric. This section begins with a brief introduction to Hyperledger Fabric and secure access control in general, and subsequently discusses the main components and methodologies that Hyperledger Fabric currently uses to provide secure access control.

4.1 Hyperledger Fabric

Hyperledger Fabric is an "open-source enterprise-grade permissioned distributed ledger technology (DLT) platform, designed for use in enterprise contexts" [12]. While many well-established blockchain platforms such as Bitcoin and Ethereum are currently being modified to be used in enterprise-grade applications, Hyperledger Fabric has been built around enterprise applications from the beginning. First, Hyperledger Fabric is highly modular, which allows core parts of the blockchain network to be customized. Second, Hyperledger Fabric has support for writing smart contracts in general-purpose languages, including Go, Java, and NodeJS, while most other blockchain technologies require developers to learn new languages, such as Vyper or Solidity in the case of Ethereum [5]. Finally, Hyperledger Fabric is permissioned, which means that the identity of all participants of the network is known and can therefore be verified using access control systems, allowing organizations to establish trust.

Each node in the network maintains a local Membership Service Provider (MSP). These service providers store all X.509 certificates that have been issued by the Certificate Authorities of their corresponding organizations, which are then used by network nodes to map X.509 identities to internal roles. Together with the Certificate Authorities, these providers are therefore responsible for providing the initial layer of identity-based access control.

[2] https://github.com/daangordijn/Fabric-Access-Control.

4.2 Secure Access Control

Access control is "a security technique that regulates who or what can view or use resources in a computing environment" [19]. Different types of access control exist, including Identity-Based Access Control (IBAC), Role-Based Access Control (RBAC), and Attribute-Based Access Control (ABAC) [4]. While older, established blockchain technologies such as Bitcoin and Ethereum are non-permissioned and therefore do not implement these types of access control systems, Hyperledger Fabric is a permissioned blockchain technology, which enforces it to perform access control.

Currently, Hyperledger Fabric employs multiple layers of access control to provide security and privacy within the blockchain network. First, at the most basic level, Hyperledger Fabric uses a simple identity-based access control system, which prevents unauthorized entities from accessing anything on the blockchain network. This layer is explained in more detail in Sects. 4.3 and 4.4 since the purpose of this paper is to extend this simple system to a more complex attribute-based access control system. Second, at an organizational level, Hyperledger Fabric can restrict access to smart contracts and the ledger through the use of channels, as described in Sect. 4.1. By only granting individual organizations access to the minimal required subset of channels, the privacy of smart contracts and ledger states can be preserved.

4.3 Certificate Authorities (CAs)

A Certificate Authority is an "organization that acts to validate the identities of entities and bind them to cryptographic keys through the issuance of electronic documents known as digital certificates" [27]. Hyperledger Fabric provides a special implementation, called the "Fabric Certificate Authority" or "Fabric CA" in short, which can be used to create and sign these digital certificates using the international X.509 standard [14]. Fabric CA consists of both a client-side and server-side command line interface (CLI), called `fabric-ca-client` and `fabric-ca-server`, respectively. Fabric CA provides many features including "registration of identities, issuance of enrollment certificates, and certificate renewal and revocation" [7].

When an administrator wants to enroll a new identity, Fabric CA will generate a key-value pair that consists of a private key and a public key. Together with the parameters provided by the administrator, a Certificate Signing Request (CSR) will be created, which is then processed by Fabric CA.

In Sect. 4.5, this process of registering and enrolling a new identity with the Fabric CA server is visualized. This section will also describe a new command line interface (CLI) that has been implemented as part of this study and makes the creation of new identities much easier.

4.4 Membership Service Providers (MSPs)

A Membership Service Provider is a component within Hyperledger Fabric that can be used by participants of the blockchain network to prove their identity to

other participants of this network. When a user wants to start interacting with a Hyperledger Fabric blockchain network, it needs to create a key pair, which consists of a public key and a private key, which is needed to prove its identity to the rest of the network. Next, this public key must be included in a Certificate Signing Request (CSR), which is then submitted to a Certificate Authority and used to issue a new X.509 certificate. While X.509 certificates, including public keys, can be shared publicly, private keys must always be kept secret to comply with the principles of Public-Key Infrastructure (PKI) [18].

When a participant of the blockchain network now wants to submit a transaction, it needs to create a transaction proposal and sign this proposal using its private key. All nodes on the blockchain network are then able to verify this transaction proposal using the public X.509 certificate of this participant since it is stored inside the Membership Service Providers. Because of this, Membership Service Providers can establish trust on the permissioned blockchain network, without the need of sharing private keys.

4.5 Generating Certificates

In Fig. 1, a simplified version of the process of generating X.509 certificates using Fabric CA is visualized. As can be seen, the Fabric CA Client has to invoke the Fabric CA Server using two commands, `fabric-ca-client register` and `fabric-ca-client enroll` [7]. By doing this, the server will generate a private key, a public key, and a corresponding signed X.509 certificate. This certificate is then automatically stored in the Membership Service Providers that are located on various nodes inside the blockchain network.

While this process of generating X.509 certificates for Hyperledger Fabric is not overly complicated, it can become cumbersome to run multiple commands with many different flags to just create one certificate. Therefore, as part of this research paper, a wrapper around the `fabric-ca-client` was created. This tool, called `certgen`, is publicly available in the GitHub repository (see footnote 4), together with a small tutorial on how to interact with it. The `certgen` tool internally uses the `fabric-ca-client` commands and has the advantage that it can automatically populate a local file system wallet with the correct files which are required to connect a client application to the blockchain network. In addition, since this tool is highly interactive, it makes it much easier for administrators to add attributes to the certificate. More about the importance of setting attributes within X.509 certificates will be explained in Sect. 5.

5 Proposed Implementation

This section discusses the proposed implementation that improves the current implementation of secure access control in Hyperledger Fabric, introduced in Sect. 4. This section begins with a brief discussion of how to independently combine multiple ID's, attributes, and policies, and subsequently presents the final design incorporating these components.

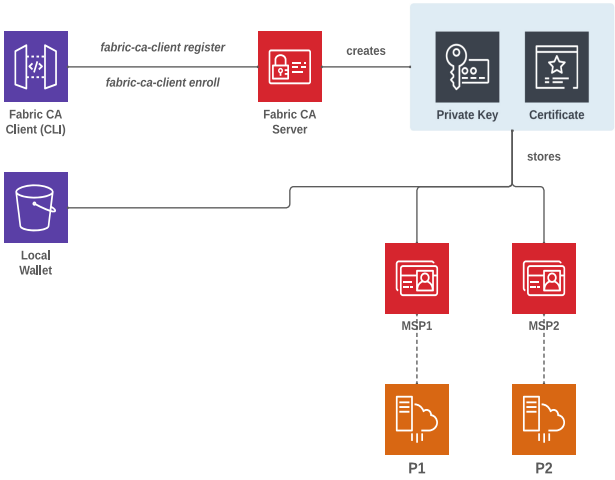

Fig. 1. Current process of enrolling a new identity within a Hyperledger Fabric network. The `fabric-ca-client` CLI is used to run the `register` and `enroll` commands, respectively. Then, the resulting X.509 certificate is stored on a set of peer nodes, while both the X.509 certificate and the corresponding private key are stored in the user's local file system wallet.

5.1 Combining Attributes

In Hyperledger Fabric, every X.509 certificate issued by Fabric CA [7] can have attributes. These attributes can be used during access control to determine whether a client should be given access, or not. To allow for more complex access control decisions, multiple attributes can be combined into so-called "policies", which are visualized in Fig. 2.

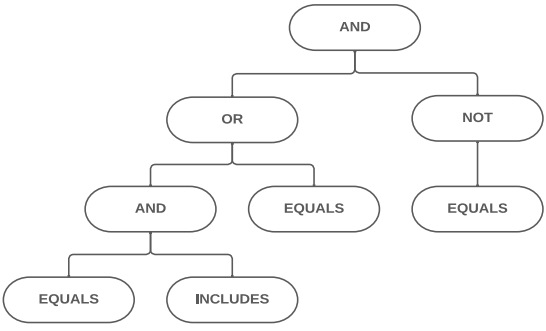

Fig. 2. Combining multiple attributes. The `EQUALS` and `INCLUDES` operators validate whether a specified attribute equals or includes a certain value, respectively. The `AND`, `OR`, and `NOT` boolean operators can be then be applied to combine or negate these individual attribute checks, allowing the client to create complex policies.

For this study, the following boolean operators have been selected that can be used for building access control policies:

- `EQUALS`: Checks whether an attribute is present on the certificate, and whether it is equal to the provided value.
- `INCLUDES`: Checks whether an attribute is present on the certificate, and whether it includes the provided value. This operator can be used when the specified attribute on the certificate has a comma-separated list of strings as its value, which must include a particular value.
- `AND`: Logical operator that combines two or more operator trees. This operator returns `true` if and only if all operator trees combined by this operator evaluate to `true`, and returns `false` otherwise.
- `OR`: Logical operator that combines two or more operator trees. This operator returns `true` if and only if at least one of the operator trees combined by this operator evaluates to `true`, and returns `false` otherwise.
- `NOT`: Logical operator that negates the output of another the given tree. This operator returns `true` if and only if the operator tree provided to this operator evaluates to `false`, and returns `false` otherwise.

Together, these operators can build complex policies that can later be evaluated to determine whether a client has access to a resource on the blockchain network, or not.

5.2 Combining Policies

As described in the previous subsection, an access policy is a rule that enforces an X.509 certificate to possess a particular combination of attributes and values. These access policies can be used in Hyperledger Fabric to verify whether an entity invoking a smart contract has sufficient permissions to invoke the endpoint. Figure 3 shows a simplified example of a client invoking three different operations on a smart contract: reading an asset, updating the asset, and deleting the asset.

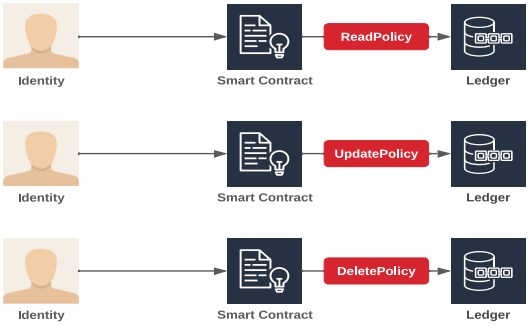

Fig. 3. Combining multiple policies. Each smart contract has a different purpose and might need different policies for different operations. Multiple policies can be defined in a single smart contract, and depending on the operation requested by the client, the correct validation policy will be selected and used for access control.

As can be seen in the image, the invoked smart contract has a different access policy for each of the three supported operations. For example, a client might be able to satisfy the `ReadPolicy` with its X.509 certificate, but might not be able to satisfy the `UpdatePolicy` and `DeletePolicy`. Therefore, this client will only be allowed to read the asset and will be denied access when it tries to update or delete the asset.

5.3 Combining ID's

In Hyperledger Fabric, IDs are composed of X.509 certificates [22], issued by Certificate Authorities and managed by Membership Service Providers. Research into combining multiple such X.509 certificates has not been published to the date of writing. In fact, X.509 certificates cannot be combined by a simple merge, since the X.509 standard [14] does not allow this. Therefore, for this study, alternative ways of combining multiple X.509 certificates had to be found.

The solution proposed in this study can integrate one X.509 certificate, referred to as the "parent", into another X.509 certificate. The process by which this integration can be realized is visualized in Fig. 4 and described below.

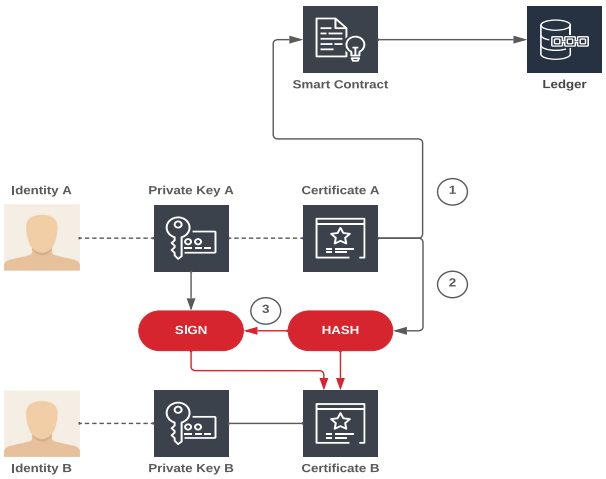

Fig. 4. Combining multiple ID's. First, the X.509 certificate of identity A is hashed. Next, this hash is signed with the private key of identity A. Finally, these two values, `hash(certificate)` and `sign(hash(certificate))`, are added to the X.509 certificate of identity B as custom attributes.

- First, the member invokes a special smart contract using certificate A (the member is authenticated with certificate A to blockchain network). This smart contract then extracts the certificate from the request, and subsequently stores it into a hashmap on the distributed blockchain ledger;
- Second, the member creates the SHA-256 hash of certificate A.

- Third, the member signs the obtained SHA-256 hash using private key corresponding to certificate A.
- Fourth, the member provides the previously performed hash value and signature to an admin using the *certgen* tool. These arguments allows the *certgen* tool to set the `hfa.ParentHash` and `hfa.ParentSignature` attributes of the child certificate (e.g., certificate B).

Whenever a client now invokes a smart contract on the blockchain network using identity B, this smart contract can verify that this client also owns identity A, since it needed access to private key A in step (3) to calculate the `hfa.ParentSignature` attribute. If the client would not have access to this private key, the signature provided in this attribute cannot be valid. Since certificate A was previously stored on the ledger in step (1), the invoked smart contract has access to the public key of identity A, and could therefore easily establish that the provided signature was forged, thus denying access to the network.

Having established that the client invoking the smart contract with identity B also owns identity A, the smart contract can retrieve the certificate of identity A from the hashmap stored on the distributed ledger, and use it to make access control decisions. The proposed smart contract has been implemented and made available in the GitHub repository[3]. This implementation currently supports one parent certificate to be set in the `hfa.ParentHash` and `hfa.ParentSignature` attributes, although it can easily be extended to support multiple parents or recursive ancestor lookups in the future.

The proposed solution to combine multiple ID's can be particularly useful for decentralized applications and IoT-device applications, where the device or application belongs to a specific owner. In these cases, the identity and access rights of the applications can easily be identified by setting the owner's certificate as the parent certificate within the X.509 certificate of each application. Furthermore, it guarantees that if an application belongs to user B, and therefore contains the hash of user's B identity in its X.509 certificate, it will not be able to access data related to user A.

5.4 Workflow in a Blockchain-IPFS-Based Network

In the previous subsections, the proposed methods of combining multiple ID's, attributes, and policies have been discussed on an individual basis. This subsection will explain how these three concepts will fit together, and how this combined design has been implemented using Hyperledger Fabric. Figure 5 shows a simplified version of the final system architecture[4].

The final system design consists of four main components, which will be described below.

[3] https://github.com/daangordijn/Fabric-Access-Control/tree/master/access-chaincode.

[4] More detailed version available at https://github.com/daangordijn/Fabric-Access-Control/blob/master/images.

Fabric CA Server. A Fabric CA Server instance will be used to issue certificates to various nodes and clients within a particular organization. Fabric CA plays a key role when combining multiple ID's, as it is responsible for creating the basic X.509 certificates and their corresponding private keys, as well as setting the `hfa.ParentHash` and `hfa.ParentSignature` attributes if applicable.

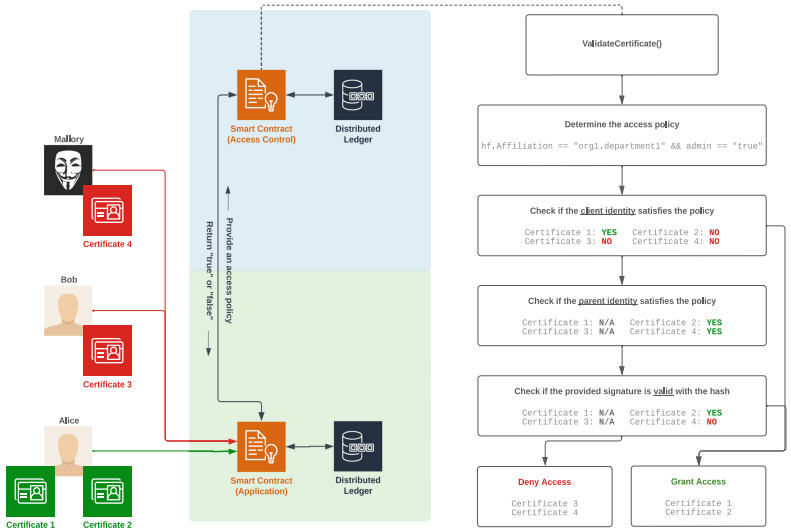

Fig. 5. Final system design, combining all discussed concepts. Certificate 1 is granted access to the resource since it satisfies the defined access policy. Certificate 2 is granted access to the resource since it contains the `hfa.ParentHash` and `hfa.ParentSignature` attributes, which connects it to certificate 1. Certificate 3 is denied access since it does not satisfy the access policy, while certificate 4 is denied access since it contains an invalid hash signature.

Security Smart Contract. The security smart contract is a custom-made smart contract that has two responsibilities.

- First, this smart contract is responsible for maintaining the "parent" X.509 certificates stored on the ledger, as described in Sect. 5.1. Clients that want to combine two identities, e.g., identity A and identity B, have to invoke this smart contract with identity A. The smart contract will then calculate the SHA-256 hash of the provided certificate, store it in the hashmap on the ledger, and return the hash to the client. Now, the client can calculate the signature and set the required attributes.
- Second, this smart contract can be invoked by other smart contracts that live on the blockchain network to determine whether a client satisfies a particular access policy. Smart contracts can make use of the `ctx.stub.invokeChaincode()` method to invoke this security smart contract, provide the access policy that has to be validated, and will then be returned a boolean

value indicating whether the client certificate satisfies the specified policy. The internal logic of this smart contract method is visualized on the right side in Fig. 5.

Client Smart Contract(s). The client smart contracts are basic smart contracts that allow clients of the blockchain network to interact with the ledger. Examples of such smart contracts are the `asset-transfer` or `commercial-paper` chaincodes provided in the `fabric-samples` repository[5]. While previously, these smart contracts had to implement their business logic to validate whether a client has access to the requested resource, developers are now able to simply invoke the security smart contract using the `ctx.stub.invokeChaincode()` method of the Hyperledger Fabric SDK, and use the returned boolean to allow or deny the client from accessing the requested resource.

Client Application(s). The client applications are basic applications that allow clients of the blockchain network to more easily interact with smart contracts, instead of having to use the peer CLI. Examples of such client applications are the `asset-transfer` or `commercial-paper` applications provided in the `fabric-samples` repository. To client applications, changes made to the proposed solution are not visible, except for the fact that some X.509 certificates containing valid `hfa.ParentHash` and `hfa.ParentSignature` attributes will now be granted access, while they would previously have been denied access from the network.

In summary, this section has presented a solution for combining multiple ID's, attributes, and policies in Hyperledger Fabric. Since this solution can be fully implemented using a single smart contract, the core components of the Hyperledger Fabric blockchain can remain unchanged. In the next section, a performance analysis will be presented, which analyses the increase in runtime due to the invocation and execution of the security smart contract.

6 Results

One of the most important considerations when proposing a new implementation is to minimize the latency and maximize the transaction throughput. To objectively analyze these performance indicators, two benchmarks of the implemented smart contract were performed with the help of the Hyperledger Caliper [8] blockchain benchmarking tool:

- **Basic:** This benchmark analyzes the average latency and throughput when the entity that submits the transaction proposal can satisfy the access policy with its own X.509 attributes; and
- **Parent:** This benchmark analyzes the average latency and throughput when the entity that submits the transaction proposal can only satisfy the access policy with a parent certificate.

[5] Available at https://github.com/hyperledger/fabric-samples.

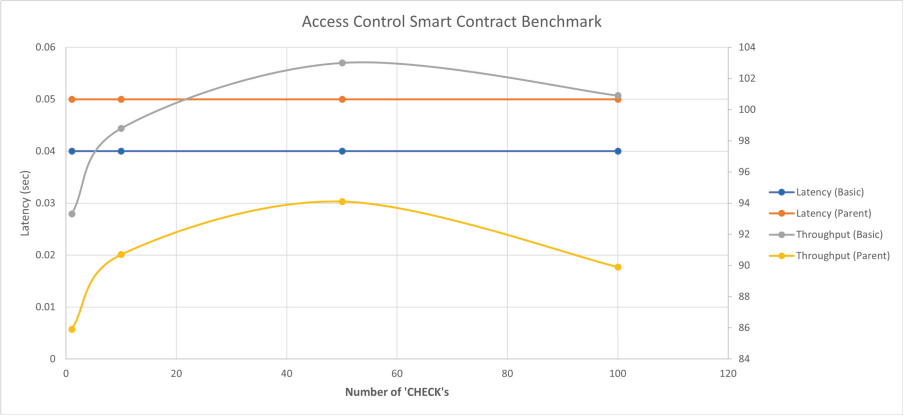

Fig. 6. Average latency and throughput of the access control smart contract, measured using the Hyperledger Caliper benchmarking tool. The blue and grey lines respectively show the average latency and throughput that corresponds to the case where the submitting entity satisfies the access policy with its own attributes, while the orange and yellow lines show the case where the access policy had to be satisfied with the parent X.509 certificate, i.e., using the `hfa.ParentHash` and `hfa.ParentSignature` attributes. (Color figure online)

The exact configuration files that have been used to perform these two benchmarks can be found in the `caliper` directory of the public GitHub repository[6].

During this study, all benchmarks were performed on a virtual machine running Ubuntu 20.04 LTS, with a total RAM memory of 8 GiB. The results that have been obtained are listed in Table 1 and visualized in Fig. 6. All reports generated by Hyperledger Caliper can be found in the previously mentioned GitHub repository.

Table 1. Average latency and throughput of the access control smart contract, measured using the Hyperledger Caliper benchmarking tool. Each row reports the measured latency and throughput associated with validating the submitted X.509 certificate on the defined access policy, which consists of n attribute checks.

Checks (n)	Latency (Basic)	Latency (Parent)	Throughput (Basic)	Throughput (Parent)
1	0.04 s	0.05 s	93.3 tx/s	85.9 tx/s
10	0.04 s	0.05 s	98.8 tx/s	90.7 tx/s
50	0.04 s	0.05 s	103.0 tx/s	94.1 tx/s
100	0.04 s	0.05 s	100.9 tx/s	89.9 tx/s

[6] Caliper configs: https://github.com/daangordijn/Fabric-Access-Control/tree/master/caliper.

As can be seen in the image, the average latency increases linearly with the number of attribute checks that have to be performed by the smart contract. On the contrary, the average throughput decreases exponentially with this same number of attribute checks. In addition, as can be seen in the image, the performance corresponding to satisfying the access policy with a parent X.509 certificate is slightly worse compared to satisfying this same access policy with its own attributes.

Finally, to objectively quantify these benchmark results, a base case was created and benchmarked using the same Hyperledger Caliper configuration. The smart contract method invoked during this base case benchmark immediately returned a Boolean value, without running any additional code. Hyperledger Caliper reported the average latency of this benchmark to be 0.04 s, and the average throughput to be 102.1 transactions per second. Comparing these values with the values listed in Table 1, it can be concluded that the increase in latency and decrease in throughput is very small. When keeping the number of attribute checks below 100, which is considered to be sufficient in most real-world applications, the decrease in performance can be disregarded.

7 Discussion

While this paper provides a working solution to solve the identified problem within Hyperledger Fabric, some improvements can be explored in future research. First, although the benchmarks performed by Hyperledger Caliper indicate that the performance impact caused by the proposed implementation is minor, research could be done into ways of improving the algorithms used to validate access policies within the smart contract. Second, the proposed implementation currently only allows users to set one certificate as their parent certificate using the `hfa.ParentHash` and `hfa.ParentSignature` attributes. Future research could be done to study whether multiple such parent certificates can be set, for example by allowing array-typed values for these two attributes. Third, since the proposed implementation only allows users to define complex access policies by combining one or more `EQUALS` or `INCLUDES` operators using the `AND`, `OR`, and `NOT` operators, research could be done into ways of allowing users to define even richer access policies. Finally, clients must currently store their private key data using file system wallets, which are considered insecure [25]. Future research could be done to allow users to store their private key data in Hardware Security Modules (HSM) to improve the security of this data.

8 Conclusions

One of the major problems of Hyperledger Fabric is that its current access control mechanism is not flexible enough for business scenarios. This study aimed to solve this issue by combining multiple ID's, attributes, and policies with the components that regulate access control.

First, to combine multiple ID's within Hyperledger Fabric, a technique has been proposed that hashes and signs one certificate, referred to as the parent certificate, and adds this hash and signature as attributes to another certificate. A smart contract has been implemented that verifies the ownership of this parent certificate.

Second, to combine multiple attributes, a flexible logic within a smart contract has been proposed that allows access policies to be defined using policy checks combined with Boolean operators. Finally, to combine multiple policies, a technique has been proposed that maintains multiple policy definitions on the distributed ledger, which can dynamically be selected as the validating policy depending on the method invoked with the transaction proposal.

Finally, in terms of performance, it has been established that for real-world applications the performance impact is negligible. For access policies with less than 100 attributes to check, the increase in average latency is below 0.01 s compared to the base case of always allowing access. However, an increase in average latency of 0.01 s has been measured when comparing the case where the access policy is satisfied with a member's own attributes with the case where the access policy is satisfied with a member's parent certificate.

Acknowledgements. This research is supported by European Unions Horizon 2020 researchand innovation programme under grant agreement No. 952697 (ASSURED), No. 101021727 (IRIS), and No. 101070052 (TANGO).

References

1. Bandara, H.D., Chen, S., Staples, M., Sai, Y.: Modeling multi-layer access control policies of a hyperledger-fabric-based agriculture supply chain. In: 2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA), pp. 355–364 (2021). https://doi.org/10.1109/TPSISA52974.2021.00039
2. Cai, W., Hong, Z., Wang, Z., Feng, C.: Decentralized applications: the blockchain-empowered software system. IEEE Access (2018). https://www.researchgate.net/publication/327711685_Decentralized_Applications_The_Blockchain-Empowered_Software_System
3. Ding, S., Cao, J., Li, C., Fan, K., Li, H.: A novel attribute-based access control scheme using blockchain for IoT. IEEE Access **7**, 38431–38441 (2019). https://doi.org/10.1109/ACCESS.2019.2905846, https://ieeexplore.ieee.org/document/8668769
4. Ed-Daibouni, M., Lebbat, A., Tallal, S., Medromi, H.: Toward a new extension of the access control model ABAC for cloud computing. In: Sabir, E., Medromi, H., Sadik, M. (eds.) Advances in Ubiquitous Networking. LNEE, vol. 366, pp. 79–89. Springer, Singapore (2016). https://doi.org/10.1007/978-981-287-990-5_7
5. Ethereum: Smart Contract Languages. https://ethereum.org/en/developers/docs/smart-contracts/languages/
6. Frankenfield, J.: Smart Contracts. https://www.investopedia.com/terms/s/smart-contracts.asp
7. Hyperledger: Fabric CA User's Guide. https://hyperledger-fabric-ca.readthedocs.io/en/release-1.4/users-guide.html

8. Hyperledger: Hyperledger Caliper. https://hyperledger.github.io/caliper/v0.5.0/getting-started/
9. Hyperledger: Hyperledger Fabric Documentation. https://hyperledger-fabric.readthedocs.io/en/release-2.2/
10. Hyperledger: Hyperledger Fabric SDKs. https://hyperledger-fabric.readthedocs.io/en/release-2.2/fabric-sdks.html
11. Hyperledger: Using the Fabric Test Network. https://hyperledger-fabric.readthedocs.io/en/release-2.2/test_network.html
12. Hyperledger: What is Hyperledger Fabric? https://hyperledger-fabric.readthedocs.io/en/release-2.2/whatis.html
13. Iftekhar, A., Cui, X., Tao, Q., Zheng, C.: Hyperledger fabric access control system for internet of things layer in blockchain-based applications. Entropy **23**(8) (2021). https://doi.org/10.3390/e23081054, https://www.mdpi.com/1099-4300/23/8/1054
14. International Telecommunication Union: Public-Key and Attribute Certificate Frameworks. https://www.itu.int/rec/T-REC-X.509-201910-I/en
15. IPFS: IPFS Documentation. https://docs.ipfs.io/
16. Islam, M.A., Madria, S.: A permissioned blockchain based access control system for IOT. In: 2019 IEEE International Conference on Blockchain (Blockchain), pp. 469–476 (2019). https://doi.org/10.1109/Blockchain.2019.00071
17. Kelleher, J.: Why do bitcoins have value? https://www.investopedia.com/ask/answers/100314/why-do-bitcoins-have-value.asp
18. KeyFactor: What is PKI and How Does it Work? https://www.keyfactor.com/resources/what-is-pki/
19. Lutkevich, B.: What is access control? https://www.techtarget.com/searchsecurity/definition/access-control0
20. McGovern, T.: How many blockchains are there in 2022? https://earthweb.com/how-many-blockchains-are-there/
21. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system. Bitcoin.org (2008). https://bitcoin.org/bitcoin.pdf
22. National Institute of Standards and Technology: X.509 Public Key Certificate. https://csrc.nist.gov/glossary/term/x_509_public_key_certificate
23. Peng, L., Feng, W., Yan, Z., Li, Y., Zhou, X., Shimizu, S.: Privacy preservation in permissionless blockchain: a survey. Digit. Commun. Netw. (2020). https://www.researchgate.net/publication/342455474_Privacy_preservation_in_permissionless_blockchain_A_survey
24. Rouhani, S., Deters, R.: Blockchain based access control systems: state of the art and challenges. In: IEEE/WIC/ACM International Conference on Web Intelligence, WI 2019, pp. 423–428. Association for Computing Machinery, New York (2019). https://doi.org/10.1145/3350546.3352561
25. Solana: Command Line Wallets. https://docs.solana.com/wallet-guide/cli
26. Song, L., Li, M., Zhu, Z., Yuan, P., He, Y.: Attribute-based access control using smart contracts for the internet of things. Proc. Comput. Sci. **174**, 231–242 (2020). https://doi.org/10.1016/j.procs.2020.06.079, https://www.sciencedirect.com/science/article/pii/S1877050920315933. 2019 International Conference on Identification, Information and Knowledge in the Internet of Things
27. SSL.com: What is a certificate authority? https://www.ssl.com/faqs/what-is-a-certificate-authority/
28. Tapscott, D., Tapscott, A.: The blockchain revolution: how the technology behind bitcoin is changing money, business, and the world (2016)

29. Xu, M., Chen, X., Kou, G.: A systematic review of blockchain. Financial Innovation (2019). https://jfin-swufe.springeropen.com/articles/10.1186/s40854-019-0147-z
30. Yang, Z., Shao, D., Qu, L., Zhang, M.: Internet of things access control system based on hyperledger. J. Phys.: Conf. Ser. **1748**(4), 042031 (2021). https://doi.org/10.1088/1742-6596/1748/4/042031
31. Yutaka, M., Zhang, Y., Sasabe, M., Kasahara, S.: Using ethereum blockchain for distributed attribute-based access control in the internet of things. In: 2019 IEEE Global Communications Conference (GLOBECOM), pp. 1–6 (2019). https://doi.org/10.1109/GLOBECOM38437.2019.9014155
32. Zhao, X., Wang, S., Zhang, Y., Wang, Y.: Attribute-based access control scheme for data sharing on hyperledger fabric. J. Inf. Secur. Appl. **67**, 103182 (2022). https://doi.org/10.1016/j.jisa.2022.103182, https://www.sciencedirect.com/science/article/pii/S2214212622000643