Flexible Fine-grained Data Access Management for Hyperledger Fabric

1st João Parente
INESC TEC & U.Minho
Braga, Portugal
joao.p.parente@inesctec.pt

2nd Ana Nunes Alonso
INESC TEC & U.Minho
Braga, Portugal
ana.n.alonso@inesctec.pt

3rd Fábio Coelho
INESC TEC & U.Minho
Braga, Portugal
fabio.a.coelho@inesctec.pt

4th João Vinagre
INESC TEC & U.Porto
Porto, Portugal
joao.m.silva@inesctec.pt

5th Paulo Bastos
NAU 21
Porto, Portugal
paulo.bastos@nau-21.com

***Abstract*—As blockchains go beyond cryptocurrencies into applications in multiple industries such as Insurance, Healthcare and Banking, handling personal or sensitive data, data access control becomes increasingly relevant. Access control mechanisms proposed so far are mostly based on requester identity, particularly for permissioned blockchain platforms, and are limited to binary, all-or-nothing access decisions. This is the case with Hyperledger Fabric's native access control mechanisms and, as permission updates require consensus, these fall short regarding the flexibility required to address GDPR-derived policies and client consent management. We propose SDAM, a novel access control mechanism for Fabric that enables fine-grained and dynamic control policies, using both contextual and resource attributes for decisions. Instead of binary results, decisions may also include mandatory data transformations as to conform with the expressed policy, all without modifications to Fabric. Results show that SDAM's overhead w.r.t baseline Fabric is acceptable. The scalability of the approach w.r.t to the number of concurrent clients is also evaluated and found to follow Fabric's.**

***Index Terms*—access control, blockchain, privacy, confidentiality**

I. Introduction

Since inception, blockchain applications have diversified from implementations of cryptocurrencies [1] to enabling distributed application in a multitude of domains. Along with a multitude of uses, different trust models emerged, from the permissionless model where anyone can participate in the blockchain network and trust is tied to the consensus protocol (PoW [1], e.g), to the permissioned model, as in Hyperledger Fabric (HLF) [2], where a consortium of nodes decide on the set of operations to be committed, where more traditional BFT consensus protocols can be used [3], [4]. Public permissionless networks, such as Bitcoin and Ethereum, by design, expose transaction attributes and records, even if the association between wallets and users is not direct. However, some use cases may require transactions and/or its attributes to be private, while maintaing public accountability. Different concerns can be grouped under the concept of privacy:

1) keeping values, which might represent personal data, involved in transactions, private;
2) that proposed or committed transactions between peers are kept secret from competitors;
3) or that the rules that govern negotiation, and that might be implemented in smart contracts, be opaque.

Several approaches to handle each of these concerns in both permissioned and permissionless blockchains have been proposed. In this work, we focus on 1. For example, Fabric uses two different constructs to accommodate 1) and 2). Channels define groups of transaction visibility (and persistence). Private collections define which participants can see the actual data in a collection or just a hashed version.

Handling sensitive information, such as clinically relevant or simply personally-identifiable information, increasingly requires striving for confidentiality and managing user consent, namely regarding how, for which purposes, and by whom their data can be processed [5].

This work was motivated by the development of a collaborative platform that enables processes in the insurance domain to be dematerialized and automated focusing on contract negotiation and claim processing, using Hyperledger Fabric. The issue of handling personal data impacts not only insurance underwriting and claim processing, but also other uses such as performing analytics, including using machine learning, to generate new business insights.

However, Fabric's current access control mechanisms, with channels and collections, are coarse-grained, geared towards enforcing permissions for peers and organizations in accessing related sets of data. Thus, these are not well-suited for enforcing policies such as a user giving consent for personal data to be used in for a specific purpose (e.g. claim pro-

The work of A. N. Alonso was financed by the ERDF - European Regional Development Fund, through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme under the Portugal 2020 Partnership Agreement, and by National Funds through the FCT - Portuguese Foundation for Science and Technology, I.P. on the scope of the CMU Portugal Program within project AIDA, with reference POCI-01-0247-FEDER-045907. The work of F. Coelho was financed by the ERDF - European Regional Development Fund through the North Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement within project SISˆ1 (NORTE-01-0247-FEDER-45355).

cessing) but not others (e.g. cross-selling) within the same organization, the same peer. An alternative could be to use libraries such as CID[1], to define identity-based access control in chaincode. However, issues arise: maintainability, as access control code would need to be written for each chaincode function; updatability as each change in access policy might require changing chaincode directly and redeploying. Additionally, enforcing access control mediated by user consent, which can be dynamic, is even less amenable to chaincode-based implementation. This makes it necessary to find suitable mechanisms for expressing and enforcing dynamic access control policies, pliable to user consent, specifically for reading information. Suitability depends not only on the expressiveness and flexibility of the chosen mechanism, as to fit the different needs that may arise from different organizations, but also on the simplicity of conveying and managing permissions.

We propose the Smart Data Access Management system (SDAM), a middleware-based access control system for Hyperledger Fabric that enables the flexible and manageable definition and enforcement of dynamic fine-grained access control policies. It does not require any change to Fabric's implementation and works in tandem with the native control mechanisms, nor does it require all peers or organizations in a network or channel to adopt this system. It provides a simple way to implement GDPR-based policies, namely in terms of purpose and right to forget and can do so dynamically, as changes in permissions do not require chaincode or Fabric to be re-instantiated or changed in any way. Additionally, it can be used to enable off-chain access to on-chain data, as well as on-chain access to off-chain data. The remaining paper is organized as follows: Section II presents relevant background, while Section III addresses related work. Section IV introduces SDAM and Section V assesses the overhead and scalability of the system. Finally, Section VI concludes the work and sets challenges for future work.

II. Background

First, we provide a focused overview of access control, and of the XACML standard [6] in particular. Then, we describe how Hyperledger Fabric implements access control.

A. Access Control

Access control serves to limit the actions that a user or process can take regarding a given resource or sets of resources. Ideally, it should be complemented by auditing, thus recording who, when, the intended action and its potential success [7]. Implementations vary: access control lists (ACLs) associate, to each resource, a list of users or processes and the respective level of access. Capabilities take a dual approach, associating to each user or process the list of accessible resources and the level of access. Policies can then be built on these implementations:

- discretionary policies follow directly from access control lists, using a user's identity to determine permissions;
- mandatory policies define a hierarchical ordered set of levels of access for resources and as a clearance associated to each user;
- role-based policies define roles based on the activities that users perform in the system, associate a set of permissions to each role, and require authorization to allow users to assume roles;
- attribute-based policies enable complex rules to be defined based on characteristics or properties of the user, the resource intended to be accessed and possibly environmental attributes to decide whether a given access request should be granted.

Attribute-based access control can implement discretionary, mandatory and role-based policies as well as finer-grained policies, being particularly well-suited for federated systems [8].

The XACML standard [6] defines an architecture that separates policy enforcement, decision-making and administration into a set of components geared for attribute-based access control.

- Policy Enforcement Point (PEP): component to which the user makes requests and that applies request decisions.
- Policy Information Point (PIP): a repository of client information (attributes).
- Policy Decision Point (PDP): the component that decides whether to allow or deny the client request.
- Policy Administration Point (PAP): the component that manages the access control policies.
- Policy Retrieval Point (PRP): a component that stores access control policies.
- Accounting or Auditing: a component responsible for tracking access attempts.

The request flow is as follows:

1) The user makes a request to the PEP, which forwards the request to the PDP.
2) The PDP retrieves the relevant policies from the PRP (managed by the PAP, as needed) and attributes from the PIP (if needed).
3) The PDP decides on the request and notifies the PEP of the decision.

Policies are specified using a markup language, with three main elements: rules, that contain boolean expressions; policies, i.e. sets of rules; and policy sets, containing multiple policy or policy set elements. Other standardization proposals include PERMIS [9] and SDDL [10].

B. Hyperledger Fabric

Hyperledger Fabric is part of the Hyperledger meta-project, an effort to promote blockchain utilization in industry. Fabric is a permissioned distributed ledger, geared towards use cases where the identities of participants are (or must be) known, such as to comply with anti-money-laundering regulations. The system is modelled as a replicated state machine, modified by transactions with immediate finality. The ledger is composed of: a blockchain, to store an immutable sequence

[1]https://github.com/hyperledger/fabric-chaincode-go/tree/main/pkg/cid

of transactions and a state database, to store the current state. Along with a global ledger, Fabric allows groups of participants to create separate transaction ledgers, thereby creating closed, private channels. Only channel members store a copy of the channel's ledger.

One of Fabric's distinguishing features is its execute-order-validate philosophy, which contrasts the order-execute philosophy of competing implementations. Instead of requiring transactions to be deterministic, by executing the transactions and then ordering their tentative results precludes the possible divergence caused by obtaining different outcomes at different peers when executing a non-deterministic transaction. Participant nodes in a Fabric network can have one of the following roles: clients propose transactions, request transaction ordering; peers maintain the ledger; endorsing peers (endorsers), are defined according to the chaincode's endorsement policy, execute proposals and validate transactions; orderers implement consensus to totally order transactions; the same set of nodes can be used by multiple channels as these are unaware of the global state.

Smart contracts (chaincode) on the Fabric framework can be implemented in multiple programming languages and are typically executed in Docker containers. External applications interact with a Fabric network by invoking chaincode. In most cases, chaincode queries or updates the global state, but does not interact directly with the transaction log proper. Chaincode is installed by a transaction, only on endorsing peers and requires appropriate permissions. An endorsement policy must be associated to the chaincode, which defaults to requiring the endorsement of a member of any of the organizations present in the channel. Other policies, such as requiring a given number of endorsers or specific combinations of roles and organizations are possible.

Due to its permissioned nature, participants need be identified. Identity management (and authentication) in Fabric is done through Membership Service Providers (MSP). Each actor (users/clients or nodes (peers, orderers)) has an associated local MSP that: authenticates and handles access control and permissions for users within its organizational domain; handles permissions for the capabilities of the node.

Fabric implements two types of access control: (1) simply put, to define who can operate on given resources, for example, defining the set of organizations that must agree for an update to be committed, be it a transaction, chaincode update or channel configuration update; and (2) to regulate data access, using channels and private collections to implement multiple data sharing patterns among organizations. Type (1) is orthogonal to our proposal and is therefore not the subject of further discussion here. Regarding (2), channels are the primary mechanism for data segmentation, as these have separate ledgers and only peers of member organizations store and access data associated to a channel. Private data collections make it possible to restrict data sharing to a subset of channel organizations and implement more flexible sharing patterns[2]. Permissions are based on identity and/or organization membership, and changes requiring consensus are embedded in channel's configuration or defined in chaincode. Also, decisions are binary, i.e. allow or deny. We propose a more flexible and dynamic access control mechanism that complements existing controls, predicated on multiple attributes other than identity and capable of implementing more flexible decisions, i.e. by defining mandatory data transformations.

III. Related Work

Focus on access control in the blockchain domain has been spurred, in part, by the need to handle personal data, increasingly desired for value extraction and increasingly protected by legislation. In order for personal data to be legally processed, the owner must consent to it explicitly, considering a set of purposes and processors. An analysis of the interplay between the GDPR and blockchains is available in [11].

Bhaskaran et at [12] propose a blockchain-based platform that integrates consent management, access control and data sharing, using Fabric. The goal of the platform is to share KYC information to avoid repeated instances of the customer enrolment process. Two of the main goals are to guarantee that data sharing conforms strictly to the consent expressed by the customer while maintaining customer/provider relations hidden. The identity of the providers is concealed by through pseudonymous certificates, while data sharing rules are enforced through smart contracts that regulate the availability of decryption keys. As presented, the access control policy is very simple, with customers stating explicitly the list of documents a provider should have access to.

The approach presented in [13] exemplifies some pitfalls in consent management. Specific issues include storing user passwords in smart contracts and, more relevant to the particular goal of the system, the terms of consent the user actually agrees to are apparently not stored in the blockchain. Also, consent seems to be binary, without the possibility of considering some granularity of implementing complex access control policies.

A different approach is to map relevant aspects of the GDPR for data handling and user rights to blockchain records and transactions [14]. Three types of participants are considered: users, those that own the data and can consent to its use; data controllers, those that collect and store the data; and data processors, those that process the data for some purpose. Consent records should explicitly state to which categories of data it extends, the authorized processing purposes, conditions of storage, authorized time frame for processing, identify the data controller or joint controllers and authorized processors. When a data controller grants a data processor access to some collected data, information related to this action are also stored, including the categories of data that where shared, the authorized processing purposes and time frame and the identity

[2]https://hyperledger-fabric.readthedocs.io/en/release-2.4/private-data/private-data.html

of the data processor. This makes it possible for users to audit how their data are being handled.

While most presented solutions do not provide support for implementing complex access control policies, there are proposals for leveraging machine learning techniques for policy analysis, conflict detection and recommendation [15]. Truong et al. [16] propose a framework combining consent management and access control over Fabric (or Ethereum), taking compatibility with the GDPR into consideration in defining roles and requirements. Two ledgers are used: one for authentication, authorization and access control and the other for validation and logging. Data are not shared through the blockchain platform, storing a pointer to the encrypted data instead. A Distributed Hash Table (DHT) may be used as a distributed data store for this purpose [17]. Moreover, data policy is recorded in the blockchain, defined as an access control list of public keys associated to CRUD operations, in JSON.

Rouhani et al. [18] propose using Fabric to implement an access control system, compatible with XACML, that implements enforcement, administration and decision components as smart contracts in Fabric, storing access control attributes and policies as JSON, in the ledger. The protected resources are off-chain. Access requests and decisions are thus part of the ledger and can be audited. Fabric's private data collections are used to store sensitive user attributes. By configuring endorsement policies appropriately, multiple (or specific) organizations can be required to agree on an access request for it to be granted.

The need for multiple authorities to validate different sets of user attributes in the context of a given access request is the focus of [19]. In short, data are encrypted by the owner, who also creates an Ethereum smart contract that specifies the data's access policy. A prospective user attempting to access the data will need to obtain authorization tokens from a set of authorities, which depend on the specific attributes. In order to do so, the user creates a set of smart contracts: one for each required authority and one to request the decryption key from the data owner. While the decryption key is encrypted with the requesting user's public key, to avoid exposing it in the ledger, it should be noted that there is no mechanism that prevents the user from leaking the decryption key and compromising the confidentiality of the data. While SDAM also uses XACML, enforcement, administration and decision components are implemented as middleware, instead of smart contracts, thus extending Fabric capabilities generically.

A significantly different approach, the Digital Asset Modelling Language (DAML) [20] enables the implementation of smart contracts in a strongly-typed, functional language. This makes contracts amenable to formal verification, a measure of which is included in the framework. DAML has built-in types and functions that allow concepts such as contracts, parties, rights, obligations, and authorization to be expressed directly in the language. DAML supports multiple blockchains, including Fabric (in alpha version) through either commercial or community editions. DAML implements access control in its runtime, along with business logic, sidestepping Fabric's access control mechanisms.

Complementary approaches for controlling data access include establishing a trustworthy oracle for accessing external data, which is particularly useful for Ethereum, as by design, smart contracts are restricted to on-chain interactions. Another approach is to control computation along with data access by establishing limits on the computation that can be performed [21]. Data do not leave the owner's infrastructure and, instead, the computation expressed in a MapReduce model by the requester is validated and executed in the infrastructure where the data reside. Validation includes approving specific operations to be executed over the data.

In summary, there have been several proposals for integrating blockchain and access control but these mainly consist of devising access control systems as applications of blockchain technology, i.e. using a blockchain to register and enforce user consent, optionally supporting (possibly external) data sharing [12]–[14], [16], [18], [22]. We propose a significantly different approach: extending the blockchain platform (Fabric) with the capability for flexible and fine-grained access control. In fact, we can take advantage of the same approach to provide seamless but controlled access to off-chain data from chaincode execution, as well as controlled access to on- or off-chain data for external applications. Moreover, we extend policies with the optional definition of mandatory data transformations as conditions for data access.

While this discussion is out of scope for this work, SDAM can leverage this type of platforms for managing user consent, further integrated into policy definitions to be enforced by the SDAM middleware. Another important difference is that organizations can choose to use SDAM and still seamlessly interact with those that choose not to. This means that SDAM can be used and useful without requiring all organizations to agree on its usage, thus lowering the barrier for entry.

IV. Smart Data Access Management (SDAM)

SDAM is designed as middleware, serving as a proxy between data requestors and data sources. This enables requests and responses to be transformed as needed to enforce flexible access control policies. It replicates CouchDB's REST API, so that no changes to Fabric's implementation are required. All that is required is for a peer to be configured to direct its queries to SDAM as if it were its CouchDB instance. This could be done by changing the appropriate Dockerfile. SDAM can also be used to allow Fabric's peers to access off-chain data, by configuring external data sources in the middleware, redirecting requests and processing queries and responses to the external data sources. It can also be used to allow external access to a peer's on-chain data, with the same guarantees, by serving external requests through the same REST API that is exposed to Fabric peers. The remainder of this section focuses on using SDAM within typical Fabric operation, i.e. with queries for on-chain data, within the regular transaction lifecycle. It should be highlighted that each peer's (or organization's) SDAM proxy is independent. There is no

need to synchronize SDAM instances across organizations as each may require different policies to be implemented and some may not wish to use SDAM at all.

A. Architecture

The system architecture is presented in Figure 1. SDAM is based on two modules implementing XACML functionality: (i) the Proxy, which interfaces with a Fabric peer, queries the underlying state database and serves as a PEP, enforcing access control decisions emitted by the Policy module; and (ii) the Policy module, which stores access control policies and emits decisions on authorization requests, serving as PAP, PRP and PDP. SDAM does not require a PIP, but one could be added if needed.

1) SDAM Proxy Module: The Proxy is the central module of SDAM. It exposes CouchDB's REST API enabling seamless integration with Fabric, without requiring customizations. As an example, retrieving a document from the database by document id can use CouchDB API method `GET /{db}/{docid}` [3] which translates to `GET http://url/database/documentID`. Other API methods used by Fabric include retrieving multiple documents `POST /{db}/_bulk_get` and querying for documents matching given criteria `POST /{db}/_find`. SDAM supports all API calls used by Fabric. With SDAM, API requests carry additional information in the body of the request (e.g. *a1*, *b1* and *b2*), using JSON declarative syntax, i.e. attributes relevant for access control. Using a declarative syntax provides the flexibility to define attributes as needed for each request: these can be related to identity, purpose, conditional use according to a defined time interval, etc. Application logic dictates which context attributes to inject into a given request, e.g. intent. Others, such as requester identity are always injected.

When a request is received by the Proxy (e.g.*query(A,a1)*), it strips away the access control attributes before forwarding it (*query(A)*) to the CouchDB instance. An XACML request is created from these attributes along with attributes from the retrieved documents: it may identify the document type being accessed, subjects to whom information pertains [4], etc. The XACML request is sent to the Policy module which replies with a decision to deny or allow access, along with optional *ObligationExpressions*, a list of mandatory transformations to perform on the documents before returning these to the peer. For example, an obligation may require the *JobInfo* field and any sub-fields to be removed from the documents the policy applies to, for a given request *intention*.

The Proxy implements a set of generic *Transformations* that operate on document fields. Current supported transformations include *Filter* (pink pentagon), which removes the specified fields from a document (matching circle disappears - depicted as *query(A,a1)*), and *Obfuscation* (purple and green pentagons), which instead of removing fields applies a transformation. The latter hides the raw value and replaces it with one that can still be used to perform some operations, resembling a fingerprint (hash code) (purple circle becomes diamond - depicted as *query(A,a1)*), or replacing a numeric value with one with less precision, e.g. replacing age with age bracket (green circle becomes triangle - depicted as *query(B,b1)*). Adding new transformations simply requires writing a Java class that implements a method that operates on ObjectNode type and a constructor that receives the corresponding *ObligationExpression*. Write requests and replies are simply forwarded through the Proxy and are only subject to Fabric's native access control mechanisms. The Proxy module has been implemented using the Spring Reactive Framework.

2) SDAM Policy Module: The Policy module manages and stores access control policies based on XACML and decides on access requests. We use DPD AuthZForce[5] for this purpose without modification. For most use cases, it would make sense to deploy one Policy module per organization, allowing policies to be managed at the organisation level. We envision policies originating from two main mechanisms: structural policies, stemming from business logic processes and/or organization management structure, which would probably be generally applicable per document type, for example; an consent-driven policies, with a more dynamic nature, generated from the expression of consent by individuals. A platform for consent management is out of scope but could probably be addressed by adapting some of the proposals discussed in Section III.

B. SDAM and Insurance

The need for SDAM was motivated by the development of a collaborative network for automating and simplifying business processes mainly in contract negotiation and claim processing, with applications to multiple insurance domains: event-related insurance, reinsurance management, savings plans, etc. Consortia to support these use cases would consist of multiple stakeholders with possibly very different profiles, including: SMEs, large companies, healthcare providers, event organizers, regulators, etc. SIS, represented in Figure 1, is a system under active development geared mainly towards insurance companies to mediate their participation in blockchain-based collaborative networks such as Fabric. Opportunities arise from collaborative data sharing, for example, to leverage on-chain data to calculate more accurate estimates of factors that impact premium calculations. But these can come into conflict with the need for tracking individual consent, to comply with data protection regulations, namely for processing for non-essential purposes. We can, therefore, improve individual privacy by restricting visible information to that strictly relevant for a given purpose, within organizations. Other blockchain-based use cases for Insurance have been proposed [23], [24] with different purposes.

Let us consider a client's address, composed of several fields: street, building and/or apartment number and city. Taken

[3]https://docs.couchdb.org/en/3.2.0/api/document/common.html#get–db-docid

[4]Implementing "right to forget" would simply require denying access to all documents pertaining to a specific subject"

[5]https://authzforce-ce-fiware.readthedocs.io/en/latest/

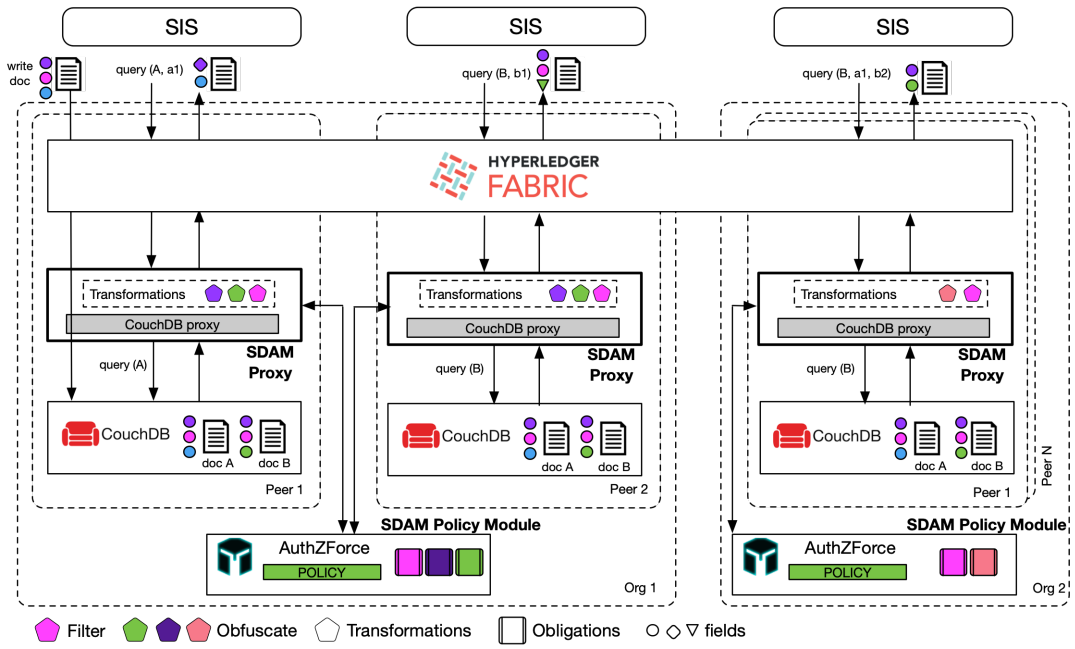

Fig. 1. SDAM Architecture. Pentagons represent available transformation implementations and squares represent obligations defined at the Policy Module.

as a whole it can probably be used to identify the client, without being provided with a name. The full address will be required if underwriting house insurance. But what about car insurance? Knowing the city should be enough, maybe the neighborhood for larger cities, but not the full address. As such, there should be no need to expose the full address to an insurance mediator when simulating car insurance premiums. Moreover, it may happen that a client consents to sharing only a subset of information to be processed for cross-selling purposes. An even simpler example can be found in the banking industry: when making an in-person deposit, the teller doesn't really require access to previous banking operations (privacy-breaking) on that or other accounts, but just to the before and after balances (privacy-preserving), even if the organization, the bank has that information. Naturally, if a customer requests an account extract, the teller has a purpose for accessing this information. These insights fuel the need for finer-grained access control that takes multiple attributes into account along with identity, namely intention or purpose.

SIS instances hold application context and invoke chaincode functions on their organization's peers, also injecting the appropriate attributes for access control with each request, on behalf of employees or automatic processes. Each peer requires an instance of an SDAM Proxy and CouchDB, but there can be a single SDAM Policy module per organization, common to all Proxies. As mentioned in Section IV-A writes and other operations that do not retrieve data from the state database are simply forwarded through the Proxy along with its replies. It should be pointed out that SDAM does not impact the content of the state database in any way. That is managed completely by Fabric's consensus and peer-replication mechanisms, thus documents *doc A* and *doc B* are the same in all state databases, unless if part of some private collection. Naturally, transaction validation also eschews access control.

The insurance use case is as follows: SIS provides insurance companies with a platform for managing multiple workflows regarding insurance negotiation, underwriting, claim processing and persistence, among others, interacting with customers, competing insurance companies and other consortium members. Documents may represent insurance contracts, with coverage and conditions, for example. Figure 1 depicts an example with two organizations, with two peers for *Org1* and one for *Org2*. As an example, let us consider that the pink circle represents a client's name, the purple circle, a job title and the green circle, age. On the left, the query for document *doc A* with attribute *a1* results in the application of the Filter policy (pink), which removes the client's name from the document, and the purple Obfuscate policy to be applied, which replaces the job title with its hashed value. The other queries, in the middle and on the right both query for document *doc b*, but get different results: in the middle, in what could be an analytical task, attribute *b1* resulted in the application of the green Obfuscate policy, which replaces the age with an age bracket possibly a consequence of the expression of user consent; on the right, attributes *a1* and *b2*, resulted in just filtering out that client's name.

While this section describes an insurance centric use-case, SDAM is applicable to any sort of use-case where consortium-

generated data require fine-grain access control policies geared by according to application context.

V. Evaluation

We evaluate SDAM by assessing its overhead w.r.t. a *vanilla* deployment of Fabric and its scalability as the number of concurrent clients increases. The evaluation setup uses Fabric's test network, configured with two organizations, with a peer each. Each peer uses its own CouchDB instance as the state database. Where SDAM is used, an instance of the SDAMProxy is deployed, co-located with each CouchDB instance, and an instance of the SDAM Policy Module (AuthZForce) is deployed per organization, co-located the respective organization's peer. The test-network is deployed in a single machine with 12 CPUs, 16 GB RAM and SSD storage. Clients are evenly deployed to two similar host machines, connected to the test-network deployment by a Gigabit network. The only Fabric test-network configuration that is modified from the default is the address and port of the state database, pointing it to the SDAM Proxy instead.

The experiment assesses the overhead of reading data from the state database (i.e. CouchDB) through SDAM. Requests use the rich querying mechanism (getQueryResult()), more useful for exploratory or analytical purposes, which also prevents Fabric from caching responses. Using Apache JMeter[6], each client (thread) invokes Fabric's `peer` command, repeatedly, for 30 seconds[7], issuing a new request after receiving the response from the previous one. All requests are directed to the same peer. Each test was run 3 times, with results showing the average of each run set.

We defined a policy that applies to the retrieved document that specifies that if the user (application-dependent attribute) belongs to a given group (application-dependent attribute), the document can be accessed but only after applying a Filter transformation, to hide the document attribute `name`. Data access policies are cached at the SDAM Proxy, as these are unlikely to be very dynamic, to avoid the performance penalty of consulting the Policy Module whenever a request is made. Caching can be turned off which we found to add an average of 2 milliseconds (1 client) to 571 milliseconds (200 clients) to the global response time.

Figure 2(a) shows the overhead of using SDAM w.r.t the baseline (Fabric) as the number of clients increases. Figure 2(b) shows a break down of the response time. Figure 3 shows the cumulative distribution of response times when using SDAM for increasing numbers of clients.

A. Overhead

As the number of clients increases, we observe that SDAM follows the same trend as the baseline, with an increase in response time. Between 1 and 10 clients, the overhead is almost negligible. The average overhead of using SDAM over all workloads is of 92 milliseconds.

[6]https://jmeter.apache.org

[7]Longer runs showed the same behaviour.

We instrumented the SDAM Proxy so that we can decompose the response time into: querying CouchDB; retrieving decisions and policies from the Policy Module or from the cache; applying transformations to the data; as well as network latency, HLF request and internal processing and Spring routing (*Other*). Figure 2(b) offers a breakdown of the time each distinct step takes. Regardless of the number of clients, querying CouchDB and *Other* tasks are by far the main contributions to the overall response time, rendering the impact of retrieving the policy and transforming the data negligible (below 0.03 and 0.005 milliseconds, respectively).

We highlight that, overall, HLF is not a system geared for querying performance. Experiments with the same requests being directed to SDAM, bypassing Fabric, showed that Fabric contributes significantly to the response time: going through Fabric imposes a penalty of 45 to 402 milliseconds w.r.t. querying SDAM directly, for 1 to 200 clients. We could observe a reasonable amount of variability in the results stemming, in part, from the behaviour of the peer's CouchDB instance. For example, for 200 clients, the difference between the average response time between SDAM and the baseline is 118 milliseconds, while the standard deviation of querying CouchDB was between 164 and 212 milliseconds. This effect propagates to the overall evaluation of SDAM. Still, the average overhead of 92 milliseconds across all tests ranks as a good compromise in favor of the added capabilities.

B. Scalability

Figure 2(a) and (b) showcase the behavior as the number of clients increases from 1 to 200. The trend shows an increase in the total response time per request. The steady performance penalty and the growing trend for both HLF and SDAM suggest that the system can potentially scale beyond 200 clients, which we found to be an adequate upper-bound for the described use case. We should highlight that these results refer to a single peer and that multiple peers scale independently, even within the same organization.

The empirical CDFs for response time with SDAM, as the number of clients increases is depicted in Figure 3. With 1 up to 10 clients, over 90% of requests completed in less than 5 milliseconds. With 30 clients, 90% of clients completed in less than 10 milliseconds, with the remaining 10% accumulate response times going up to 100 milliseconds.

With 100 or more clients, distributions have larger tails, which indicates larger variability in response times. As mentioned, this seems to be a direct consequence of the variability introduced by the underlying system, namely CouchDB.

VI. Conclusion and Future Work

This paper introduced SDAM, a middleware system that extends HLF with the capacity for flexible and fine grained access control policies on on-chain data, leveraging data transformations. Working as a proxy, SDAM sits between each HLF peer and its CouchDB state database. It exports the same API as CouchDB for transparent operation, thus eschewing the need to change HLF's implementation. We

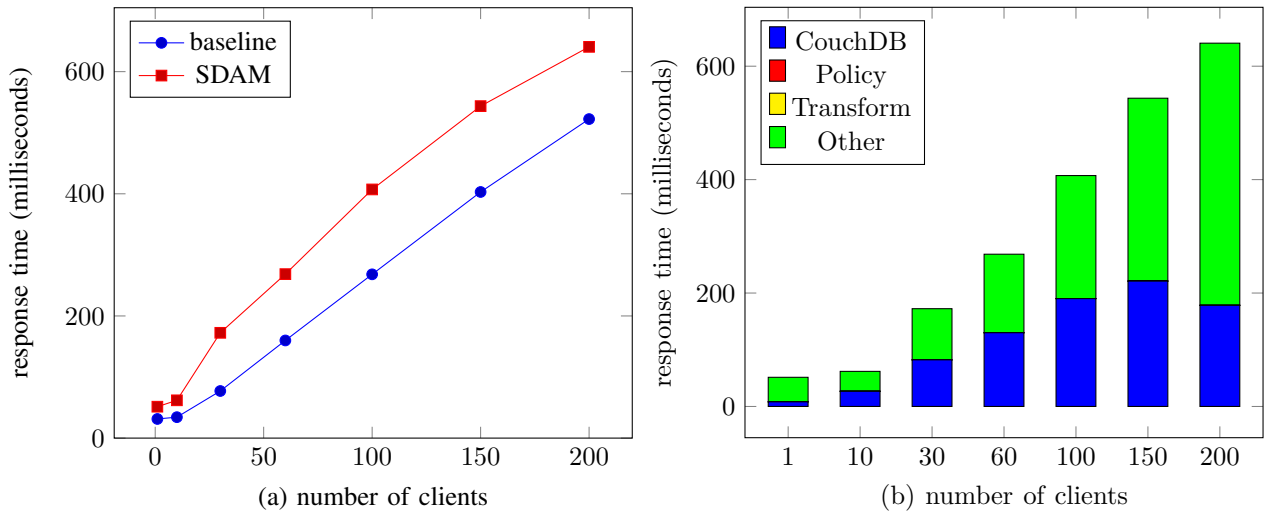

Fig. 2. (a) Comparison of the baseline HLF and HLF with SDAM. (b) Response time breakdown.

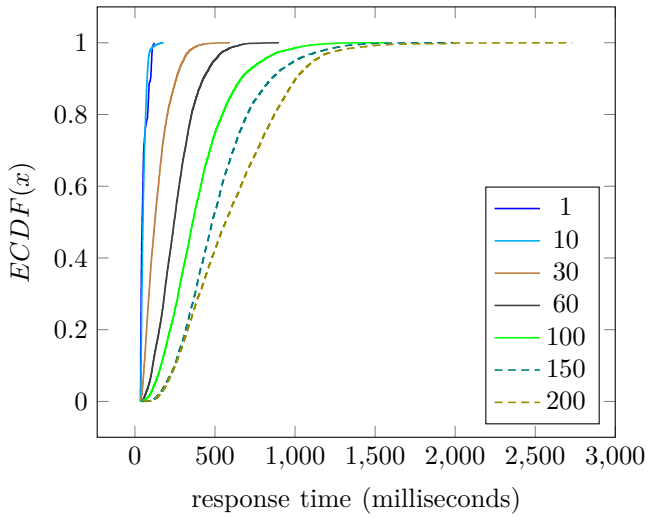

Fig. 3. Empirical CDFs for SDAM for increasing numbers of clients.

extend data access policies with the optional definition of mandatory data transformations as conditions for data access to improve flexibility. Also, organizations can choose to use flexible access control and still seamlessly interact with those that choose not to, thus lowering the barrier for adoption.

The evaluation campaign validated the query workflow and assessed the overhead and scalability of the solution: the average response time penalty is 92 milliseconds, yielding a small, acceptable impact on performance, considering the added functionality; and SDAM's scalability follows HLF's.

The current version of SDAM is geared towards mediating access when querying the state database, not querying nor writing to the ledger. Thus, SDAM only intercepts getQueryResult() calls, which generally should not be used in transactions submitted for ordering. SDAM has no effect on transactions which update the ledger. Extending SDAM functionality, as future work, to impact reading/writing to the ledger will require intercepting writes to CouchDB. One option is for writes to require full document visibility. If not, the full document must be reconstructed in the Proxy merging the write set into the full document read from the state database, before writing. Additionally, getState() requests would also need to be processed by SDAM with policies synced across all endorsing peers.

While such a demonstration is future work, SDAM's middleware approach can be used to provide seamless but controlled access to off-chain data from chaincode execution (as an oracle), as well as controlled access to on- or off-chain data for external applications.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009, (Retrieved on 2022-08-01). [Online]. Available: http://www.bitcoin.org/bitcoin.pdf

[2] V. E.Androulaki, A.Barger *et al.*, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proceedings of the 13th EuroSys Conference*, ser. EuroSys '18. Association for Computing Machinery, 2018.

[3] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 USENIX Annual Technical Conference (Usenix ATC 14)*, 2014, pp. 305–319.

[4] A. Barger, Y. Manevich, H. Meir, and Y. Tock, "A byzantine fault-tolerant consensus library for hyperledger fabric," in *IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2021, pp. 1–9.

[5] E. Commission, D.-G. for Justice, and Consumers, *The GDPR : new opportunities, new obligations : what every business needs to know about the EUs General Data Protection Regulation*. Publications Office, 2018.

[6] O. technical commitee, "Xacml: extensible access control markup language," 2005, (Retrieved on 2022-08-01). [Online]. Available: http: www.oasisopen.org/committees/xacml/repository

[7] P. R.Sandhu, "Access control: principle and practice," *IEEE communications magazine*, vol. 32, no. 9, pp. 40–48, 1994.

[8] S. Rouhani and R. Deters, "Blockchain based access control systems: State of the art and challenges," in *IEEE/WIC/ACM International Conference on Web Intelligence*, 2019, pp. 423–428.

[9] G. D.Chadwick *et al.*, "Permis: a modular authorization infrastructure," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 11, pp. 1341–1357, 2008.

[10] Microsoft, "Security descriptor description language," (Retrieved on 2022-08-01). [Online]. Available: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-dtyp/4f4251cc-23b6-44b6-93ba-69688422cb06

[11] C. Wirth and M. Kolain, "Privacy by blockchain design: a blockchain-enabled gdpr-compliant approach for handling personal data," in *Proceedings of 1st ERCIM Blockchain Workshop 2018*. European Society for Socially Embedded Technologies (EUSSET), 2018.

[12] P. K.Bhaskaran *et al.*, "Double-blind consent-driven data sharing on blockchain," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2018, pp. 385–391.

[13] K. C. Kouzinopoulos *et al.*, "Implementing a forms of consent smart contract on an iot-based blockchain to promote user trust," in *2018 Innovations in Intelligent Systems and Applications (INISTA)*. IEEE, 2018, pp. 1–6.

[14] J. Camilo *et al.*, "Blockchain-based consent manager for gdpr compliance," *Open Identity Summit*, 2019.

[15] K. Rantos, G. Drosatos, K. Demertzis, C. Ilioudis, and A. Papanikolaou, "Blockchain-based consents management for personal data processing in the iot ecosystem." *ICETE (2)*, vol. 298, 2018.

[16] K. N.Truong *et al.*, "Gdpr-compliant personal data management: A blockchain-based solution," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1746–1761, 2019.

[17] G. Zyskind, O. Nathan *et al.*, "Decentralizing privacy: Using blockchain to protect personal data," in *2015 IEEE Security and Privacy Workshops*. IEEE, 2015, pp. 180–184.

[18] R. S.Rouhani *et al.*, "Distributed attribute-based access control system using permissioned blockchain," *World Wide Web*, vol. 24, no. 5, pp. 1617–1644, 2021.

[19] H. Guo, E. Meamari, and C.-C. Shen, "Multi-authority attribute-based access control with smart contract," in *Proceedings of the 2019 international conference on blockchain technology*, 2019, pp. 6–11.

[20] I. EU, "Daml on fabric - technical document," 2020, (Retrieved on 2022-08-01). [Online]. Available: https://github.com/digital-asset/daml-on-fabric/blob/master/docs/DAML_on_Fabric_v2_Architecture.pdf

[21] A. T.Jose *et al.*, "Totem: Token for controlled computation: Integrating blockchain with big data," in *10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. IEEE, 2019, pp. 1–7.

[22] S. J.Gazsi *et al.*, "Vault: A scalable blockchain-based protocol for secure data access and collaboration," in *2021 IEEE International Conference on Blockchain (Blockchain)*, 2021, pp. 376–381.

[23] V. N.Bhamidipati *et al.*, "Claimchain: Secure blockchain platform for handling insurance claims processing," in *IEEE International Conference on Blockchain (Blockchain)*. IEEE, 2021, pp. 55–64.

[24] F. Loukil, K. Boukadi, R. Hussain, and M. Abed, "Ciosy: A collaborative blockchain-based insurance system," *Electronics*, vol. 10, no. 11, p. 1343, 2021.