Towards a Distributed Data-Sharing Economy

Samuel R. Cauvin, Martin J. Kollingbaum, Derek Sleeman, and
Wamberto W. Vasconcelos

Dept. of Computing Science, University of Aberdeen, U.K.
{r01src15, m.j.kollingbaum, d.sleeman, w.w.vasconcelos}@abdn.ac.uk

Abstract. We propose access to data and knowledge to be controlled through fine-grained, user-specified explicitly represented policies. Fine-grained policies allow stakeholders to have a more precise level of control over who, when, and how their data is accessed. We propose a representation for policies and a mechanism to control data access within a fully distributed system, creating a secure environment for data sharing. Our proposal provides guarantees against standard attacks, and ensures data-security across the network. We present and justify the goals, requirements, and a reference architecture for our proposal. We illustrate through an intuitive example how our proposal supports a typical data-sharing transaction. We also perform an analysis of the various potential attacks against this system, and how they are countered. In addition to this, we provide details of a proof-of-concept prototype which we used to refine our mechanism.

1 Introduction

Large scale data sharing is important, especially now, with more open societies of components such as Smart Cities [26,4] and the Internet of Things [1,12] creating data sharing ecosystems. Currently, data access policies tend to be managed centrally, which comes with a number of problems such as information ownership and reliance on a centralised authority.

In [17] the author suggests taking a "data-oriented view" and developing methods for treating access policies and data items as a single unit. This allows data to prescribe their own policies, which can be checked when the data is shipped around between data management systems. Such a proposal of tying policies directly to data is described by, e.g., [24] as *policy-carrying data* that allows the specification of fine-grained policies for data items. In this paper, we present policy-based data sharing concepts for distributed peer-to-peer networks of data providers and consumers. Our working hypothesis is that it is possible to (a) create a fully distributed mechanism to facilitate data sharing with security guarantees, and (b) to implement a fine-grained control over how data may be exchanged between stakeholders.

We propose access to data and knowledge to be controlled through fine-grained, user-specified explicitly represented policies. These policies are used to regulate data exchange in a peer-to-peer environment in which some peers have data which they want to provide (called Providers) and some peers have data which they want to acquire (called Requestors). Providers set policies that establish how their data can be accessed and by whom. These policies can be defined with different levels of granularity, allowing peers precise control over their data.

Our policies may express general regulatory statements such as, for example, "no drug records and medical records can be obtained by the same party", or more specific, such as "I will only provide 10 records to each person". Fine-grained policies allow stakeholders to have a more precise level of control over who, when, and how their data is accessed. We propose a representation for policies and a mechanism to control data access within a fully distributed system, creating a secure environment for data sharing.

These policies will be enforced by a distributed infrastructure of "policy decision points" (taking inspiration from the traditionally centralized XACML PDP architecture [8]) throughout the network. We regard a data exchange or sharing activity between peers (provider and requestor) as a transaction. Transactions are recorded and are an important means for checking policy compliance. During a data request, transaction records are taken into account to test whether a requestor complies with the policies specific to such a request and the data involved. Due to the distributed nature of making policy decisions at peer-to-peer network nodes, a requirement for encrypting information components to be exchanged for this decision process arises. We take inspirations from encryption concepts in distributed applications, such as CryptDB [20], BlockChain [21,11] and Bitcoin [18].

This paper focuses on providing a simple case example demonstrating the feasibility of this mechanism, including reasoning on encrypted data using the mechanism. Ours is a starting point from where more sophisticated policy representations and reasoning mechanism can be developed, with more expressive and flexible representations for policies to provide greater control to the user. The work presented here is an initial investigation into this kind of reasoning process which can be made more sophisticated, to address arbitrary deontic reasoning and more complex interactions.

Section 2 details a general example of a simple transaction between two parties and then discusses the key components and concepts within our solution. Section 3 provides an overview of the requirements and architecture of the system. Section 4 describes the detail of a transaction scenario, discussing how each part of the mechanism is involved in the process. Section 5 evaluates the mechanism's resistance to standard attacks. Section 6 discusses a proof-of-concept implementation of our solution. Section 7 provides an overview of related research. Section 8 discusses the limitations of our solution, provides overall conclusions, and outlines future work.

2 Policy Compliance

In our approach, so-called "transaction records" play an important role in whether any action related to sharing data is compliant with the policies relevant for this data. To illustrate how our mechanism performs a simple transaction, we consider a general case where two parties, a so-called "requestor" and a "provider", want to exchange data. Such a transaction represents a secure, tamperproof interaction between requestor and provider. Following this example we discuss transaction records (Section 2.1), numerical encoding of data elements (Section 2.2), and policies (Section 2.3) in more detail.

The requestor will be represented by R, the provider by P, and the data element by D. The transaction will proceed as follows:

1. Requestor R sends a data request for data D to provider P.

2. *P* processes this request, and if the provider possesses *D*, it will create a list of policies relevant to *D* or *R*. If any policy in this list prohibits sending *D* to *R* (regardless of transaction records), then the the data request will be denied, a transaction record will be generated and sent to *R*, and the process will terminate here. If not, *P* will send a message to *R* containing the policies associated with *D*.
3. *R* will reason on these policies to determine which transaction records are "relevant" (see Section 2.4). To achieve this, the mechanism loops through each policy and extracts a list of unique data elements referred to in the policy. At the end of this loop the list will contain each "relevant" data element (encoded as a number as discussed in Section 2.2).
4. The mechanism will then identify which of *R*'s transaction records are relevant using Algorithm 1 (in Section 2.4).
5. *P* receives records from *R* and needs to determine if any of the records prohibit the provision of *D*. For each of *P*'s policies the mechanism can process each record to determine if the data element is subsumed by a data element of *P*, and thus if the conditions of *P* hold. While processing, a cumulative total for each type of record will be kept. This total can be calculated without decrypting, as it requires only basic arithmetic on numerical entries. After processing all records, this total will be checked against the policy to determine if it holds or not. The order of policies is important, as earlier policies supersede later policies, that is as soon as a policy is triggered then the reasoning process can cease.
6. If this policy is a P (permit) policy, then sending *D* (the requested data and a record of the transaction, encrypted in a single package) to *R* is approved, and *D* will be sent from *P* to *R*.
7. *R* will decrypt the package, adding the transaction record to its records, and store the data. This single encrypted package is received by the mechanism, ensuring that the transaction record will be stored – as ignoring it will prevent receipt of data.

2.1 Transaction Records

Our policies relate data collections and events following the usual semantic of norms/policies (e.g., [19,22]), whereby events and their authors are explicitly represented (together with additional information such as time, location, duration, etc.) and used to check for (non-) compliance. In our proposal, events are named transaction records, and are stored encrypted within the information kept by each peer. Whenever a policy needs to be checked for its applicability, a subset of transaction records is retrieved from the encrypted storage, and used to compare the credentials/identification of the peer, assess the applicability to data elements currently available and verify if the conditions of our policies hold.

Transaction records are tuples of the form $\langle dataset, m \rangle$. The $dataset$ component refers to an ontological term, which is defined in one of the ontologies held by peers. Policies and transaction records refer to descriptions of data elements – these are labels describing, for instance, fields of a data base or names of predicates of an ontology [5]. We adopt a numeric representation for these labels, and rather than using, for instance, *nameOfClient* or *fatherOf* (to represent, respectively a field of a database or a predicate), we use a numeric encoding.

2.2 Numerical Encoding of Data Elements

Policy checking is performed on encrypted transaction records without decrypting them, and performing operations on encrypted numerical data is far easier than on encrypted string data. To facilitate this, we introduce a numbering scheme that represents such a hierarchy of concepts and sub-concepts, including the encoding of concept properties. For this, we assign to each level in the subsumption hierarchy found in an ontology a code out of the range of [0 .. 99]: when we use the notation $[00..99]_1$, $[00..99]_2$, $[00..99]_3$, then we are expressing that a concept hierarchy has three levels (where the subscripts indicate levels), and each concept can relate to a maximum of 100 (0 to 99) sub-concepts. By concatenating the level codes from a top-level concept to a particular sub-concept, we arrive at a unique code for each concept in a hierarchy. Consider the taxonomy below (with the encoded number at the start of each line):

- 010000 Prescriptions
  - 010100 Name
  - 010200 Drugs
  - 010300 Patient Notes
    - 010301 Other Medications
    - 010302 Other Conditions
  - 010400 Renewal Date
- 020000 DrugX
  - 020100 Trial Number
  - 020200 Patient Notes
    - 020201 Other Medications
    - 020202 Other Conditions
  - 020300 Recorded Side-effects
  - 020400 Treatment Effectiveness
- 030000 Vehicles
  - 030100 Motorcycles
    - 030101 Owner

For example, In the above taxonomy, Vehicles is the third top level concept $[03]_1[00]_2[00]_3$. A concept below that, Motorcycles, is $[03]_1[01]_2[00]_3$ which indicates it is the first sub-concept of Vehicles. The size of each level and total number of levels can be increased, but this will also increase the size of each encoded number. The subsumption relation between two encoded numbers allows us to capture "is-a" relationships among concepts of a taxonomy, as in $030100 \sqsubseteq 030000$.

2.3 Policies

Policies enforce how data can be shared within the network. Some are network-wide (e.g., "no drug records and medical records can be obtained by the same party"), while others can be specified by an individual provider (e.g., "I will only provide 10 records to each person"). These policies are stored by each peer locally.

We define our policies as follows:

Definition 1 (Policies). *A policy π is a tuple $\langle \mathbf{M}, \mathbf{I}, \mathbf{D}, \mathbf{P} \rangle$ where*

- $\mathbf{M} \in \{\mathsf{O}, \mathsf{F}, \mathsf{P}\}$ *is a deontic modality/operator, denoting an obligation (*O*), a prohibition (*F*) or a permission (*P*).*
- $\mathbf{I} \in \{id_1, \ldots, id_n\}$ *is a unique peer identifier*
- $\mathbf{D} \in \mathbf{T}$ *is a descriptor of a data element (cf. Def. 2)*
- $\mathbf{P} = L_1 \wedge \cdots \wedge L_m$ *is a conjunction of possibly negated literals (cf. Def. 3)*

Our policies above refer to descriptions of data elements – these are labels describing, for instance, fields of a data base or names of predicates of an ontology [5]. We adopt a numeric representation for these labels, and rather than using, for instance, *nameOfClient* or *fatherOf* (to represent, respectively a field of a database or a predicate), we use a numeric encoding. Our taxonomies are subsets of natural numbers with a subsumption relationship, and are thus defined:

Definition 2 (Taxonomy). *A taxonomy* $\mathbf{T} \subset \mathbb{N}$ *is a subset of natural numbers. We define a reflexive and transitive subsumption relation* $\sqsubseteq \subseteq \mathbf{T} \times \mathbf{T}$*, over a taxonomy* $\mathbf{T}$ *to define its structure.*

An example of a taxonomy is given in section 2.1.

Our policies allow the representation of *conditions* under which the policy should hold – this is what the component $\mathbf{P}$ of Def. 1 is meant for. We have designed a simple vocabulary of "built-in" tests which are relevant to our envisaged application scenarios, and these are defined below:

Definition 3 (Literals). *A literal* L *is one of the following, where* $\mathbf{D} \in \mathbf{T}$ *(a descriptor of a data element),* $\circ \in \{<, >, \leq, \geq, =\}$ *is a comparison operator, and* $n \in \mathbb{N}$ *is a natural number:*

- $noRec(\mathbf{D}) \circ n$ *– it holds if the number of retrieved instances of data element* $\mathbf{D}$ *satisfies the test "*$\circ n$*".*
- $lastReq(\mathbf{D}) \circ n$ *– it holds if the (time point of the) last retrieved instance of data element* $\mathbf{D}$ *satisfies the test "*$\circ n$*".*
- $lastAccess(\mathbf{D}) \circ n$ *– it holds if the (time point of the) last granted access to an instance of data element* $\mathbf{D}$ *satisfies the test "*$\circ n$*".*
- $\bot$ *and* $\top$ *represent, respectively, the vacuously false and true values.*

We adopt a simple account of time which can allow all events to be associated with a natural number.

In the remainder of our presentation, however, we make use of a "customised" version of policies, as these are more commonly used in our envisaged scenarios. We use the following shorthand:

$$\langle \mathbf{M}, \mathbf{I}, \mathbf{D}, (noRec(\mathbf{D}) < n \wedge noRec(\mathbf{D}') < n') \rangle \equiv \langle \mathbf{M}, \mathbf{I}, \mathbf{D}, n, \mathbf{D}', n' \rangle$$

Some examples of policies are as follows:

- $\pi_1 = \langle \mathsf{P}, id_1, 010000, 5, 0, 1 \rangle$, that is, peer id_1 is permitted to access 5 items of data element 010000; the remainder of the policy condition is idle (it imposes no further constraints).
- $\pi_2 = \langle \mathsf{P}, id_2, 020200, \infty, 0, 1 \rangle$, that is, peer id_2 is permitted to access unlimited (∞ stands for a very high natural number) items of data element 020200; the remainder of the policy condition is idle, that is, $noRec(0) < 1$ imposes no further restrictions.

– $\pi_3 = \langle \mathsf{P}, any, 010200, 5, 010000, 1 \rangle$ that is, any peer (denoted by the any identifier) is permitted to access 5 items of data element 010200; provided that they accessed less than 1 record of 010000.

This is a simple representation of policies which ignores the context of time and presents a simple example. The language of policies can be more expressive for the mechanism we are proposing. A more expressive language would allow more complex interactions between policies, which would also require a more complex reasoning process (we give potential expansions in Section 8).

In Def. 1 we discuss the notion of obligations, which can be thought of as deferred policies: actions to be taken (or not taken) after data has been received from a provider for a pre-specified period of time (or possibly indefinitely). For instance, an obligation could be defined that requires the requestor to provide 5 records of data element 010000 to the provider in exchange for 10 records of data element 020000.

We define in Equation 1 how to check if two data elements $\mathbf{D}$ and $\mathbf{D}'$ encoded in our numbering scheme of Section 2.2, are subsumed by one another. The definition makes use of two extra parameters, namely, BS which provides the size of each band (2, in the above example), and ZB which provides the number of zero-bands in $\mathbf{D}'$ (for instance, 020200 above has 1 zero-band $[02][02][00]$; calculating the number of zero bands is trivial for an unencrypted integer):

$$\mathbf{D} \sqsubseteq_{ZB}^{BS} \mathbf{D}' \text{ if, and only if, } \lfloor \mathbf{D}/10^{BS \times ZB} \rfloor = \lfloor \mathbf{D}'/10^{BS \times ZB} \rfloor \tag{1}$$

Each peer is provided a copy of the encoded ontology upon joining the network. If the ontology is too large, a subset could be provided containing concepts that the peer deals with and each transaction would provide the "vocabulary" of the requestor. In this way only a small amount of data is transferred when a new peer joins the network, but peers will slowly converge towards holding a complete ontology as transactions occur. Alternatively, the peer could be provided only with a URI; allowing them to download the full encoded ontology at any time.

Automatic encoding of the ontology is fairly trivial. The superclass-subclass relationships can be condensed into a simple tree structure; from this tree we can then count the maximum depth and maximum size at each depth to determine number of bands, and size of banding, respectively. This may take some time to complete, but this operation only has to be performed once on network initialisation.

Alternative numerical encoding mechanisms have been suggested that used ring theory, prime numbers, or multiples; however none seemed to precisely suit our needs. Mechanisms of this type have been widely explored [7,14], and these mechanisms could replace the one currently proposed. For the purposes of our research we wanted to create a simple example encoding, however others could have been used.

2.4 Finding Relevant Transaction Records

The mechanism itself chooses relevant transaction records to send to a provider, the peer is unable to intervene. The challenge is ensuring that the records held by a given peer are tamper-proof; however this is achieved by storing records in an encrypted format, using the numerical encoding in Section 2.2. Equation 1 allows identification of records

that match a specific concept (or one of its parents). Using this information, and a reasoning process that references both policies and what is known about the requested data, a subset of relevant records can be identified and sent to a provider. On receipt of these records, the provider must also reason with them to determine if they violate any policies.

The mechanism identifies relevant records by looping through each transaction record and performing a numerical comparison operation, without decrypting the data. Each transaction has an associated data element, which is compared to each data element in the policies for the current transaction using Equation 1. If the test is passed, then the transaction will be retained as a relevant record. When all records have been processed, all relevant records will be sent to *P*. This process is detailed in Algorithm 1.

Algorithm 1 Finds Relevant Transaction Records

Require: Π (a set of policies), $Records$ (a set of records)
Ensure: $RelevantRecords$ (a set of relevant records)
procedure FINDRELEVANTRECORDS()
$RelevantRecords \leftarrow \emptyset$
for all $R \in Records$ **do**
for all $\pi \in \Pi$ **do** ▷ Each data type referred to in policies
if $encodedComparison(\pi, R)$ **then** ▷ $encodedComparison$ refers to Equation 1
$RelevantRecords \leftarrow RelevantRecords \cup \{R\}$
end if
end for
end for
end procedure

The mechanism must be able to detect potential violations and protect against them; either by updating policies, anonymising part of the data, or rejecting the request. When making this decision the mechanism will check if the users identity allows them to access the data, if they have fulfilled all past obligations, and if the records they have provided would prohibit them from receiving the requested data.

When deciding whether to share data, both ends of the transaction are black-boxed; this prevents either the requestor or provider from tampering with records. The encrypted records and (unencrypted) policies get passed into a black-box mechanism, which returns a boolean value to indicate if the transaction can go ahead. If the transaction is denied, then an encrypted record will be returned to the requestor that contains a justification (and full proof of reasoning) as to why it was denied. This can then "bootstrap" the reasoning process next time; as this record will be sent (by the requestor) as a relevant record. The provider can then examine the proof and decide if it still applies, reducing reasoning overheads.

The other challenge is designing the selection procedure in the mechanism so that just the right amount of information can be shared; since peer-to-peer connections are opportunistic, the less information sent the better – however enough has to be sent to allow the provider to make an informed decision about whether to share.

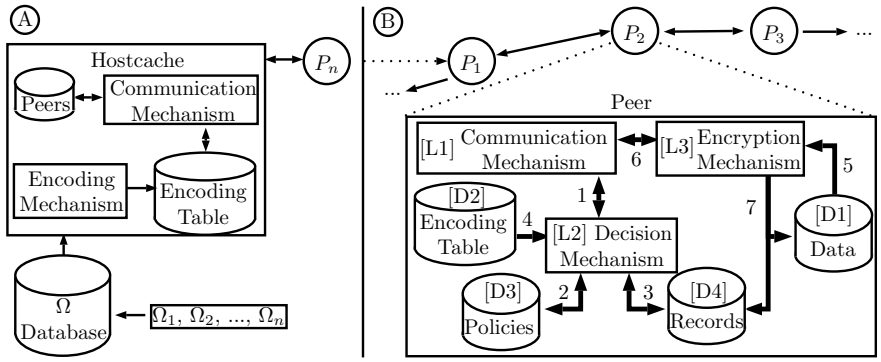

Fig. 1. Architecture

3 Requirements and Architecture

The hypotheses in Section 1 emphasise the aspects of the problem that we are concentrating on, and can be broken down further into the following requirements:

R1 To allow fine-grained (table and column level) control over data access policies.
R2 To ensure transaction records and data remain tamper-proof throughout the lifetime of a transaction.
R3 To allow operations to be performed on encrypted transaction records, without exposing those records to the user.
R4 To ensure that policies are enforced across the network and cannot be subverted to the advantage of an attacker.[1]

An architecture to meet these requirements is presented in Figure 1. The architecture above has two main components: the hostcache sub-architecture (A), and the peer sub-architecture (B).

It should be noted that our approach refers to data using database terms (tables, columns, and rows); however this is a specific case for our broader solution. While we assume that our mechanism will be used on data stored in a database, any knowledge base could be used instead.

The hostcache sub-architecture (A), which follows established peer-to-peer hostcache operations [3], has access to a collection of ontologies (obtained from many parties), which are input to the encoding mechanism. The encoding mechanism outputs the encoding table, which is a numerically encoded representation of the ontology (explained in Section 2.2). The hostcache also stores a collection of peer ids, each new peer that contacts the hostcache will have its peer id added to the collection. The hostcache processes requests from peers by generating ids, providing copies of the encoding table to peers and providing sets of potential neighbours to enquiring peers. The hostcache is a central element whose main functions are to generate ids for each peer, and to provide

[1] An attacker is any party (requestor, provider, or third party) who attempts to subvert the system.

a list of potential neighbours on request. It also handles the one-time encoding of the underlying ontology.

The peer sub-architecture (B) is a collection of storage and logic components. The encoding table (D2) on the peer is obtained directly from the hostcache, and is only referred to by the decision mechanism (L2). The decision mechanism is responsible for performing the decision operations discussed later in this document (whether to provide data, what records are "relevant"). The peer also holds data (D1 – the data which it provides), records (D4 – encrypted transaction records), and policies (D3 – policies detailing how data is shared, discussed in Section 2.3). There is also the communication mechanism (L1) which handles message processing (both receiving and sending), generating data requests, and invoking the decision mechanism. Lastly is the encryption mechanism (L3), which can encrypt and decrypt data and record packages (but not records themselves) received from the network (discussed further in Section 4). While not noted in the architecture, each peer also holds an encrypted id, issued by the hostcache, that confirms who they are.

Each component in the peer sub-architecture is needed to fulfil at least one requirement. Requirement 1 needs the decision mechanism and policies. Requirement 2 needs all components *except* policies. Requirement 3 needs the decision mechanism, encryption mechanism, encoding table, and records. Requirement 4 needs the communication mechanism, encryption mechanism, and encoding table.

We engineer the behaviour of peers so as to make contact with the hostcache, establish neighbours, and then go into a loop responding to messages and requesting data. The protocol adopts non-blocking message exchanges, that is, peers do not wait for replies (as communication is unreliable and these may never arrive or be delivered). The interactions in sub-architecture B are numbered to represent a rough interaction protocol, but as interactions occur in a distributed environment they cannot be considered as sequential operations on a single peer. More accurately, there are four (main) paths through the architecture diagram for two interacting peers. Peer id_1, upon receiving a data request from Peer id_2, will follow steps 1, 2, 4, 1 (from the annotated arrows of Fig. 1). Peer id_2 will follow steps 1, 2, 3, 1. Peer id_1 will follow steps 1, 2, 4 and then 5, 6. Peer id_2 will then follow steps 6, 7.

4 Illustrative Scenario

We illustrate our solution with a scenario in which we consider two parties: *P* (the provider) and *R* (the requestor). The provider is a research lab that developed DrugX, and tracks prescriptions of DrugX. The requestor is a health authority who regulate all prescriptions for the region they operate in attempting to counteract the side effects of Drug X. This example uses a subset of the encoding table from Section 2.2

The requestor wishes to get information on the trials carried out on DrugX by the provider, so sends a data request for ten 020000 (DrugX and subclasses) records. The provider checks its policies and finds nothing prohibiting the requestor's access to 020000, so the provider then sends the following (relevant) policies to the requestor:

- $\langle P, any, 010300, \infty, 020200, 1\rangle$ – Deny 010300 to anyone who has 020200

– $\langle P, any, 020200, \infty, 010300, 1\rangle$ – Deny 020200 to anyone who has 010300

The requestor loops through these policies and extracts the following data elements: 010300 and 020200. The requestor then has to check through their transaction records (the format is $\langle dataset, numberOfRecords\rangle$):

– $\langle 010100, 50\rangle$
– $\langle 010301, 50\rangle$
– $\langle 010302, 50\rangle$
– $\langle 010100, 10\rangle$
– $\langle 010200, 10\rangle$
– $\langle 010400, 10\rangle$

Each relevant data element is then compared with the records to determine its entailment, following Equation1, that is, $010301 \sqsubseteq^{BS}_{ZB} 010300$, and $010302 \sqsubseteq^{BS}_{ZB} 010300$ hold; none of the remaining cases hold.

For each pair $(\mathbf{D}, \mathbf{D}')$ we must test both $\mathbf{D} \sqsubseteq^{BS}_{ZB} \mathbf{D}'$ and $\mathbf{D}' \sqsubseteq^{BS}_{ZB} \mathbf{D}$, as the test will only capture if the first element is a subclass of the second. Applying both tests allows both relationships to be captured. Of the six records two of them are found to be relevant: $\langle 010301, 50\rangle$ and $\langle 010302, 50\rangle$. These records are now sent to the provider to be reasoned on to determine if they violate any of their policies. This process is similar to the process performed by the requestor, so we will not discuss it in as much detail. Performing the same basic loop the mechanism determines that Policy 2 (Deny 020200 to anyone who has 010300) holds for both records. At this point, the provider can do one of two things: the Data Request can be rejected (a justification record will be generated and sent to the requestor), or part of the requested data can be omitted. The latter will be used in this situation, as the policy only prevents a specific part (020200) of the requested data (020000) from being sent.

The provider then generates records for the current transaction ($\langle 020100, 10\rangle$, $\langle 020300, 10\rangle$, $\langle 020400, 10\rangle$), and assembles the result package (containing 10 records of 020100, 020300, and 020400). These are then encrypted together using the requestor's public key[2] and sent to the requestor. The requestor's mechanism receives this package and decrypts it using the requestor's private key. The "receipt" is added to the requestor's collection of transaction records and the mechanism returns the extracted data to the requestor, completing the transaction.

5 Analysis of our Solution

We evaluated our proposal by exploring many cases and concluded that there was no incentive for any of the participants to subvert the system, as it provided no advantages. Below we provide an analysis of our proposal against classic attacks.

– *Impersonation* – All peers, in order to join the network, must be given a unique encrypted id by the host cache. Ids cannot be falsified as only the hostcache has keys to generate these appropriately and the chances of falsifying ids coherently are very low.

[2] This is an extra security precaution, assuming that all peers have Public/Private Key pairs ensures that data can be sent across a peer-to-peer network securely.

- *Modification of policies* – Providers could modify policies during transactions, however doing so could cause them to receive irrelevant transaction records. These irrelevant records could cause them to make an incorrect decision to provide or withhold data, which they would have no incentive to do.
- *Modification of transaction records* – Transaction records cannot be tampered with as they are encrypted throughout exchanges; attempts to tamper with records would need to break the encryption mechanisms.
- *Man in the Middle* – Transaction records and data both travel encrypted. Policies are transmitted unencrypted, but it would be trivial to create a RSA-like encryption to transmit them. Man-in-the-middle can not access the data as it travels encrypted.
- *Denial of Service (DOS)* – Requiring a hostcache creates a vulnerability to DOS attacks, however this DOS would only affect new peers joining the network. Existing peers in the network would be able to function as normal. A DOS could also target individual peers, but this will not have a major effect on the rest of the network.
- *Subvert timestamp in records (Provider)* – This timestamp is generated by the mechanism, so cannot be altered. The provider could potentially alter it by garbling the record, but this would only serve to disadvantage them in the future.
- *Provider sends malformed record* – A malformed record will never be considered a "relevant" record, as it cannot be processed properly by the mechanism, so if they are sent, they will be ignored. There is no incentive for a peer to send malformed records. To prevent this record from remaining indefinitely a record purging functionality periodically scans the set of records and discards those elements which cannot be processed/parsed.
- *Requestor does not record transaction* – The mechanism forces transaction records to be stored. Providing the data and updating the set of records are two stages of an atomic operation carried out within the black-box mechanism.
- *Code tampering* – Tampering with code is impossible, as it is provided as a black-box.
- *Record fabrication* – Records could be fabricated, but the chances of producing anything meaningful are very low, since these have to be encrypted and the peers do not hold the keys or indeed have access to the encryption mechanism by itself.
- *Sybil Attack*[3] */Fake peer generation* – The only purpose to generating extra peers would be to generate fake records for yourself, but there is no benefit from having extra records as these will not make approval more likely, moreover, it could cause data requests to be rejected.
- *Data Modification* – After a peer receives data from another peer, that data is no longer under the control of the data provider. We propose a way of mitigating this by adding the concept of data "ownership". All data within the network can be stored in "packages" encrypted with the id of the original owner. Anyone can decrypt these packages to get the data, but they are only able to encrypt data packages with their own id. This means that the original source of the data is in no way associated with the data after it has been modified by a third party.

[3] A sybil attack [9] happens when one of the participants generates many fake ids to skew the balance of power in one's own favour, as in, for instance, voting.

Our solution incorporates a small amount of centralisation: a one-time check-in when connecting to the network, to aid with system functions. It may be possible to design a system where this is not the case, but we would have to make trade-offs (no verified identities, no shared encoding, cold-start issues, etc.) to achieve this. This minor centralisation ensures that no one "owns" all the data within the system, and also creates a robust network for data exchange; the only contact with a central authority (hostcache) is when a peer joins the network, after that no data is sent to the hostcache.

6 Proof of Concept Implementation

We investigated the design space by creating a proof-of-concept prototype[4] to perform the operations of single peer with a set of simulated neighbours. Our prototype does not implement full message passing, but does demonstrate the mechanism which we have described. The prototype is implemented in Java, so some definitions have been adapted to fit with object-oriented programming concepts. The cryptography implemented in the prototype is not a full encryption mechanism, but simulates one through the use of numerical objects that can have simple mathematical operations performed on them without exposing their value. If we ignore these adaptations, our implementation follows the peer architecture (part B of Fig. 1) faithfully.

To reflect the modularity of our architecture we have introduced features to customise the simulation using a number of parameters, currently specified as variables within the code. These parameters supply the (ontology) encoding table, data, records, and policies of each neighbouring peer. The simulation itself tracks a number of metrics to provide an analysis of performance. The policies implemented within our prototype follow our policy language provided in Def. 1, specifically they make use of the shorthand we describe in Section 2.3.

We have also performed a feasibility analysis by using this prototype to simulate an extended version of the scenario from Section 4. The scenario considers a single peer attempting to get data from four neighbours that each have data and policies. This simulation completes in a single cycle (each neighbour is queried for each desired data item once) with all of the requested data being received. The prototype tracks which peers provided the data, allowing this to be compared to their policies; through this we observed that policies were not violated at any point.

Using our prototype we tracked total number of messages sent between peers, total simulation time, and the minimum, average, and maximum size of messages. Message sizes are given in quantity of numbers transferred, with encrypted numbers taking twice the space, and each array adding an extra number as overhead. 40 messages were exchanged, with a minimum size of 1 (initial data request), an average size of 4.5, and a maximum size of 17. The simulation took a total of 10 milliseconds to complete, 2 milliseconds of which was the single cycle; the other 8 were network initialisation. This time could be considered inaccurate as we envisage our mechanism running on a large number of devices with little computing power; rather than the one powerful device that our simulation was run on.

[4] The source code for our implementation is `https://github.com/Glenugie/REND-Peer`

Implementing this prototype allowed us to locate and correct a number of inconsistencies in our mechanism. One such correction was to apply the encoded number comparison from Equation 1 in pairs to capture additional interactions.

7 Related Work

Our investigation taps onto many disparate areas such as Smart Cities [26,4], Internet of Things [1,12], BlockChain [21,11], bitcoin [18], and encryption [10,20]. Below we review the work that we consider most relevant.

This paper draws upon the techniques and methods reported in [19], but it has a significantly different focus, and most importantly, provides a distributed solution which will scale up and is resilient to many kinds of attacks. Our proposal extends the idea of combining data and access policies into one single computational entity with benefits such as increased control over how your data is used. There have been other research threads which also use the term "Policy Carrying Data" [23,25], which suggests similar concepts but without the focus on a distributed environment. They instead focus on maintaining data policies when the data, and policies, are uploaded to the cloud.

Berners-Lee makes a case for an online Magna Carta [16] to protect the openness and neutrality of the internet. The work being proposed here attempts to develop a mechanism to support the normative principles promoted in Berners-Lee's design [2].

Role Based Access Control (RBAC) could be seen as a similar line of work, though with a stronger focus on a controlled environment. While work has been pursued to begin addressing RBAC in a distributed environment [6,13,15,22], it has not been completely resolved.

One candidate for operations on encrypted data is homomorphic encryption schemes [10] which are applicable to our proposal. This method of encryption allows operations to be applied to encrypted data without decrypting. One limitation of this approach is that a data request must specify the amount of data to be retrieved, and the result will either be truncated or padded out. This method is semantically secure, i.e. given a cipher c that encrypts either m_0 or m_1, adversary α (when given the two choices) has probability of $\frac{1}{2} + \epsilon$ of guessing correctly. ϵ, called α's advantages should be negligible, else (informally) α has "broken" the semantic security.

Another candidate is CryptDB [20], though this is less suited to the required context. CryptDB relies on a trusted proxy to process a user's query and send it to the database management system (DBMS), which then returns the decrypted result. This seems problematic, as the proxy returns the result in a decrypted format (so, while the DBMS has not seen anything decrypted, the decrypted result could be intercepted between proxy and user).

We note a substantial overlap between our proposal and initiatives such as BlockChain[5] and Bitcoin[6]. BlockChain is a permissionless distributed database [21,11] based on the bitcoin protocol [18] that achieves tamper resistance by timestamping a hash of "batches" of recent valid transactions into "blocks". Each block references the prior

[5] `https://blockchain.info/`
[6] `https://bitcoin.org/en/`

timestamp, creating a cryptographically enforced chain. Blockchain requires either a group of always-on data-store nodes, or for every individual "peer" to store a copy of the full chain. There are important similarities between BlockChain and our proposal, but BlockChain is centralised in nature and has high storage requirements on data store nodes.

8 Conclusions, Discussions, and Future Work

We proposed a solution to enable the control of data through fine-grained, user-specified access policies. This solution was designed to operate in a peer-to-peer scenario in which some peers have data which they want to provide (called Providers) and some peers have data which they want to acquire (called Requestors). Providers can set "policies", i.e. rules which govern how their data can be accessed and also by whom. These policies will be enforced by mechanisms throughout the network.

For simplicity we assume that a fixed ontology is provided on network initialisation, and that this is then encoded by the hostcache. In the future, it would be possible for this to be extended to a dynamic ontology where each peer reports their sub-ontology on joining the network, which is then added to the master encoding table. This fixed ontology allows for the correct banding size and depth to be determined for the encoding. If the encoding is dynamic, one shortcoming is that it is then possible to run out of encoding space; this can be offset by choosing a high starting size but this will increase message size.

The mechanism and example that we have presented in this paper consider a simple case with a number of limitations which can be improved upon through a number of extensions, some of which we have sketched. Below we present some potential extensions of the proposed mechanism.

Our mechanism currently only allows a requestor to (implicitly) accept or reject policies within a transaction; if they reject the policies specified by the provider they simply do not send relevant records to the provider. In the future we could implement a "policy negotiation" phase, in which requestor and provider can propose and counter-propose policies to attempt to reach an agreement. For instance, the provider could propose an obligation which requires the requestor to provider 10 temperature readings. The requestor could counter-propose that they only provide 5 temperature readings. This process can continue until an agreement is reached, or either party withdraws.

We have considered the notion of obligations, which can be thought of as deferred policies: they are actions to be taken (or not taken) after data has been received from a provider for a pre-specified period of time (or possibly indefinitely). Obligations could also be set to expire when certain conditions are satisfied (not just time-related), for instance once an obligation has been triggered a certain number of times. We could consider a more complex system where multiple obligations can be attached to a single piece of data, and each obligation can be individually negotiated. Another possibility would be to allow obligations to be assigned to the provider (and not just the requestor). This would allow obligations such as "If I send data to you, then I am obliged to keep you updated if that data changes." This could either be proposed by the provider or requestor during negotiations.

Another extension is automated record purging and clean-up. Peers want to hold a minimal set of records (as they take up storage space), so there needs to be an operation to purge records that are no longer useful. Each peer would purge its own records periodically. Records with unfulfilled obligations will always be kept (another incentive to fulfil obligations, as otherwise your storage space will get filled quickly). Peers could also perform record compaction, merging equivalent records (for example, $\langle 010200, 20\rangle$ and $\langle 010200, 30\rangle$ become $\langle 010200, 50\rangle$).

Our implementation currently only addresses one peer, but we have already started looking into ways of simulating realistic P2P networks, using a technology such as PeerSim[7], which allows hundreds of thousands of peers to be simulated efficiently.

Our policy language is a starting point, and we have many possible extensions we would like to explore to provide finer-grained control but with adequate computational (performance) features. We have considered extensions that allow policies to have a time component. We also plan to provide reasoning mechanisms that allow users to see what the consequences of accessing a given piece of data are. As part of this we would also need to introduce more complex reasoning into the mechanism, allowing it to deal with complex interactions between policies. The mechanism could also be extended to allow policies to target groups of users; the present formalisation considers each peer to be an independent agent.

References

1. Atzori, L., Iera, A., Morabito, G.: The internet of things: A survey. Computer networks 54(15), 2787–2805 (2010)
2. Berners-Lee, T., Fischetti, M.: Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor. Harper San Francisco (1999)
3. Buford, J., Yu, H., Lua, E.K.: P2P networking and applications. Morgan Kaufmann (2009)
4. Caragliu, A., Bo, C., Nijkamp, P.: Smart cities in europe. Journal of Urban Technology 18(2), 65–82 (2011)
5. Chandrasekaran, B., Josephson, J.R., Benjamins, V.R.: What are ontologies, and why do we need them? IEEE Intelligent systems (1), 20–26 (1999)
6. Cheng, Y., Park, J., Sandhu, R.: A user-to-user relationship-based access control model for online social networks. In: Data and applications security and privacy XXVI, pp. 8–24. Springer (2012)
7. Curé, O., Naacke, H., Randriamalala, T., Amann, B.: Litemat: a scalable, cost-efficient inference encoding scheme for large rdf graphs. In: Big Data (Big Data), 2015 IEEE International Conference on. pp. 1823–1830. IEEE (2015)
8. Dhankhar, V., Kaushik, S., Wijesekera, D.: Securing Workflows with XACML, RDF and BPEL. In: Proceeedings of the 22Nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security. pp. 330–345. Springer-Verlag, Berlin, Heidelberg (2008), http://dx.doi.org/10.1007/978-3-540-70567-3_25
9. Douceur, J.R.: The sybil attack. In: Peer-to-peer Systems, pp. 251–260. Springer (2002)
10. Gentry, C.: Computing arbitrary functions of encrypted data. Communications of the ACM 53(3), 97–105 (2010)
11. Grigorik, I.: Minimum viable block chain. "https://www.igvita.com/2014/05/05/minimum-viable-block-chain/" (2014)

[7] http://peersim.sourceforge.net/

12. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems 29(7), 1645–1660 (2013)
13. Hansen, M.: Top 10 mistakes in system design from a privacy perspective and privacy protection goals. In: Privacy and Identity Management for Life, pp. 14–31. Springer (2011)
14. Harrison, J.: Theorem proving with the real numbers (1996)
15. Karjoth, G., Schunter, M., Waidner, M.: Platform for enterprise privacy practices: Privacy-enabled management of customer data. In: Privacy Enhancing Technologies. pp. 69–84. Springer (2002)
16. Kiss, J.: An online magna carta: Berners-Lee calls for bill of rights for web. The Guardian 12 (2014)
17. Landwehr, C.: Privacy research directions. Communications of the ACM 59(2), 29–31 (2016)
18. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
19. Padget, J., Vasconcelos, W.W.: Policy-carrying data: A step towards transparent data sharing. Procedia Computer Science 52, 59–66 (2015)
20. Popa, R.A., Redfield, C., Zeldovich, N., Balakrishnan, H.: CryptDB: Processing queries on an encrypted database (2012)
21. Postscapes: Blockchains and the internet of things. http://postscapes.com/blockchains-and-the-internet-of-things (Accessed: March, 2016)
22. Sackmann, S., Kahmer, M.: Expdt: A policy-based approach for automating compliance. Wirtschaftsinformatik 50(5), 366 (2008)
23. Saroiu, S., Wolman, A., Agarwal, S.: Policy-carrying data: A privacy abstraction for attaching terms of service to mobile data. In: Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications. pp. 129–134. ACM (2015)
24. Wang, X., Yong, Q., Dai, Y., Ren, J., Hang, Z.: Protecting Outsourced Data Privacy with Lifelong Policy Carrying. In: 10th IEEE International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing, HPCC/EUC 2013, Zhangjiajie, China, November 13-15, 2013. pp. 896–905 (2013), http://dx.doi.org/10.1109/HPCC.and.EUC.2013.128
25. Wang, X., Yong, Q., Dai, Y., Ren, J., Hang, Z.: Protecting outsourced data privacy with lifelong policy carrying. In: High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC). pp. 896–905. IEEE (2013)
26. Zheng, Y., Capra, L., Wolfson, O., Yang, H.: Urban computing: concepts, methodologies, and applications. ACM Transactions on Intelligent ... (2014), http://dl.acm.org/citation.cfm?id=2629592