SPECIAL

Scalable Policy-awarE Linked Data arChitecture for prIvacy, trAnsparency and compLiance

Deliverable D2.4

Transparency and Compliance Algorithms V1

Document version: V1.0

SPECIAL DELIVERABLE

Name, title and organisation of the scientific representative of the project's coordinator:

Ms Jessica Michel t: +33 4 92 38 50 89 f: +33 4 92 38 78 22 e: jessica.michel@ercim.eu

GEIE ERCIM, 2004, route des Lucioles, Sophia Antipolis, 06410 Biot, France

Project website address: http://www.specialprivacy.eu/

Project	
Grant Agreement number	731601
Project acronym:	SPECIAL
Project title:	Scalable Policy-awarE Linked Data arChitecture for prIvacy, trAnsparency and compLiance
Funding Scheme:	Research & Innovation Action (RIA)
Date of latest version of DoW against which the assessment will be made:	17/10/2016
Document	
Period covered:	M1-M23
Deliverable number:	D2.4
Deliverable title	Transparency and Compliance Algorithms V1
Contractual Date of Delivery:	28-02-2018
Actual Date of Delivery:	28-02-2018
Editor (s):	Sabrina Kirrane (WU)
Author (s):	Sabrina Kirrane (WU), Piero Bonatti (CeRICT), Javier D. Fernández (WU), Clemente Galdi (CeRICT), Luigi Sauro (CeRICT)
Reviewer (s):	Uroš Milošević (TF), Axel Polleres (WU)
Participant(s):	WU, TF, ERCIM, CeRICT
Work package no.:	2
Work package title:	Policy and Transparency Framework
Work package leader:	WU
Distribution:	PU
Version/Revision:	1.0
Draft/Final:	Final
Total number of pages (including cover):	37

Disclaimer

This document contains description of the SPECIAL project work and findings.

The authors of this document have taken any available measure in order for its content to be accurate, consistent and lawful. However, neither the project consortium as a whole nor the individual partners that implicitly or explicitly participated in the creation and publication of this document hold any responsibility for actions that might occur as a result of using its content.

This publication has been produced with the assistance of the European Union. The content of this publication is the sole responsibility of the SPECIAL consortium and can in no way be taken to reflect the views of the European Union.

The European Union is established in accordance with the Treaty on European Union (Maastricht). There are currently 28 Member States of the Union. It is based on the European Communities and the Member States cooperation in the fields of Common Foreign and Security Policy and Justice and Home Affairs. The five main institutions of the European Union are the European Parliament, the Council of Ministers, the European Commission, the Court of Justice and the Court of Auditors (http://europa.eu/).

SPECIAL has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731601.

Contents

1 Summary . . . 7

1 Compliance Checking **8**
1 Introduction . . . 8
2 The general structure of policies . . . 9
3 The compliance checking algorithm and its complexity . . . 10
4 Summary of results . . . 13

2 Distributed and Decentralised Frameworks **14**
1 Motivation . . . 14
2 Background information on blockchain . . . 15
2.1 The Bitcoin blockchain . . . 15
2.2 Alternative blockchain consensus mechanisms . . . 15
2.3 Blockchain platforms . . . 16
3 Ethereum and HyperLedger distributed ledger platforms . . . 17
3.1 Ethereum distributed ledger platform . . . 17
3.2 HyperLedger distributed ledger frameworks . . . 17
4 Solid decentralised social application platform . . . 18
5 Relevance for SPECIAL . . . 19

3 Leveraging the Big Data Europe Infrastructure **20**
1 The SANSA stack . . . 20
2 SPIRIT: leveraging the SANSA stack for transparency and compliance . . . 22
2.1 Transaction logs . . . 22
2.2 The SPIRIT dashboard . . . 23
2.3 Transaction log processing with SANSA . . . 23
3 Relevance for SPECIAL . . . 25
3.1 Knowledge distribution, representation & querying . . . 25
3.2 Reasoning . . . 25
3.3 Machine learning . . . 25
4 Summary of results and future outlook . . . 25

4 Fair Exchange **27**
1 Motivation . . . 27
2 Basic notions and desired properties . . . 28
3 Classification based on TTP and optimistic protocols . . . 29
4 Micali's optimistic fair-exchange protocol . . . 31

5 Optimistic fair-exchange protocols for large data transfers 32

List of Figures

3.1 The SANSA Stack Layers 21
3.2 SPIRIT architecture exemplifying the transparency use case 23

1 Summary

The aim of this deliverable is to dig deeper into open technical and research challenges in terms of personal data processing and sharing transparency, and compliance checking.

Towards this end, in this version of the deliverable we pay particular attention to: (i) compliance checking over SPECIAL's policies in *Chapter* 1 ; (ii) how we can leverage existing distributed ledgers in order to cater for the desired data processing and sharing requirements in *Chapter* 2; (iii) how we can leverage the Big Data Europe engine for semantic data processing on large-scale RDF data in *Chapter* 3; and (iv) the guarantees in term of non-repudiation that could be provided by existing fair exchange protocols in *Chapter* 4.

This deliverable builds upon technical requirements from *D1.3: Policy, transparency and compliance guidelines V1*, the SPECIAL policy language which is described in *D2.1: Policy Language V1*, and the SPECIAL transparency and compliance framework presented in *Deliverable D2.3: Transparency Framework V1*.

It is worth noting that the algorithms and protocols, either described specifically herein or offered as part of existing platforms and tools that we leverage in SPECIAL, will be refined based on practical experimentation in WP3, guided by the real world use cases described in *Deliverable D1.5: Use case scenarios V2*. This information will be used to inform later versions of this deliverable. Moreover, in later versions, we will also include additional information on the synthesis of policies for mined and aggregated data, and alternative security and robustness strategies.

Chapter 1

Compliance Checking

In this chapter we illustrate a complete and tractable structural subsumption algorithm for compliance checking over SPECIAL's policies. This reasoner applies to a fragment of OWL2-DL that slightly generalises the policy languages adopted by special (usage policies, business policies, and the partial GDPR formalisation). In particular it tolerates the creation of new policy attributes and new vocabulary terms, as well as attribute nesting at arbitrary levels. The chapter includes also a consistency checking algorithm for policies, useful for policy validation.

1 Introduction

Recall that in SPECIAL policies are encoded using a fragment of OWL2-DL and the main policy-related reasoning tasks are reduced to subsumption and concept consistency checking. Such tasks include - among others:

- *permission checking*: given an operation request, decide whether it is permitted;
- *compliance checking*: does a policy P_1 fulfill all the restrictions requested by policy P_2? (Policy comparison);
- *policy validation*: e.g. is the policy contradictory? Does a policy update strengthen or relax the previous policy?

Compliance checking is the predominant task in this project: the data usage policies of the industrial partners must be compared both with a (partial) formalisation of the GDPR itself, and with the consent to the usage of personal data granted by each of the *data subjects* whose data are collected and processed by the company (that is called *data controller* in the GDPR). The number of data subjects (and their policies) can be as large as the number of customers of a major communication service provider. Moreover, in the absence of explicit consent, some data cannot be stored, even temporarily; so some of the project's use cases consist in checking storage permissions against a stream of incoming data points, at the rate of hundreds of thousands per minute. Then one of the crucial project tasks is the development of scalable reasoning procedures for reasoning in the policy fragment of OWL 2.

In this chapter we illustrate this fragment and introduce a fast structural subsumption algorithm for scalable compliance checking. We introduce also a consistency checking algorithm for identifying contradictions in the policies (which is useful for validation purposes).

In the following we adopt the logical notation for description logics (DL for short), because it is way more compact than the alternative syntax for OWL, like XML syntax, Manchester syntax, etc. The following table shows the correspondence between the notation used in the other deliverables and the symbols used here:

`ObjectUnionOf`$(C_1 \ldots C_n)$	$C_1 \sqcup \ldots \sqcup C_n$
`ObjectIntersectionOf`$(C_1 \ldots C_n)$	$C_1 \sqcap \ldots \sqcap C_n$
`ObjectSomeValueFrom`$(AttrC)$	$\exists Attr.C$
`DatatypeSomeValueFrom`$(Attr$ `DatatypeRestriction`$($`xsd:integer` `xsd:mininclusive` x `xsd:maxinclusive` $y))$	$[x, y](Attr)$
`SubClassOf`$(C_1\ C_2)$	$C_1 \sqsubseteq C_2$
`DisjointClasses`$(C_1\ C_2)$	$\mathsf{disj}(C_1, C_2)$
`FunctionalObjectProperty`$(Attr)$	$\mathsf{func}(Attr)$
`ObjectPropertyRange`$(Attr\ C)$	$\mathsf{range}(Attr, C)$
`owl:Nothing`	$\bot$

Another difference between the logical terminology and the terms used in previous deliverables is that *properties* (i.e. attributes) are called *roles*.

2 The general structure of policies

The common structure of policies and vocabularies is captured and generalised by the following definition. Here the ontologies that define the vocabularies and the properties of policy attributes (reported in the appendix of D2.1) are called *knowledge bases*.

Definition 1 (Policy logic $\mathcal{PL}$) *A $\mathcal{PL}$* knowledge base *$\mathcal{K}$ is a set of axioms of the following kinds:*

- $\mathsf{func}(R)$ *where R is a role name or a datatype property;*
- $\mathsf{range}(S, A)$ *where S is a role (object property) and A a concept name;*
- $A \sqsubseteq B$ *where A, B are concept names;*
- $\mathsf{disj}(A, B)$ *where A, B are concept names.*

A simple $\mathcal{PL}$ concept *has the form:*

$$A_1 \sqcap \ldots \sqcap A_n \sqcap E_1 \sqcap \ldots \sqcap E_m \sqcap C_1 \sqcap \ldots \sqcap C_t \tag{1.1}$$

where each A_i is either a concept name or $\bot$, each E_j is an existential restriction $\exists R.C$ such that R is a role and C a simple $\mathcal{PL}$ concept, and each C_k is a constraint $[l, u](f)$. A (full) $\mathcal{PL}$ concept *is a union $D_1 \sqcup \ldots \sqcup D_n$ of simple $\mathcal{PL}$ concepts. $\mathcal{PL}$'s* subsumption queries *are expressions $C \sqsubseteq D$ where C, D are (full) $\mathcal{PL}$ concepts.*

3 The compliance checking algorithm and its complexity

Recall that checking whether a data controller's policy P_0 complies with a data subject policy (i.e. consent) P_1 amounts to checking whether the inclusion

$$P_0 \sqsubseteq P_1$$

is entailed by the ontologies for the policy language and the vocabulary (Appendix of D2.1), that will be denoted with $\mathcal{K}$.

SPECIAL's algorithm for performing this check is articulated in three phases:

Phase 1 Normalisation of the interval constraints $[\ell, u](f)$ in P_0;

Phase 2 Normalisation of the resulting policy P_0';

Phase 3 Subsumption checking of $P_0'' \sqsubseteq P_1$, where P_0'' is the result of step 2.

We will argue in the following that in practice the first two phases need to be done only once, when P_0 is deployed. Only Phase 3 shall be repeated for each compliance check. And even this phase can be optimised so that the actual reasoning is done only once, when either P_0 is modified or a new P_1 is introduced.

Phase 1

Phase 1 enforces the following property, that is needed for the correctness of Phase 3: *for all expressions $[\ell_0, u_0](f)$ occurring in P_0 and all $[\ell_1, u_1](f)$ occurring in P_1 (with the same f), either the two intervals $[\ell_0, u_0]$ and $[\ell_1, u_1]$ are disjoint, or the former is completely included in the latter.* This is achieved as follows. For all expressions $[\ell_0, u_0](f)$ occurring in P_0:

1. collect the expressions $[\ell, u](f)$ (with the same f) occurring in P_1;
2. select from those expressions the integers ℓ and u that belong to the interval $[\ell_0, u_0]$;
3. split $[\ell_0, u_0](f)$ using the selected integers;
4. move the new instances of $\sqcup$ introduced in step 3 to the top level.

Example 1 Suppose that the controller's policy P_0 contains $[1, 10](f)$ ($f = \texttt{durationInDays}$), while consent P_1 is the union of two simple policies containing $[0, 4](f)$ and $[7, 15](f)$, respectively.

From the two intervals $[0, 4]$ and $[7, 15]$ we select integers 4 and 7 (those that belong to $[1, 10]$).

Next we "break" $[1, 10]$ in points 4 and 7, obtaining $[1, 3]$, $[4, 4]$, $[5, 6]$, $[7, 7]$, $[8, 10]$; then we replace $[1, 10](f)$ with the logically equivalent expression

$$[1, 3](f) \sqcup [4, 4](f) \sqcup [5, 6](f) \sqcup [7, 7](f) \sqcup [8, 10](f)\,.$$

Consequently, the simple policy that contained the original expression $[1, 10](f)$ becomes as follows, where C represents the part concerning data categories, purpose, processing, and recipients:

$$C \sqcap \exists \texttt{hasStorage}.\big([1, 3](f) \sqcup \ldots \sqcup [8, 10](f)\big).$$

Finally, this policy is normalised by replacing it with

$$(C \sqcap \exists \texttt{hasStorage}.[1,3](f)) \sqcup \ldots \sqcup (C \sqcap \exists \texttt{hasStorage}.[8,10](f)).$$

Note that all the intervals in this new policy satisfy the desired all-or-nothing condition: for example, $[1,3]$ is contained in P_1's interval $[0,4]$ and disjoint from $[7,15]$; similarly for $[4,4]$; $[5,6]$ is disjoint from both; and so on. ∎

The *worst case complexity* of Phase 1 is $O(|P_0| \times |P_1|)$[1] because each simple policy contains at most one expression $[\ell, u](f)$, therefore the last step (where $\sqcup$ is moved to the top level) does not cause any combinatorial explosion. More generally, Phase 1 has the same asymptotic complexity for *every* constant bound on the number of expression $[\ell, u](f)$ that may occur in a simple policy.

In practice, however, we noticed that the number of different values ℓ and u occurring in policies is small and fixed, because retention limits derive from the applicable legislation, i.e. they are standard durations. Since duration values are known a priori, it is not necessary to gather them by scanning P_1, so the complexity of Phase 1 becomes simply $|P_0|$. Moreover, since Phase 1 does not depend on consent policies, it can be performed once when P_0 is deployed (and each time it is modified, but only on the new or updated parts).

Phase 2

This phase has two purposes: (i) compiling the knowledge contained in $\mathcal{K}$ (i.e. SPECIAL's ontologies) into the policy P_0, and (ii) detecting inconsistencies (e.g. attributes that are simultaneously required to belong to two disjoint classes).

Phase 2 applies the rewrite rules in Table 1.1 repeatedly until no more rules are applicable. A rule $X \rightsquigarrow Y$ means that the expression X in P_0 is replaced with Y. Rules can be applied in any order, the final result is the same.

1)	$\bot \sqcap D \rightsquigarrow \bot$	
2)	$\exists R.\bot \rightsquigarrow \bot$	
3)	$[l,u](f) \rightsquigarrow \bot$	if $l > u$
4)	$(\exists R.D) \sqcap (\exists R.D') \sqcap D'' \rightsquigarrow \exists R.(D \sqcap D') \sqcap D''$	if $\mathsf{func}(R) \in \mathcal{K}$
5)	$[l_1,u_1](f) \sqcap [l_2,u_2](f) \sqcap D \rightsquigarrow$ $[\max(l_1,l_2), \min(u_1,u_2)](f) \sqcap D$	if $\mathsf{func}(f) \in \mathcal{K}$
6)	$\exists R.D \sqcap D' \rightsquigarrow \exists R.(D \sqcap A) \sqcap D'$	if $\mathsf{range}(R,A) \in \mathcal{K}$ and A not a conjunct of D
7)	$A_1 \sqcap A_2 \sqcap D \rightsquigarrow \bot$	if $A_1 \sqsubseteq^* A_1'$, $A_2 \sqsubseteq^* A_2'$, and $\mathsf{disj}(A_1', A_2') \in \mathcal{K}$

Table 1.1: *Normalisation rules w.r.t. $\mathcal{K}$. Intersections are treated as sets (the ordering of conjuncts and their repetitions are irrelevant).*

The *worst case complexity* is $O(|P_0|^2 \times |\mathcal{K}|)$. This phase depends only on P_0 and $\mathcal{K}$, so it needs to be executed only when P_0 is deployed or updated, and when $\mathcal{K}$ is modified (e.g. in order to add new terms to the vocabularies).

We conclude this section by pointing out that the normalisation rules in Table 1.1 can be used as a *policy validation* method, to check that a $\mathcal{PL}$ concept (policy) is satisfiable, as specified by the next result:

[1] The expression $|X|$ denotes the size of X.

Proposition 1 *Let $\mathcal{K}$ be a $\mathcal{PL}$ knowledge base. A simple $\mathcal{PL}$ concept C is unsatisfiable w.r.t. $\mathcal{K}$ iff C is rewritten to $\bot$.*

Phase 3

In Phase 3 we are given two policies $P_0 = P_{0,1} \sqcup \ldots \sqcup P_{0,n}$ and $P_1 = P_{1,1} \sqcup \ldots \sqcup P_{1,m}$, and we have to check whether $\mathcal{K}$ entails $P_0 \sqsubseteq P_1$.

At this stage, P_0 has already been normalised by Phase 1 and Phase 2. This is necessary for the algorithm STS below to be correct and complete (i.e. to return "true" if and only if the entailment is valid).

The main algorithm is Algorithm 1. It checks whether each $P_{0,i}$ in P_0 is subsumed by some $P_{1,j}$ in P_1, using algorithm STS, that applies only to *simple* $\mathcal{PL}$ concepts.

Algorithm 1: `main`$(\mathcal{K}, P_0 \sqsubseteq P_1)$

Input: $\mathcal{K}$ and a $\mathcal{PL}$ inclusion $P_0 \sqsubseteq P_1$ where P_0 is normalised
Output: `true` if $\mathcal{K} \models P_0 \sqsubseteq P_1$, `false` otherwise
Note: By $C = C' \sqcap C''$ we mean that either $C = C'$ or C' is a conjunct of C (possibly not the 1st)

```
begin
    foreach P_{0,i} in P_0 do
        subsumed := false
        j := 1
        repeat
            if STS(K, P_{0,i} ⊑ P_{1,j}) then subsumed := true
            j := j+1
        until subsumed = true OR j > m
        if subsumed = false then return false
    end
    return true
end
```

We are only left to illustrate STS. It makes use of a relation $\sqsubseteq^*$ that denotes the reflexive and transitive closure of $\{(A, B) \mid (A \sqsubseteq B) \in \mathcal{K}\}$.

Algorithm 2: STS$(\mathcal{K}, C \sqsubseteq D)$

Input: $\mathcal{K}$ and an elementary $C \sqsubseteq D$ where C is normalised
Output: `true` if $\mathcal{K} \models C \sqsubseteq D$, `false` otherwise
Note: By $C = C' \sqcap C''$ we mean that either $C = C'$ or C' is a conjunct of C (possibly not the 1st)

```
begin
    if C = ⊥ then return true
    if D = A, C = A' ⊓ C' and A' ⊑* A then return true
    if D = [l, u](f) and C = [l', u'](f) ⊓ C' and l ≤ l' and u' ≤ u then return true
    if D = ∃R.D', C = (∃R.C') ⊓ C'' and STS(K, C' ⊑ D') then return true
    if D = D' ⊓ D'', STS(K, C ⊑ D'), and STS(K, C ⊑ D'') then return true
    else return false
end
```

We can prove the following result:

Theorem 2 *Algorithm 1 is correct and complete, that is, for all $\mathcal{PL}$ subsumptions $P_0 \sqsubseteq P_1$, $\texttt{main}(\mathcal{K}, P_0 \sqsubseteq P_1) = \texttt{true}$ iff $P_0 \sqsubseteq P_1$ is implied by $\mathcal{K}$.*

The *worst case complexity* of Algorithm 1 is $O(|\mathcal{K}| \cdot |P_0| \cdot |P_1|)$. Both $\mathcal{K}$ and the policies P_0 and P_1 have limited size, while there can be a very large number of consent policies P_1. Since there are no cross-dependencies between compliance checks with respect to different consent policies, *the time required for global compliance checking* (i.e. comparing the controller's policy P_0 with all the data subjects' consent) *grows linearly with the size of the consent repository*.

Several optimisations are possible; they will be illustrated in future deliverables.

4 Summary of results

We introduced a 3-phase algorithm for compliance checking and policy consistency checking. The first two phases pre-process the data controller's policy P_0; they need to be executed only when P_0 or the vocabularies are deployed or updated. Phase 2 detects also whether the simple policies in P_0 contain errors that make them inconsistent (such as attributes that are required to belong to two disjoint classes, for example, and wrong term choices, like data categories assigned to the hasPurpose attribute).

All the phases are tractable. Phase 1 takes time $O(|P_0| \times |P_1|)$ (where P_1 is a consent policy). Phase 2 takes time $O(|P_0|^2 \times |\mathcal{K}|)$ (where $\mathcal{K}$ contains the ontologies reported in the appendix of D2.1). Phase 3 (the actual policy comparison) takes time $O(|\mathcal{K}| \cdot |P_0| \cdot |P_1|)$.

The ontology $\mathcal{K}$ and P_0 have limited size, as well as each individual consent policy P_1. However, the number of consent policies may be very large. The cost of compliance checking over all consent policies grows linearly with the consent repository size.

The algorithm works not only for the specific policy language and vocabularies reported in D2.1, but also for all the policies and ontologies with the same structure, that are formally defined in Definition 1.

Chapter 2

Distributed and Decentralised Frameworks

The SPECIAL platform needs to be able to generate an immutable record of data processing and sharing events, including the original consent obtained from the data subject and any changes to consent thereafter. Information relating to data processing and sharing events could be stored in one or more distributed chains that are accessible via API's. These chains may include a hash of the data and a pointer to the actual data, which will be stored off chain in an encrypted format. In this chapter we investigate the different distributed ledgers that could potentially be used to realise said requirements. In addition, we explore the role played by decentralised platforms such as Solid, which relies heavily on World Wide Web Consortium (W3C) standards.

1 Motivation

Before discussing distributed ledger technology and how it can be used to develop decentralised applications, let us recall some of the desiderata for the transparency ledger (cf. D1.3 and [10]):

Completeness: All data processing and sharing events should be recorded in the ledger.

Confidentiality: Both data subjects and companies should only be able to see the transactions that involve their own data.

Correctness: The records stored in the ledger should accurately reflect the processing event.

Immutability: The log should be immutable such that it is not possible to go back and reinvent history.

Integrity: The log should be protected from accidental and/or malicious modification.

Interoperability: The infrastructure should be able to transcend company boundaries, in the sense that the data subject should be able to easily combine logs that they get from multiple companies.

Non-repudiation: When it comes to both data processing and sharing events it should not be possible to later deny that the event took place.

Rectification & Erasure: It should be possible to rectify errors in the stored personal data and/or delete data at the request of the data subject.

Traceability: In the case of processing it should be possible to know about any previous processing of the data. As such it should be possible to link events in a manner that supports traceability of processing.

2 Background information on blockchain

In this section, we provide an overview of existing blockchain platforms that could potentially be used in SPECIAL, we first provide some background information on the bitcoin blockchain platform, commonly used consensus mechanisms, and introduce the broader blockchain landscape.

2.1 The Bitcoin blockchain

Bitcoin [18], the first *decentralised* digital currency, is a peer-to-peer (P2P) system which is able to function without the need for a *centralised* banking authority. Essentially all transactions are stored in a public *distributed* ledger called a blockchain.

Bitcoin wallets are software program that are used to store virtual currency known as Bitcoins. In essence, Bitcoins are private keys that are recorded in the wallet software. When Party A wishes to send Bitcoins to Party B they use their wallet software to generate a transaction, which indicates the payee, the payer and the amount to be transferred. The transaction is subsequently broadcast to the P2P network.

The transaction is picked up, along with other transactions, by *Miners* who attempt to confirm the transactions by generating a new block to be added to the blockchain ledger. Essentially miners compete to find the solution to a cryptographic puzzle (i.e. a cryptographic hash with a specified number of proceeding zeros), which is difficult to solve yet easy to verify. *Proof-of-work (PoW)* refers to the solution to the cryptographic puzzle produced by the miners. When the miners solve the cryptographic puzzle they broadcast their PoW to the network. The nodes participating in a P2P system verify the PoW and the new block is subsequently added to the blockchain ledger. The miner that solves the cryptographic puzzle first is rewarded for their efforts via freshly minted Bitcoins and transactions fees associated with any transaction that is part of the newly mined block. Generally speaking a new block is added every 10 minutes, this can be controlled by changing the number of proceedings zeros the hash must exhibit).

Although the Bitcoin blockchain functions as a cryptocurrency system, the technology itself is designed to support features such as *transparency*, *pseudonymity*, *immutability* and *auditability*, and cryptographic guarantees in terms of *authenticity*, *integrity* and *non-repudiation*. As such, blockchain technology could be used for an array of different applications that require *distributed ownership* of data.

2.2 Alternative blockchain consensus mechanisms

In the following, we provide a summary of the three primary consensus mechanism algorithms that are used in well-know blockchain platforms.

Proof-of-work (PoW) is the original consensus algorithm used in the bitcoin blockchain to produce new blocks. Essentially, minders compete against each other to solve a cryptographic puzzle that is difficult to solve and easy to prove. Such puzzles are often known as CPU cost functions, computational puzzles or CPU pricing functions.

Proof of stake (PoS) is an alternative to PoW, that is used to achieve consensus between the nodes participating in a blockchain. In contrast to PoW, the PoS algorithm chooses the creator of the next block in a deterministic manner, based on their wealth, which is commonly referred to as their stake. Unlike PoW, there are no rewards for creating new blocks. PoS miners (commonly known as forgers) are however able to collect the transaction fees. A major benefit of PoS is the reduction in energy needed to create a new block.

Proof of Elapsed Time (PoET) uses a trusted execution environment to ensure that blocks are produced randomly, without the high demand for energy. Essentially random numbers are used to determine how long nodes have to wait before they are permitted to generate a new block. The trusted execution environment is in turn responsible for generating a proof that the waiting time has elapsed, which can easily be checked by the other nodes.

Additional, details on the above consensus protocols can be found in [11].

2.3 Blockchain platforms

According to a recent blockchain landscape survey conducted by Baliga [7], existing blockchain platforms can broadly be grouped according to five core categories. Here we provide a high level overview of said categories, including details of one or two popular platforms in the space. Another useful source of information is a recent blog by Rohas Nagpal, which provides an overview of 17 blockchain platforms[1].

Meta-data platforms use the existing Bitcoin blockchain to transfer information in the form of meta-data which is encoded using the OP_RETURN instruction. ColoredCoins[2] is just one example of a meta-data platform.

Financial applications are specifically designed for applications in the financial domain, e.g., payments, equity, foreign exchange, to name but a few. Well known financial blockchain platforms include Hyperledger[3] and Ripple[4].

Smart contract platforms provide functionality that enables code to be executed on the blockchain, in the form of Turing complete languages that can be used to express complex logic. Ethereum[5] and Hyperledger[6] are probably the most well known platforms in this space.

Consortium/Enterprise platforms are designed for corporate collaboration and usually involve a distributed consensus protocol instead of computationally expensive PoW algorithms. Openchain[7] which targets Enterprises offers a robust, secure and scalable platform. An alternative platform is provided by Hyperledger.

[1] https://medium.com/blockchain-blog/17-blockchain-platforms-a-brief-introduction-e07273185a0b
[2] http://coloredcoins.org/
[3] https://www.hyperledger.org/
[4] https://ripple.com/
[5] https://www.ethereum.org/
[6] https://www.hyperledger.org/
[7] https://www.openchain.org/

Sidechain platforms are blockchains in their own right that are connected to the Bitcoin blockchain via a two-way peg or as an anchored chain. Such platforms can send Bitcoins back and forth between the sidechain and the main blockchain. Additionally, using sidechains developers can attach new features to a separate chain. Both Openchain and Hyperledger have offerings in this space.

3 Ethereum and HyperLedger distributed ledger platforms

In this paper, we dig deeper into two of the most popular blockchain community efforts, namely Ethereum which is co-ordinated by a Swiss nonprofit organisation called the Ethereum Foundation[8], and HyperLedger which is hosted by The Linux Foundation[9].

3.1 Ethereum distributed ledger platform

Ethereum is an open source blockchain platform designed to support the development of decentralised applications in an adaptable and flexible manner. The currency used in Ethereum is known as Ether. The first version of Ethereum, known as the Frontier release, was a beta release designed to enable developers to experiment with decentralised applications. The second major version of the platform, known as the Homestead release, was the first production release.

In essence, Ethereum is a platform that can be used to develop different types of decentralised blockchain applications. Every node in the networks runs an Ethereum Virtual Machine (EVM). Like in Bitcoin, miners are responsible for generating new blocks and adding them to the blockchain based on a (PoW) protocol. However, while the Bitcoin blockchain simply contains transactions, the Ethereum blockchain contains accounts i.e. the Ethereum blockchain tracks the state and the exchange of information between accounts.

Ethereum distinguishes between two types of accounts: Externally Owned Accounts (EOAs) controlled via private keys, and Contract Accounts (CAs) controlled by contract code which is activated by an EOA i.e. the contract executes when a transaction is sent to the contract account. The contract code, which is usually referred to as a *smart contract* is written in a high level language known as Solidity which is known to be Turing-complete.

3.2 HyperLedger distributed ledger frameworks

At the time of writing HyperLedger incubates and promotes several different frameworks and tools. A highlevel overview of the platforms currently incubated by HyperLedger is provided below:

Hyperledger Burrow is a permissioned, smart contract, PoS client that can be used to *connect to a variety of blockchain networks*. Key features include: permissioned (private) networks, smart contracts, security and data privacy.

Hyperledger Fabric is a foundational platform that can be used to *develop modular applications* via its pluggable architecture. Key features include: identity management, privacy and confidentiality, chaincode functionality (i.e. smart contracts), modular design, and efficient processing.

[8] `https://www.ethereum.org/`
[9] `https://www.hyperledger.org/`

Hyperledger Sawtooth is a modular platform for building, deploying, and *running distributed ledgers*. Key features include: permissioned (private) networks, parallel transaction execution, event broadcasting, support for Ethereum smart contracts, and pluggable consensus algorithms (PoET, PoET simulator and a random-leader algorithm).

Hyperledger Iroha is a business blockchain framework designed to be simple and easy *integrate into corporate environments*. Key features include: support for complex assets (such as currencies or indivisible rights, serial numbers, patents, etc.), user and permission management, and support of business rules.

Hyperledger Indy is a distributed ledger that was purpose-built for managing *decentralised identity*. Key features include: support of independent digital identities, and adherence to best practices in terms of key management and cybersecurity.

Existing tools, which are designed to work across different Hyperledger frameworks, can be summarised as follows:

Hyperledger Cello focuses on reducing the effort required for managing blockchains. It is designed to work on top of several infrastructures, such as baremetals, virtual clouds (e.g., virtual machines, vsphere Clouds), and container clusters (e.g., Docker, Swarm, Kubernetes).

Hyperledger Composer makes it easier to build blockchain business networks and to develop smart contracts, by providing business-centric abstractions as well as sample apps that cater for modelling and integration.

Hyperledger Explorer is an open source browser which can be used to view and manage blockchain activity. Additionally it is possible to get access to information via the provided REST APIs.

4 Solid decentralised social application platform

An alternative decentralised platform known as Solid[10] is currently under development at Massachusetts Institute of Technology (MIT). Solid (derived from "social linked data") is a modular and extensible platform for building decentralised social applications based on Linked Data principles. Key features include: reliance on W3C standards and protocols, user and permission management, support for both Linked Data resources (RDF in the form of JSON-LD, Turtle, HTML+RDFa, etc..) and non Linked Data resources.

Identity & Profiles Web Identity and Discovery (WebID)[11], which is supported by a W3C community group, is a mechanism used to uniquely identify and authenticate a person, company, organisation or other entity, by means of a URI. In solid, WebIDs are used to uniquely identify actors. While, WebID profile documents are used to specify security credentials and preferences.

[10] https://solid.mit.edu/
[11] https://www.w3.org/2005/Incubator/webid/spec/identity/

Authentication & Access Control WebID-TLS[12] describes a protocol that can be used to authenticate a user without the need to have it signed by a trusted Certificate Authority. WebAccessControl (WAC) is an RDF vocabulary and an access control framework, which demonstrates how together WebID and access control policies specified using the WAC vocabulary, can be used to enforce distributed access control.

Consent Representation Solid supports two kinds of resources: Linked Data resources (RDF in the form of JSON-LD, Turtle, HTML+RDFa, etc) and everything else (binary data and non-linked-data structured text). Resources can be grouped using Linked Data Platform[13] containers.

Protocols Currently supported protocols include simple REST APIs, that support *Create, Read, Update, and Delete (CRUD)* operations, and WebSockets APIs that support *Publish, and Subscribe* notifications.

5 Relevance for SPECIAL

The SPECIAL transparency and compliance framework needs to be realised in the form of a scalable architecture, which is capable of providing transparency beyond company boundaries. In this context, it would be possible to leverage existing blockchain platforms and the Solid decentralised social application platform. However, each have their own strengths and weaknesses.

The distributed ledger technology offered by existing blockchain providers provide certain guarantees with respect to *immutability*, *integrity*, *non-repudiation*, *interoperability* and *traceability*. When it comes to the development of decentralised applications, Solid is designed to work with Linked Data and makes very good use of existing web standards in that space, making it particularly attractive for *interoperability*. In terms of the development of decentralised applications based on blockchain technology, both Ethereum Homestead and Hyperledger Fabric are production ready, however *Hyperledger* seems to offer more in terms of fitting with existing Enterprise applications.

However, in the context of SPECIAL the other platforms offered by *Hyperledger* could potentially also be of benefit. More specifically, *Hyperledger Iroha* stands out because of its support for permission management (i.e. *confidentiality*) and business rules. While, *Hyperledger Indy* would be interesting from an identity management perspective as data subjects could potentially use such an infrastructure to generate self sovereign identities [1] that they have full control over and only they can link across company boundaries. From an interoperability perspective *Hyperledger Burrow* is a clear favorite as it is specifically designed for this purpose. Given that companies could potentially use different ledger platforms to record data processing and sharing events, *Hyperledger Burrow* could be used to manage the interfaces between said platforms.

Ongoing work by the SPECIAL team is looking into how we combine several of the aforementioned technologies so that we can benefit from each of their strengths. The goal of the work is to identify the strengths and weaknesses of the existing platforms, to discuss how the various technologies can be combined and finally to assess what guarantees these combinations offer in terms of usage control transparency and compliance requirements.

[12] https://www.w3.org/2005/Incubator/webid/spec/tls/
[13] https://www.w3.org/TR/ldp/

Chapter 3

Leveraging the Big Data Europe Infrastructure

The work described in this chapter has been performed in close collaboration with the BDE members, Jens Lehmann[1] and Patrick Westphal[2].

In deliverables *D1.3 Policy, transparency and compliance guidelines V1*, *D1.4 Technical Requirements V1* and *D2.3 Transparency Framework V1* we motivated the need for a scalable Big Data infrastructure to support transparency with respect to data processing and compliance with respect to usage control policies. In particular, given the volume of events and policies that will need to be handled, the scalability of event data processing is a major consideration.

In the following, we discuss how existing Big Data processing techniques can be used for consent management, transparency and compliance. In particular, we build on top of the Big Data Europe (BDE) platform [4], which provides the foundational basis for large-scale integration and processing of data that is necessary to deliver such services.

We first introduce the semantic data processing stack (SANSA), which we use as an architectural basis for SPECIAL transparency and compliance. Then, we present SPIRIT, our proposed extension of SANSA to provide transparency with respect to personal data processing.

1 The SANSA stack

The current big data landscape provides a plethora of tools and frameworks covering a variety of methods and techniques for processing huge amounts of data in a distributed cluster of machines. In terms of actual general purpose big data processing frameworks, Apache Hadoop[3], Apache Spark[4], and Apache Flink[5] are examples of established open source projects. However, none of those frameworks provide built-in support for processing *big semantic data* e.g. to load and store RDF data as a uniform data format, along with ontological inference support and analytics.

SANSA[6] is an open source[7] semantic data processing stack that supports distributed compu-

[1] Smart Data Analytics Group, University of Bonn, DE. `jens.lehmann@cs.uni-bonn.de`

[2] Institute for Applied Informatics (InfAI), University of Leipzig, DE. `patrick.westphal@informatik.uni-leipzig.de`

[3] Apache Hadoop, `http://hadoop.apache.org/`

[4] Apache Spark, `http://spark.apache.org/`

[5] Apache Flink, `http://flink.apache.org/`

[6] SANSA Stack home page, `http://sansa-stack.net`

[7] SANSA Stack on GitHub, `https://github.com/SANSA-Stack`

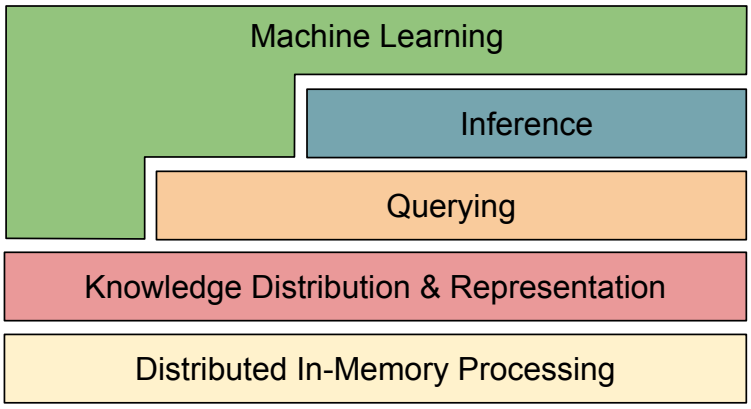

Figure 3.1: The SANSA Stack Layers

tations on large-scale RDF data. Being built on top of the two prevalent distributed in-memory big data processing frameworks Apache Spark and Apache Flink, SANSA provides performance, scalability and fault tolerance. SANSA is designed as a stack of individual layers, each covering specific aspects of RDF data processing and analytics (cf. Figure 3.1) and the corresponding description below:

Knowledge Distribution and Representation This layer provides a means to read and write RDF and OWL files, which can then serve SPECIAL needs in terms of data models, as specified in deliverables *D1.3: Policy, transparency and compliance guidelines V1* and *D2.3: Transparency Framework V1*. In terms of data structures and programming interfaces SANSA follows the common and accepted representations of Apache Jena[8] and the OWL API[9]. Hence, the RDF and OWL data is provided as a distributed collections of Apache Jena triples and OWL API axioms, respectively.

Querying The query layer comprises functionality for searching, exploring and extracting information from big semantic data. The main W3C standard for performing queries on RDF is SPARQL[10]. SANSA supports two interfaces for executing such SPARQL queries: (1) Within an Apache Spark/Flink program, and (2) via an HTTP SPARQL endpoint. In both cases the actual queries will eventually be translated into lower level Apache Spark/Flink programs and executed on the Knowledge Distribution and Representation Layer. To better exploit parallelism and data locality different partitioning strategies are explored.

Inference Apart from actual data-level assertions or facts, the Semantic Web technology stack also provides a means to express schema or ontological knowledge. The respective W3C standards, RDFS[11] and OWL, allow the inherent semantics (or parts thereof) to be express as rules, which can be used to infer new knowledge. This *forward chaining* process can be applied,

[8] Apache Jena, `http://jena.apache.org/`
[9] OWL API, `https://owlcs.github.io/owlapi/`
[10] SPARQL, `https://www.w3.org/TR/2013/REC-sparql11-query-20130321/`
[11] RDFS, `https://www.w3.org/TR/2014/REC-rdf-schema-20140225/`

e.g. to materialise all rule-based inferences, or to prepare a given knowledge base for further processing. In contrast *backward chaining* techniques infer new knowledge starting at a given 'goal', which can be a (set of) RDF triple(s). SANSA currently supports different profiles for rule-based forward chaining, while support for rule-based backward chaining is currently in development. Besides focusing on fixed profiles like RDFS or OWL Horst [21], SANSA allows an arbitrary set of rules and is able to compute an efficient execution plan by generating and analysing a rule dependency graph. Hence, users can adjust the trade-of between expressivity and performance, and furthermore introduce custom rules, representing e.g. business policies.

Machine Learning This layer is a collection of machine learning algorithms that can directly work on RDF triples or OWL axioms. In addition to machine learning algorithms that work on feature vectors, SANSA makes use of the graph structure and the inherent semantics of RDF and OWL data. The algorithms implemented thus far cover knowledge graph embeddings [19] for e.g. link prediction, graph clustering and association rule mining techniques.

2 SPIRIT: leveraging the SANSA stack for transparency and compliance

In the following, we present technical details of our transparency and compliance checking approach. Figure 3.2 sketches the transparency and compliance architecture and shows the components of the proposed architecture. We keep the notion of "Line of Business" applications as presented in deliverable *D2.3 Transparency Framework V1*.

The architecture is divided into three main parts: The company systems (bottom left), the SPIRIT dashboard component and business logic (top left), and the big data application built on top of SANSA (right). While the company systems continuously produce log information concerning transactions involving user data, the SANSA application can be used to analyse it at scale. The SPIRIT dashboard controls the SANSA application and presents results to the user. The components are described in the following.

2.1 Transaction logs

When it comes to the ledger storing all data sharing and processing events (presented in deliverable *D2.3 Transparency Framework V1*), a straightforward option would be to recommission/extend existing application logs to include the information necessary to verify compliance. Considering that all data processing and sharing events need to be recorded it may not be feasible to store all log information on disk. Consequently, with large amounts of event data comes the need for a file system that: (1) is able to store big data; (2) is fault tolerant; and (3) is capable of supporting parallel processing. The *Hadoop Distributed File System (HDFS)*[12] fulfills said criteria and is the default choice for Apache Spark and Apache Flink. Moreover, there is a stable and mature solution to transfer log data to HDFS, called Apache Flume[13]. It provides an architecture of *sources*, *channels* and *sinks*, where *interceptors* can transform the log content, e.g. obtained from an application log source before it is passed along to the channel. This allows heterogeneous transaction logs to be translated to RDF on the fly. Since HDFS can serve as an

[12] HDFS, `http://hadoop.apache.org/`
[13] Apache Flume, `http://flume.apache.org/`

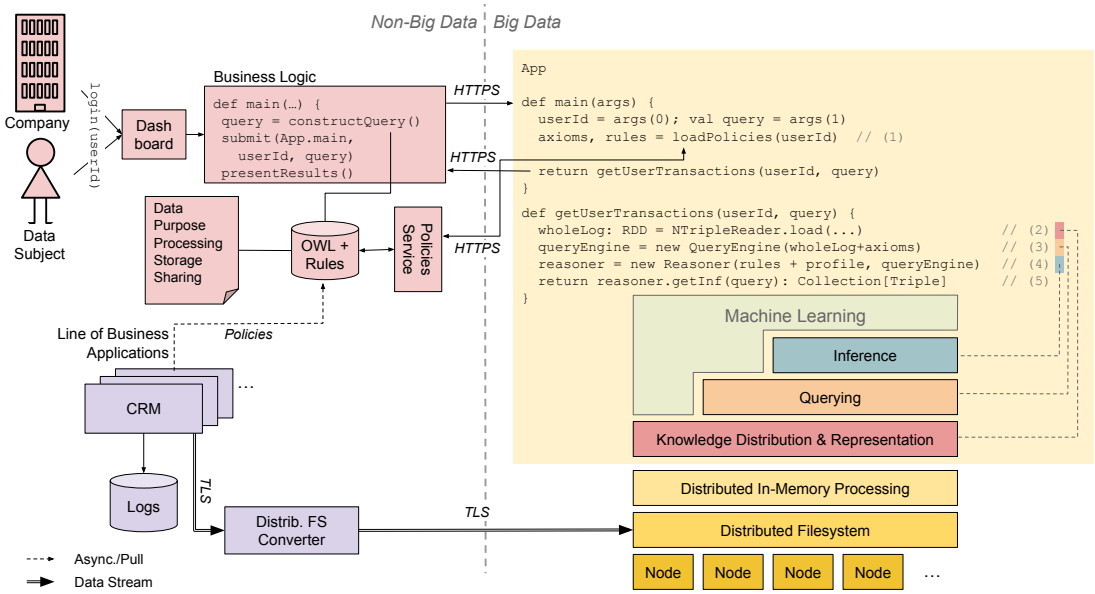

Figure 3.2: SPIRIT architecture exemplifying the transparency use case

Apache Flume sink integrated RDF data can then be persisted on a distributed storage cluster for later access.

2.2 The SPIRIT dashboard

The SPIRIT dashboard provides a means for data subjects, companies and supervisory authorities to obtain transparency with respect to the processing of personal data and compliance with respect to usage policies. For confidentiality reasons, personal data should only be accessible after the identity of the data requester is confirmed. Such functionality can be provided via existing industry standards such as OAuth2[14], OpenId Connect[15] or SAML2.0[16]. One of the key challenges for the transparency and compliance dashboard is the presentation of data in a manner that considers the users' cognitive limitations. Here a combination of paging, faceted browsing, search and filtering techniques are necessary. Moreover information can be grouped by data categories obtained from the policy storage. In all cases the request is converted into a query which is passed to the SANSA application, together with a user identifier. The results of the respective processing task is then passed back to the dashboard to be presented to the user.

2.3 Transaction log processing with SANSA

Our SANSA-based architecture inherently provides *traceability* by: i) storing and serving all log data records in a scalable, big data-ready processing framework; and ii) offering querying functionality on big semantic data and thus enabling all user-centric transactions as depicted in

[14]OAuth2, https://oauth.net/2/
[15]OpenId Connect, http://openid.net/connect/
[16]SAML2.0, https://wiki.oasis-open.org/security/FrontPage

Figure 3.2. In addition, using semantic data eases the *data integration* across several heterogeneous line of business applications, brings *interoperability* across platforms and a simple way to link user data and policies.

As sketched in Figure 3.2 the main steps that need to be performed include: (1) loading the policies (modelled according to deliverable *D2.1: Policy Language V1*) from the policy store and dividing them into rules that are used in the reasoning step, and schema/ontology axioms added to the log data later; (2) loading the RDF log data (modelled according to deliverable *D2.3 Transparency Framework V1*) stored in the distributed file system; (3) initialising a query engine with the log and schema/ontology data; (4) creating a reasoner which works on the query engine, considers the rules from the policy store and also a set reasoning profiles; and eventually (5) invoking the backward chaining on the given query goal.

Besides providing transparency to data subjects, the infrastructure should help data controllers and processors to demonstrate to data subjects and supervisory authorities that their business processes comply both with the consent provided by the data subject and relevant obligations from the GDPR. Our SPIRIT architecture can be used to implement the compliance checking presented in Chapter 1 on the basis of: (i) encoding user data policies in (subsets of) OWL 2 DL according to the SPECIAL policy language defined in *D2.1: Policy Language V1*; and (ii) providing the compliance checking mechanisms on the basis of the inference rule-engine provided by SANSA. As for the former, we allow policies to define restrictions in terms of the five data categories identified in SPECIAL (as depicted in Figure 3.2):

- Data reflects which personal data is governed by the policy.
- Processing lists the operations (e.g. anonymisation, aggregation, etc.) performed on the personal data.
- Purpose describes why data are collected/processed.
- Storage concerns where data are stored and for how long. Here, we follow a pragmatic approach specifying the location and the upper bound time period for the storage, strictly bound to the service needs.
- Sharing specifies the potential use of the personal data by third parties.

In addition to the personal data policies, the SPIRIT architecture holds rules that provide mechanism to check compliance of data processing and sharing transactions according to the usage policies and regulations obligations. Acknowledging that GDPR compliance checking cannot be fully automated (given the generality, vagueness and subjectivity inherent in the regulation), we focus on verifying minimal sets of conditions (*"if condition X holds then the data policy Y is violated"*) to assist the stakeholders in charge of providing evidence of GDPR compliance. As shown in Figure 3.2, it would be possible to support multiple reasoning profiles to enable different semantics and levels of expressivity. Also, this inference scheme is integrated in the resolution of the transparency request of the data subject, in order to offer not only a trace of their user-centric events, but also compliance evidence according to their usage policies and regulatory obligations.

3 Relevance for SPECIAL

Although SPIRIT goes a long way towards catering for transparent and compliant personal data processing, there are still a number of open research challenges to be considered in the Knowledge Distribution & Representation, Querying, Reasoning and Machine Learning layers.

3.1 Knowledge distribution, representation & querying

In *D1.3: Policy, transparency and compliance guidelines V1*, we identify requirements that need to be satisfied by a transparent personal data processing architecture. Herein we focus specifically on storage, *security*, *availability*, *performance*, and *scalability* robustness requirements. The scalability and availability features are inherently supported by our SPIRIT architecture. In order to reduce the volume of the data stored on disk, while at the same time protecting the data from unauthorised access, a combination of encryption and compression mechanisms are necessary [22]. Additionally different partitioning, indexing and materialisation strategies based on access policies and requests are required, in order to improve performance.

3.2 Reasoning

The SPIRIT architecture advocates to implement the conformance checking mechanism presented in Chapter 1, nonetheless, its efficient practical application using SANSA is not trivial. We are currently exploring optimisations to perform such task efficiently in such big data scenario.

3.3 Machine learning

Although thus far we focus on using SANSA for transparency and compliance checking, a natural next step would be to demonstrate how the SANSA machine learning layer can be used for the SPECIAL use cases, in particular to serve personalised recommendations. However, it is worth noting that there presently exists legal uncertainty with respect to ML algorithms trained over personal data where the consent is later withdrawn, and whether or not those models need to be thrown away.

4 Summary of results and future outlook

In this chapter we proposed SPIRIT, a high level big data architecture that could be used to store and process the SPECIAL ledger for transparent and compliant personal data processing. While we currently work on a prototype and an evaluation of SPIRIT, when it comes to privacy preserving data driven services there are still a number of open research challenges with respect to the extension of the architecture to cater for compressed and encrypted data and privacy preserving machine learning. We will explore these concerns in future versions of this deliverable.

Acknowledgements

We thank Jens Lehmann (Smart Data Analytics Group, University of Bonn, DE. `jens.lehmann@cs.uni-bonn.de`) and Patrick Westphal (Institute for Applied Informatics (InfAI), University of Leipzig, DE. `patrick.westphal@informatik.uni-leipzig.de`) for their contribution to *Chapter* 3 *Leveraging the Big Data Europe Infrastructure*.

Chapter 4

Fair Exchange

Creating a reliable record for joint operations, and creating records with multiple simultaneous signatures, requires the adoption of fair exchange protocols to guarantee that the operation is completed (e.g. data are transferred) if and only if all the involved parties sign the record and the record is included in the ledger. In this chapter, we how fair exchange protocols could be used to provide certain guarantees in terms of *completeness*, *correctness*, and *non-repudiation* of data sharing events.

1 Motivation

Let us recall some of the desiderata for the transparency ledger (cf. D1.3 and [10]):

Completeness: All data processing and sharing events should be recorded in the ledger.

Correctness: The records stored in the ledger should accurately reflect the processing event.

Immutability: The log should be immutable such that it is not possible to go back and reinvent history.

Integrity: The log should be protected from accidental and/or malicious modification.

Non-repudiation: When it comes to both data processing and sharing events it should not be possible to later deny that the event took place.

Some of these desiderata can be addressed with a combination of standard cryptographic techniques and trusted architectures. For instance, if a record R in the transparency ledger is signed by a party X, then X cannot deny to be the author of R, and no party $Y \neq X$ can replace R with a modified version R' – therefore digital signatures address *non-repudiation* and, to some extent, *immutability* and *integrity*. Note that X itself could tamper with R and modify it. Moreover, it may be possible to delete R. This is why suitable trusted architectures are needed in order to achieve the full *integrity* and *immutability* of the transparency ledger.

Other desiderata in the above list cannot be fully achieved. For instance, *correctness* and *completeness* cannot be guaranteed for those data processing events that involve only the data controller, as there is no way to prevent companies from logging incorrect information or not entering the information into the log.

The situation is different for those operations that involve two or more parties. The two most important examples of such events are:

- consent release;
- data transfer.

In a typical consent release, it is unlikely that the two parties (the data subject and the data controller) collude, because they have opposite interests. Similarly for data transfers where the following condition holds:

> *the recipient (resp. the sender) – that wants to comply with the GDPR and prove that the data transfer was legitimate on its side – wants to be able to prove how and when the data has been acquired (resp. released), under which policy/licence it has been released, and what is the source (resp. recipient).*

In both cases, the involved parties require the following guarantees:

1. the recipient wants the *object of interest* (the data subject's consent, or the shared data, possibly with the associated sticky policy), together with a proof of its provenance, for *non-repudiation*;
2. the sender wants a *receipt*, again for *non-repudiation*; in the simplest case, the receipt might just be a proof that the recipient has received the object of interest, but it might also be a proof that the current transaction has been faithfully recorded in the ledger; in this case, the receipt would also guarantee the *completeness* and *correctness* of the ledger with respect to the operations that involve two non-colluding parties.

This schema is an instance of a *fair exchange protocol*. The characteristic feature of such protocols is that either both parties get what they want (i.e. the object of interest and the receipt, respectively), or none of them obtains anything. In the rest of this chapter, we illustrate the general features of fair exchange protocols and discuss which methods might be applicable to the transparency ledger.

2 Basic notions and desired properties

The abstract formulation of the fair exchange problem is the following: *two parties A and B (or Alice and Bob) want to exchange two objects O_A and O_B such that A receives O_B if and only if B receives O_A.*

It is known that this objective cannot possibly be achieved without a *trusted third party* (TTP). In 1980, Even and Yacobi in [12] proved that no deterministic protocol exists, in which there is no participation of a third party. Further research is discussed in [13], that provides an extended negative result in a richer model, with multiple TTP, that highlights the role of connectivity in the network of peers. There exist some protocols without TTP, but they provide weaker (e.g. probabilistic) guarantees, and sometimes rely on assumptions on the computational power of A and B (which restrict applicability). Therefore, we shall focus on protocols *with* TTP.[1]

The following is a finer-grained list of properties that may be required of a fair-exchange protocol, depending on context:

[1] A fully detailed survey of fair exchange protocols, including the approximate and probabilistic notions of fairness and all the classes of protocols not discussed here, will be included in D1.7.

- *Fairness:* The protocol should be *fair* in the sense that neither Alice nor Bob should be able to obtain an advantage on the other player. In other words, either Bob receives the message O_A and Alice the corresponding O_B or none of them receives useful information.
- *Non-repudiation of origin:* Alice should not be able to deny the fact that she sent the message O_A. This means that Bob, at the end of the protocol, should have enough information to prove the sender's identity.
- *Non-repudiation of receipt:* Bob should not be able to deny the fact that he received the message. Alice should get a receipt for the message that can be used as a proof in a court of law.
- *Authenticity:* The players should be guaranteed of their reciprocal identity.
- *Integrity:* The parties should not be able to corrupt the message and/or its receipt, e.g., Alice should not be able to obtain a receipt for a message different from the one received by Bob and *vice versa*.
- *Confidentiality:* The protocol should be such that only Alice and Bob will be able to read the content of the message. Notice that this property applies also to the TTP, that should not be able to infer useful information about the message.
- *Timeliness:* The protocol terminates within a finite and known *a priori* time.
- *Temporal Authentication:* Some applications, e.g., patent submission, require the possibility to verify the time at which the message was sent. The timestamp should be observable by the players and should be ensured by a trusted authority.
- *Sending Receipt:* Some fair exchange protocols might involve human interaction, e.g., certified email ones. It might be desirable that the sender obtains an evidence of the fact that he has *started* the process of exchanging messages. Notice that this receipt may not contain any information generated by the recipient, e.g., it is produced by a third authority.

3 Classification based on TTP and optimistic protocols

Fair Exchange protocols with TTP can be essentially classified as on-line/in-line and off-line (or optimistic) protocols. In the first class, a Trusted Third Party (TTP for short) has a central role in the protocol in the sense that *each* exchange involves the TTP. In the optimistic protocols, the TTP comes into play only in case of disputes or errors (e.g. communication failures) while, in the other cases, the users run the protocols by themselves.

It is clear that the latter class of protocols has a number of advantages with respect to the former. In-line protocols are usually simpler than optimistic ones but have the drawback that the TTP could become a bottleneck for the system. On the contrary, in the absence of disputes, an off-line TTP does not even know that a pair of players exchanged messages. For this reason, off-line protocols are more appropriate for SPECIAL's purposes, and we will focus on them in this report.

Here is a more detailed definition of the different classes of protocols:

- *Inline TTP*. In *inline* protocols the TTP is involved in every *message transmission*. In other words, every message is either sent to the TTP or is sent by the TTP to another party.
- *Online TTP*. In an *online TTP* protocol, the TTP is involved in every *protocol execution*, i.e., in every run of the protocol there exists at least one message that is either sent to or received by the TTP, but there might exist messages exchanged directly by the other parties.
- *Offline TTP*. In a protocol, the TTP is said to be offline if the TTP participates in the protocol only in case one of the parties misbehaves or in case of technical failures.

Offline protocols are also called *optimistic*, since the underlying assumption is that in the vast majority of cases no intervention is required from the TTP. The ability of the TTP to fix problems prevents deliberate misbehavior, since malicious participants have nothing to gain from protocol violations.

Since TTP involvement is not required at each execution of an off-line protocol, optimistic protocols are typically described by defining the algorithm for the honest parties along with the recovery procedures that the parties and the TTP have to execute in case of failures/timeouts.

Among the first optimistic protocols for contract signing we mention [9]. This protocol is less interesting for SPECIAL because of its probabilistic nature (eventually, the contract C is binding for both parties with a probability p). Considering that in SPECIAL the "contracts" are consent to data usage and sticky policies – that *must* be preserved to comply with the GDPR – probabilistic guarantees are not satisfactory.

The idea of *optimistic* protocols is introduced by Micali [16, 17] in the context of certified email, and in the setting of efficient fair exchange protocols for *generic items* in [2, 3, 8].

Micali's protocol can probably be adapted to our setting and will be discussed later.

In the protocols for generic items, the degree of fairness guaranteed by the protocol depends on certain properties of the items to be exchanged: if the third party can undo a transfer of an item (so called revocability) or if it is able to produce a replacement for it (so called generatability) the protocol achieves true fairness. None of these assumptions applies to data transfer or consent transfer.

In [15] the authors consider a sort of *possibly-asymmetric* fair exchange problem. Motivated by the exchange of files in p2p networks, the authors consider the following variation of fairness. Informally, an exchange is fair either when both parties receive the requested file or when one party receives a payment for a file she provides. Again, there seems to be no match with the application to the transparency ledger.

Distributed Trusted Third Party As for inline protocols the idea of reducing the trust over a *single* third party has been developed also in the case of off-line protocols. In [5] the authors first introduce the possibility of distributing the role of TTP among a set of *honest neighbours* in the network. Intuitively the protocol initiator uses a (publicly) verifiable secret sharing scheme to generate n shares of her message m. She then encrypts each share with the public keys of n other parties in the network and sends all such encrypted shares the the receiving party. The receiving party sends back the expected message and receives the original message m. If sender and receiver act properly, the protocol terminates without the need of any external intervention. If any of the player tries to deviate from the protocol, if a sufficient number of honest players

is present, the protocol fairness is guaranteed. This solution implicitly assumes the presence of timeouts and some bounds on the number of untrusted parties. In [14], the authors show that timeouts, i.e., the existence of loosely synchronised clocks, are essential for guaranteeing fairness, unless complex (and costly) cryptographic tools like secure multiparty computation are deployed.

In [6] the authors present a solution in which perfect fairness can be achieved if a majority of parties are honest but, whenever the majority of parties are dishonest, it is possible to achieve probabilistic fairness with arbitrary low probability of error.

4 Micali's optimistic fair-exchange protocol

Let us briefly recall the notation for describing protocols.

- $X\|Y$ denotes the concatenation of strings X and Y;
- $[X]_A$ is party A's signature of string X;
- $E_A(X)$ (resp. $D_A(X)$) is the encryption (resp. decryption) of X with A's public key;
- in order to make encryption more robust, a random string R may be used; it is made explicit with notation $E_A^R(X)$; it is assumed that decryption recovers both X and R;
- $A \to B : X$ means that A sends message X to B.

Now we are ready to recall Micali's fair protocol for certified email [17]. In the following A, B and TTP denote Alice, Bob and the trusted third party, respectively.

In the absence of errors or misbehavior sending an email M requires 3 messages:

1. $A \to B : Z$, where $Z = E_{TTP}^R(A\|B\|M)$
2. $B \to A : [Z]_B$
3. $A \to B : (M\|R)$.

At the end of this sequence, if no one cheats, B has M and A has the receipt $[Z]_B$. With this receipt, A can prove that she sent M to B simply by exhibiting M, R, and the receipt. Indeed, everyone can (i) compute Z from M and R (since the rest is public information), and (ii) see that B received Z, by verifying B's signature on the receipt.

The protocol may depart from the above sequence in several places:

- If B fails to send the receipt (message #2), then A does not send the plaintext M (message #3). Consequently, B cannot read M (as it can be extracted from Z only with the private key of TTP).
- If B sends the receipt but A does not send back the plaintext M, then B resorts to the TTP:

  4. $B \to TTP : Z\|[Z]_B$
  5. $TTP \to B : (M\|R)$
  6. $TTP \to A : [Z]_B$

So B gets the plaintext message from TTP (that extracts it from Z using its private key).

- Similarly, if there is a mismatch between steps 2 and 3, B goes to step 4. A mismatch occurs if Z is different from the expression $E^{R'}_{TTP}(A\|B\|M')$ computed with the values M', R' sent in step 3. The TTP decrypts the actual value Z sent initially by A, so that B has access exactly to the message that correponds to the receipt returned in step 2.

- The lack of messages from A or B may be due to network problems. If message #2 is lost, then the transaction fails and must be restarted. If message #3 is lost, then messages 4–6 solve the problem. The connections with the TTP must be *resilient*, that is, each message is eventually delivered.

- If B tried to contact the TTP instead of sending back the receipt (which means skipping directly from step 1 to step 4), then B would eventually be able to read M, but at the same time the TTP would send the receipt back to A (step 6).

- Finally, A might put nonsense in M. However, the receipt would correspond to that nonsense; A could make a fake receipt for a different message only by forging B's signature.

In summary, fairness is guaranteed: B obtains M iff A obtains the corresponding receipt. Note that in the absence of cheating and transmission errors the TTP is not involved, and the transfer requires only 3 messages.

The confidentiality of M can be guaranteed with a simple variant: it suffices to replace M with $E_B(M)$ across all steps. In this way, only B can read M; the TTP itself would only be able to extract the cyphertext $E_B(M)$ from Z.

Moreover, it is possible to guarantee the provenance of TTP's messages by having the TTP sign them.

The limitation of this version of the protocol is that all steps contain either M or an encrypted version of it. In SPECIAL's data transfer scenarios M could be a very large dataset, whose size would make the protocol practically unfeasible.

5 Optimistic fair-exchange protocols for large data transfers

To the best of our knowledge, the only optimistic fair exchange protocol for large data is [20] (a variant of Micali's protocol). Unfortunately, it is specialised to an e-commerce scenarios, whose features do not match SPECIAL's requirements. Here are the main features that hinder the application of this protocol to the transparency ledger:

- The protocol requires a preliminary setup phase in which the TTP produces n certificates that contain a digest of the dataset. The cost of the setup is balanced by the reuse (reselling) of the same data multiple times. One of the main reference applications of [20] is selling multimedia contents, where each item is fixed and sold multiple times. However, if the transferred datasets change frequently along time, then the protocol tends to become online.

- Fairness does not concern a receipt of the data, but a payment token (which removes some of the problems related to size, since the payment token is usually very small).

- The data are transmitted in a single message. However, in SPECIAL's scenarios, the data may be too large for this kind of transfer. The transmission of data with multiple messages potentially opens the way to novel attacks, and requires a specific validation of the modified protocol. Moreover, datasets are usually not transmitted in messages, but made available at some URI, where they can be downloaded.

For these reasons we introduce in the following another variant of Micali's protocol, where data is referred to by means of URIs and the receipt is tightly related to the transferred data.

A new fair exchange protocol for large data transfers. Let URI be the location of the dataset to be transferred from A to B, and let $*URI$ denote the actual data resulting from dereferencing URI. We add to notation a hash function $h(\cdot)$ resistant to pre-image attacks (i.e. it is computationally hard, given a value $x = h(y)$ to find $z \neq y$ such that $h(z) = x$). The protocol is the following:

1. A makes the dataset D available at URI in encrypted form, using a session key k (therefore $*URI = E_k(D)$)
2. $A \rightarrow B : Z_1$, where $Z_1 = [A \| B \| URI \| h(*URI) \| t_0]_A$ (t_0 a unique timestamp)
3. B downloads $D' = *URI$ and verifies that $h(D')$ equals the field $h(*URI)$ in Z_1; in case of mismatch abort the protocol; otherwise go to the next step;
4. $B \rightarrow A : [Z_1]_B$
5. $A \rightarrow B : Z_2$, where $Z_2 = [A \| B \| URI \| E_{TTP}(k) \| t_0]_A$
6. $B \rightarrow A : [Z_2]_B$
7. $A \rightarrow B : [E_B(k)]_A$; in case of timeout, B starts the recovery procedure (step 9)
8. B decrypts the downloaded dataset D' using k and verifies the result. In case of problems, B starts the recovery procedure (step 9).

The recovery procedure (involving the TTP) is the following:

9. $B \rightarrow TTP : [Z_2]_B$
10. $TTP \rightarrow B : [E_B(k)]_{TTP}$
11. $TTP \rightarrow A : [[Z_2]_B]_{TTP}$

The above protocol enjoys the following properties:

1. If no problems occur, then at step 8 B has the dataset D and A has the two receipts $R_1 = [Z_1]_B$ and $R_2 = [Z_2]_B$. With R_1, B declares that the downloaded encrypted dataset corresponds to the hash value included in Z_1. With R_2, B declares that A delivered the session key k for decrypting D, protected with the TTP's public key. Together, the two receipts show that A sent all the messages prescribed by the protocol, and that the digest in Z_1 (i.e. $h(*URI)$) is correct. The two receipts (as well as Z_1 and Z_2) are connected by their common elements A, B, URI, and t_0, that constitute a unique identifier of the transaction.

2. If a problem occurs before step 7, then the protocol is aborted (we assume a timeout for each step). Then A has at most one of the two receipts needed, and B has no way of decrypting $*URI$.

3. If a problem occurs at steps 7 or 8, then the TTP extracts the session key k for B from Z_2, using TTP's private key, and sends the second receipt to A. After this stage, A has the two receipts, and B can decrypt $*URI$.

4. If B skips step 6 and goes to step 9 (to obtain k from TTP without releasing R_2), then the receipt R_2 will be delivered to A anyway, by the TTP (step 11).

5. The actual data D are kept confidential; k is always encrypted with B's public key so not even the TTP can decrypt it and see the dataset.

6. The provenance of all messages is guaranteed by the senders' digital signatures. This addresses man-in-the-middle attacks.

7. Replay attacks are addressed as follows: (i) all messages but those in 7 and 10 contain the unique timestamp t_0 (if the clock granularity is insufficient, then it can be complemented with a nonce); (ii) messages 7 and 10 depend on the randomly chosen key k that is changed at every transaction.

8. The dataset D might consist of rubbish, but the value $h(*URI)$ in Z_1 corresponds to the rubbish and is signed by A, like the decryption key k in Z_2, Then, using Z_1 and Z_2, B can convincingly argue that A transferred "bad" data. Indeed, by the properties of $h(\cdot)$ it would be practically impossible for B to find a $D' \neq D$ such that $h(D') = h(D)$. Moreover, the TTP can certify (using its private key and Z_2) that k is actually the key signed by A in step 7.

9. The dataset D itself needs not be processed by the TTP. Only A and B shall compute its hashed value, once for each transaction.

Bibliography

[1] A. Abraham. Self-sovereign identity. 2017.

[2] N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for fair exchange. In *CCS '97, Proceedings of the 4th ACM Conference on Computer and Communications Security, Zurich, Switzerland, April 1-4, 1997.*, pages 7–17, 1997. doi: 10.1145/266420.266426. URL `http://doi.acm.org/10.1145/266420.266426`.

[3] N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. In *Security and Privacy - 1998 IEEE Symposium on Security and Privacy, Oakland, CA, USA, May 3-6, 1998, Proceedings*, pages 86–99, 1998. doi: 10.1109/SECPRI.1998.674826. URL `http://dx.doi.org/10.1109/SECPRI.1998.674826`.

[4] S. Auer, S. Scerri, A. Versteden, E. Pauwels, A. Charalambidis, S. Konstantopoulos, J. Lehmann, H. Jabeen, I. Ermilov, G. Sejdiu, A. Ikonomopoulos, S. Andronopoulos, M. Vlachogiannis, C. Pappas, A. Davettas, I. A. Klampanos, E. Grigoropoulos, V. Karkaletsis, V. de Boer, R. Siebes, M. N. Mami, S. Albani, M. Lazzarini, P. Nunes, E. Angiuli, N. Pittaras, G. Giannakopoulos, G. Argyriou, G. Stamoulis, G. Papadakis, M. Koubarakis, P. Karampiperis, A.-C. N. Ngomo, and M.-E. Vidal. The bigdataeurope platform - supporting the variety dimension of big data. In *Proceedings of the 17th International Conference on Web Engineering (ICWE)*, Lecture Notes in Computer Science. Springer, 2017.

[5] G. Avoine and S. Vaudenay. *Optimistic Fair Exchange Based on Publicly Verifiable Secret Sharing*, pages 74–85. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. ISBN 978-3-540-27800-9. doi: 10.1007/978-3-540-27800-9_7. URL `http://dx.doi.org/10.1007/978-3-540-27800-9_7`.

[6] G. Avoine, F. Gärtner, R. Guerraoui, and M. Vukolić. *Gracefully Degrading Fair Exchange with Security Modules*, pages 55–71. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-32019-7. doi: 10.1007/11408901_5. URL `http://dx.doi.org/10.1007/11408901_5`.

[7] A. Baliga. The blockchain landscape. *Persistent Systems*, 2016.

[8] F. Bao, R. H. Deng, and W. Mao. Efficient and practical fair exchange protocols with off-line TTP. In *Security and Privacy - 1998 IEEE Symposium on Security and Privacy, Oakland, CA, USA, May 3-6, 1998, Proceedings*, pages 77–85, 1998. doi: 10.1109/SECPRI.1998.674825. URL `http://dx.doi.org/10.1109/SECPRI.1998.674825`.

[9] M. Ben-Or, O. Goldreich, S. Micali, and R. L. Rivest. A fair protocol for signing contracts. *IEEE Trans. Information Theory*, 36(1):40–46, 1990. doi: 10.1109/18.50372. URL http://dx.doi.org/10.1109/18.50372.

[10] P. Bonatti, S. Kirrane, A. Polleres, and R. Wenning. Transparent personal data processing: The road ahead. In *International Conference on Computer Safety, Reliability, and Security*, pages 337–349. Springer, 2017.

[11] C. Cachin and M. Vukolić. Blockchains consensus protocols in the wild. *arXiv preprint arXiv:1707.01873*, 2017.

[12] S. Even and Y. Yacobi. Relations among public key signature systems. Technical Report 175, Comput. Sci. Dept., Technion, Haifa, Israel, 1980.

[13] B. Garbinato and I. Rickebusch. Impossibility results on fair exchange. In G. Eichler, P. G. Kropf, U. Lechner, P. Meesad, and H. Unger, editors, *10th International Conference on Innovative Internet Community Services (I^2CS), Jubilee Edition 2010, June 3-5, 2010, Bangkok, Thailand*, volume 165 of *LNI*, pages 507–518. GI, 2010. ISBN 978-3-88579-259-8. URL http://subs.emis.de/LNI/Proceedings/Proceedings165/article5616.html.

[14] A. Küpçü and A. Lysyanskaya. *Optimistic Fair Exchange with Multiple Arbiters*, pages 488–507. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-15497-3. doi: 10.1007/978-3-642-15497-3_30. URL http://dx.doi.org/10.1007/978-3-642-15497-3_30.

[15] A. KÃijpÃğÃij and A. Lysyanskaya. Usable optimistic fair exchange. *Computer Networks*, 56(1):50 – 63, 2012. ISSN 1389-1286. doi: http://dx.doi.org/10.1016/j.comnet.2011.08.005. URL //www.sciencedirect.com/science/article/pii/S138912861100301X.

[16] S. Micali. Certified e-mail with invisible post offices. Invited presentation at RSA '97 conference, 1997.

[17] S. Micali. Simple and fast optimistic protocols for fair electronic exchange. In E. Borowsky and S. Rajsbaum, editors, *Proceedings of the Twenty-Second ACM Symposium on Principles of Distributed Computing, PODC 2003, Boston, Massachusetts, USA, July 13-16, 2003*, pages 12–19. ACM, 2003. ISBN 1-58113-708-7. doi: 10.1145/872035.872038. URL http://doi.acm.org/10.1145/872035.872038.

[18] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[19] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2016.

[20] C. C. Oniz, E. Savas, and A. Levi. An optimistic fair e-commerce protocol for large e-goods. In *Proceedings of the International Symposium on Computer Networks, ISCN 2006, June 16-18, 2006, Istanbul, Turkey*, pages 214–219. IEEE, 2006. doi: 10.1109/ISCN.2006.1662536. URL https://doi.org/10.1109/ISCN.2006.1662536.

[21] H. J. ter Horst. Completeness, decidability and complexity of entailment for RDF schema and a semantic extension involving the OWL vocabulary. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2):79–115, Oct. 2005.

[22] W. Zheng, F. Li, R. A. Popa, I. Stoica, and R. Agarwal. Minicrypt: Reconciling encryption and compression for big data stores. In *Proceedings of the Twelfth European Conference on Computer Systems*, pages 191–204. ACM, 2017.