SPECIAL

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

Deliverable D2.8

Transparency and Compliance Algorithms V2

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:	D2.8
Deliverable title	Transparency and Compliance Algorithms V2
Contractual Date of Delivery:	30-11-2018
Actual Date of Delivery:	30-11-2018
Editor (s):	Sabrina Kirrane (WU)
Author (s):	Sabrina Kirrane (WU), Piero Bonatti (CeRICT), Javier D. Fernández (WU), Clemente Galdi (CeRICT), Luigi Sauro (CeRICT), Daniele Dell'Erba (CeRICT), Iliana Petrova (CeRICT), Ida Siahaan (CeRICT)
Reviewer (s):	Uroš Milošević (TF), Axel Polleres (WU), Philip Raschke (TUB)
Participant(s):	WU, TF, ERCIM, CeRICT, TUB
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):	52

Disclaimer

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

Contents

List of Figures

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, 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 *D1.7: Policy, transparency and compliance guidelines V2*, the SPECIAL policy language which is described in *D2.1: Policy Language V1*, and the SPECIAL transparency and compliance framework presented in *Deliverable D2.7: Transparency Framework V2*.

Chapter 1

Compliance Checking

In this chapter we illustrate a complete and tractable algorithm for compliance checking of SPECIAL's policies. The algorithm can be used indifferently for ex-ante as well as ex-post compliance checking.[1] The policy language $\mathcal{PL}$ introduced in D2.4 has been extended in order to import external OWL2-EL vocabularies defining domain specific information such as data/purpose/recipient categories. This is needed to support some of the features introduced in the second version of SPECIAL's vocabularies (cf. D2.5). We use a novel and specifically tailored procedure which processes $\mathcal{PL}$ policies and OWL2-EL external vocabularies separately. 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 2-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

[1]The only difference is that in ex-ante checking the policy P_0 fed to the algorithm represents a processing action that the data controller is about to execute, while in ex-post checking P_0 represents an action that has been executed in the past.

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 OWL2.

The policy language $\mathcal{PL}$ introduced in D2.4 provides limited means to specify policy-related categories of data, purposes, recipients and legal basis – essentially, $\mathcal{PL}$ allows only to structure a simple taxonomy of (possibly disjoint) concept names [15]. However, several application contexts may require more expressiveness in order to associate, for instance, parental consent to vulnerable users or social security purposes to union or member state authorizations. Consequently, we extend the SPECIAL's policy language by allowing to import external vocabularies defined in OWL2-EL. Our choice fell on this OWL2 profile for the following reasons:

- differently from other profiles, it provides qualified existential restrictions that are essential to specify properties like the ones described above;
- it is a tractable fragment of OWL2-DL that encompasses high-performance reasoners such as ELK [22].

However, we could not take any advantage from specifically optimized reasoners if we adopted a naive methodology that simply merges in a monolithic knowledge base $\mathcal{PL}$ structural specifications with external vocabularies. Conversely, we present a novel approach that enhances import-by-query (IBQ) techniques [20] in order to combine a fast structural subsumption algorithm for $\mathcal{PL}$ with the state-of-art OWL2-EL reasoner ELK.

In this chapter we illustrate the resulting OWL2-DL fragment and introduce the IBQ-based 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, et cetera. The following table shows the correspondence between the notation used in the other deliverables and the symbols used here:

ObjectUnionOf$(C_1 \dots C_n)$	$C_1 \sqcup \dots \sqcup C_n$
ObjectIntersectionOf$(C_1 \dots C_n)$	$C_1 \sqcap \dots \sqcap C_n$
ObjectSomeValueFrom$(Attr C)$	$\exists Attr.C$
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$(C_1\ C_2)$	disj(C_1, C_2)
FunctionalObjectProperty$(Attr)$	func$(Attr)$
ObjectPropertyRange$(Attr\ C)$	range$(Attr, C)$
owl : Nothing	$\bot$

2 The general structure of policies

We extend the logic $\mathcal{PL}$ presented in D2.4 so that the specification of a policy can make use of SPECIAL's vocabularies defined in the OWL2-EL profile (see www.w3.org/TR/owl2-profiles for a detailed description). First, we consider a dedicated signature Σ_S of data and object properties used to specify the structural attributes of a policy such as spl:hasData and spl:hasProcessing defined in the Appendix of D2.1. The resulting logic $\mathcal{PL} \mid$ OWL2-EL is captured by the following definition.

Definition 1 (Policy logic $\mathcal{PL}$ | OWL2-EL) *A $\mathcal{PL}|$*OWL2-EL *knowledge base $\mathcal{K} = \mathcal{K}_m \cup \mathcal{K}_e$ consists of a* main *knowledge base $\mathcal{K}_m$ and a* external *one $\mathcal{K}_e$. The main ontology $\mathcal{K}_m$ contains axioms of the following kinds:*

- $\mathsf{func}(R)$ *where R is a object or datatype property in Σ_S;*
- $\mathsf{range}(S, A)$ *where S is a object property in Σ_S and A a concept name.*

$\mathcal{K}_e$ is a OWL2-EL knowledge base not containing nominals (i.e. the class expressions ObjectOneOf *and* ObjectHasValue*) and object/datatype properties occurring in Σ_S.*

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

$$A_1 \sqcap \ldots \sqcap A_n \sqcap \exists R_1.C_1 \sqcap \ldots \sqcap \exists R_m.C_m \sqcap C_1 \sqcap \ldots \sqcap C_t \tag{1.1}$$

where each A_i is either a concept name or $\bot$, each $\exists R_j.C_j$ is an existential restriction such that R_j is a object property and C_j is a simple $\mathcal{PL}$ concept, and each C_k is a constraint $[l,u](f_k)$. All R_j and f_k are taken from Σ_S. 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.*

We adopt IBQ techniques using the external knowledge base $\mathcal{K}_e$ as an oracle. In a paper that is currently under revision, this approach has been proved to be correct under the conditions in Definition 1.1 that (i) $\mathcal{K}_e$ does not contain nominals and object/data properties occurring in $\mathcal{K}_m$ and (ii) $\mathcal{PL}$ concepts do not contain roles occurring in $\mathcal{K}_e$. IBQ reasoning provides two advantages: first, it allows to prove that the *worst case complexity* of the compliance checking presented in the next section is tractable (note that, in general, extending OWL2-EL with functional assertions $\mathsf{func}(R)$ leads to EXPTIME reasoning tasks). Secondly, adopting standard techniques that process $\mathcal{K}$ as a whole would require to use of full OWL2 reasoners, conversely IBQ reasoning allows to combine reasoners that are specifically optimized for $\mathcal{PL}$ and OWL2-EL.

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 knowledge base for the policy language and the external vocabulary, that will be denoted with $\mathcal{K}$. The SPECIAL's compliance checking is articulated as follows:

Phase 0 Classification of the external knowledge base $\mathcal{K}_e$;

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.

Phase 0

Phase 0 allows to use $\mathcal{K}_e$ as an oracle in Phase 3 and 4. More specifically, we exploit the capability of the state-of-art OWL2-EL reasoner ELK to "compile" $\mathcal{K}_e$ by producing in a single reasoning task (called classification) all the direct inclusions $A \sqsubseteq B$ between concept names that are entailed by $\mathcal{K}_e$. To ensure the correctness of the SPECIAL's compliance checking, we assume that $\mathcal{K}_e$ in extended by including, for each subformula $A_1 \sqcap A_2$ in P_0, an equivalence axiom $A_{1,2} \equiv A_1 \sqcap A_2$, where $A_{1,2}$ a fresh concept name not previously occurring in $\mathcal{K}_e$. This allows to use $A_{1,2}$ as "placeholder" for $A_1 \sqcap A_2$ in the classification process.

Thereafter, by $A \sqsubseteq_{\mathcal{K}_e} B$ we mean that $A \sqsubseteq B$ is logical consequence of $\mathcal{K}_e$, i.e. the resulting classification contains a sequence $A \sqsubseteq A_1, \ldots, A_m \sqsubseteq B$. Analogously, we write $\mathsf{disj}_{\mathcal{K}_e}(A_1, A_2)$ to mean that $\mathcal{K}_e$ entails $A_{1,2} \sqsubseteq \perp$ where $A_{1,2}$ is the placeholder of $A_1 \sqcap A_2$.

Note that Phase 0 needs to be done only when the data controller's policy P_0 are deployed or updated, and when $\mathcal{K}_e$ is modified (e.g. in order to add new terms to the vocabularies). Being OWL2-EL classification tractable, Phase 0 is in PTIME w.r.t. $|\mathcal{K}_e| + |P_0|^2$. In practice, classification is performed in a few seconds even for large knowledge bases (e.g. SNOMED CT) of $\sim 10^5$ axioms.

Phase 1

Phase 1 is the same as in D2.1, we report it here for the sake of completeness. Informally, it enforces 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) that 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.

The *worst case complexity* of Phase 1 is $O(|P_0| \times |P_1|)^2$ 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.

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

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 $\mathsf{func}(R) \in \mathcal{K}_m$
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 $\mathsf{func}(f) \in \mathcal{K}_m$
6)	$\exists R.D \sqcap D' \rightsquigarrow \exists R.(D \sqcap A) \sqcap D'$	if $\mathsf{range}(R,A) \in \mathcal{K}_m$ and A not a conjunct of D
7)	$A_1 \sqcap A_2 \sqcap D \rightsquigarrow \bot$	if $\mathsf{disj}_{\mathcal{K}_e}(A_1,A_2)$

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 first three rules are general (in)consistency checks, rules 4–6 compile in P_0 functional and range axioms of $\mathcal{K}_m$. Rule 7 searches inconsistent conjuncts in P_0 by using $\mathcal{K}_e$ as an oracle. 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.

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:

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. The STS algorithm applies a structural subsumption making use of the external knowledge base $\mathcal{K}_e$ as an oracle.

We can prove the following result:

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

```
    Input: $\mathcal{K}$ and a $\mathcal{PL}$ inclusion $P_0 \sqsubseteq P_1$ where $P_0$ is normalised
    Output: $\texttt{true}$ if $\mathcal{K} \models P_0 \sqsubseteq P_1$,   $\texttt{false}$ otherwise
            Note: By $C = C' \sqcap C''$ we mean that either $C = C'$ or $C'$ is a conjunct of $C$ (possibly
            not the 1st)
 1  begin
 2      foreach $P_{0,i}$ in $P_0$ do
 3          subsumed := $\texttt{false}$
 4          j := 1
 5          repeat
 6              if STS($\mathcal{K}, P_{0,i} \sqsubseteq P_{1,j}$) then subsumed :=$\texttt{true}$
 7              j := j+1
 8          until subsumed = $\texttt{true}$ OR j > $m$
 9          if subsumed = $\texttt{false}$ then return $\texttt{false}$
10      end
11      return $\texttt{true}$
12  end
```

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

```
    Input: $\mathcal{K}$ and an elementary $C \sqsubseteq D$ where $C$ is normalised
    Output: $\texttt{true}$ if $\mathcal{K} \models C \sqsubseteq D$,   $\texttt{false}$ otherwise
            Note: By $C = C' \sqcap C''$ we mean that either $C = C'$ or $C'$ is a conjunct of $C$ (possibly
            not the 1st)
 1  begin
 2      if $C = \bot$ then return $\texttt{true}$
 3      if $D = A$, $C = A' \sqcap C'$ and $A' \sqsubseteq_{\mathcal{K}_e} A$ then return $\texttt{true}$
 4      if $D = [l,u](f)$ and $C = [l',u'](f) \sqcap C'$ and $l \leq l'$ and $u' \leq u$ then return $\texttt{true}$
 5      if $D = \exists R.D'$, $C = (\exists R.C') \sqcap C''$ and STS($\mathcal{K}, C' \sqsubseteq D'$) then return $\texttt{true}$
 6      if $D = D' \sqcap D''$, STS($\mathcal{K}, C \sqsubseteq D'$), and STS($\mathcal{K}, C \sqsubseteq D''$) then return $\texttt{true}$
 7      else return $\texttt{false}$
 8  end
```

Proposition 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.*

4 Summary of results

We introduced an algorithm for compliance checking and policy consistency checking which exploits novel IBQ techniques to process $\mathcal{PL}$ and external OWL2-EL vocabularies separately. The first three phases pre-process the data controller's policy P_0 and the external vocabulary; they need to be executed only when P_0 or the external 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 0, which is a OWL2-EL classification, is in PTIME with respect to $|\mathcal{K}| + |P_0|^2$ and can be performed by well engineered OWL2-EL-tailored resoners such as ELK. 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

Before discussing existing distributed and decentralised platforms and frameworks and how they could potentially be leverage in SPECIAL, we first distinguish between centralised, decentralised and distributed systems. In a *centralised* system, clients connect to a single machine or a collection of independent machines that function as a single entity, that acts as a central controller for data storage and processing. In contrast in a *decentralised* system, there is no single centralised controller, but instead machines are able to act autonomously. While, a *distributed* system is composed of co-operating machines that serve a common function.

In the context of the SPECIAL project, information relating to data processing and sharing events could be stored in one or more distributed chains that are accessible via Application Programming Interfaces (APIs). 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. Also, existing blockchain platforms and frameworks could potentially be used to perform compliance checking in a transparent manner. Alternatively, decentralised platforms such as Solid, which relies heavily on World Wide Web Consortium (W3C) standards, could be used to provide individuals with more control over their personal data and how it is used. In this chapter, we provide background information on such distributed and decentralised platforms and frameworks and discuss how they can be used in the context of SPECIAL.

1 Distributed ledger technology

We start by providing some background information on the blockchain platforms in general, before digging 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[1] and HyperLedger which is hosted by The Linux Foundation[2].

1.1 Background information on blockchain

In this section, we provide background information on the bitcoin blockchain platform, commonly used consensus mechanisms, and introduce the broader blockchain landscape in terms

[1]`https://www.ethereum.org/`

[2]`https://www.hyperledger.org/`

of different types of blockchain platforms. Finally, we identify some challenges with respect to GDPR compliance and blockchain applications, as highlighted in a recent report produced by the European Union Blockchain Observatory and Forum.

1.1.1 The Bitcoin blockchain

Bitcoin [28], 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.

1.1.2 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.

Byzantine-fault tolerant (BFT) consensus strives to achieve consensus, considering that some peers in the distributed network of devices may be faulty, while others could potentially behave maliciously. Here message passing is used in order for honest nodes to form a consensus as to the state of the system through a majority.

Additional, details on the above consensus protocols can be found in [16].

1.1.3 Blockchain platforms

According to a recent blockchain landscape survey conducted by Baliga [8], 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[3].

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[4] 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[5] and Ripple[6].

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[7] and Hyperledger[8] 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[9] which targets Enterprises offers a robust, secure and scalable platform. An alternative platform is provided by Hyperledger.

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

1.1.4 Blockchain and the GDPR

A recent article [25] produced by the European Union Blockchain Observatory and Forum, examined some of the primary tensions between the GDPR and blockchain. From the offset the article makes it clear that the focus is on the compliance of use cases and applications that leverage blockchain technology, as opposed to the compliance of blockchain technology as a whole. In the following we summarize the key take home messages from the report:

- Considering the anonymity guarantees offered by blockchain platforms the identification of data controllers and processors could be difficult or perhaps even impossible.
- Although anonymous data is outside of the scope of the GDPR, the report highlights that there is intense debate with respect to the effectiveness of existing anonymisation techniques.
- The report recommends that personal data is not stored on the blockchain in clear text, irrespective of whether it is a permissionless or permissioned blockchain, however there are questions with respect to the effectiveness of existing encryption mechanisms and especially the futureproofing of existing encryption mechanisms. Also, it is not clear if hashed personal data should be considered as personal data or not.
- Given the immutability guarantees provided by such platforms, it remains to be seen how the rectification and erasure rights of the data subject can be supported. However, the article does point to a report produced by the Commission nationale de l'informatique et des libertés (CNIL)[10], which states that cryptographic delete could potentially be a solution.
- Smart contracts and chaincode could also prove to be troublesome in the case of automatic decision making, as the GDPR affords data subjects the right to be informed about such processing and to request human intervention and/or challenge the decision.
- Where the actual processing takes place could potentially also pose problems, as the GDPR states that data can only be transferred to third party countries if their data protection is at least equivalent to that of the GDPR, whereas public blockchains are not limited by territorial scope.

The SPECIAL project will continue to monitor discussions with respect to the aforementioned issues, especially those coming from Data Protection Authorities (DPAs), the European Data Protection Board (EDPB) and those arising from related court rulings.

1.2 Distributed ledger platforms and frameworks

In this section, we provide additional information on Ethereum and HyperLedger, which are at the time of writing two of the most popular distributed ledger platforms.

[10]CNIL Blockchain report, https://www.cnil.fr/sites/default/files/atoms/files/la_blockchain.pdf

1.2.1 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.

1.2.2 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.

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.

1.2.3 Distributed ledger platforms and SPECIAL

In the context of the SPECIAL project, information relating to data processing and sharing events could be stored in one or more distributed chains that are accessible via Application Programming Interfaces (APIs). 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. Depending on the use case, such information could be stored in a public permissionless blockchain such as Bitcoin or Ethereum or alternatively distributed ledger platforms such as Hyperledger could be used to develop a private permissioned blockchain dedicated to providing transparency with respect to personal data processing. However, as indicated by the European Union Blockchain Observatory and Forum, personal data should not stored on the blockchain in clear text and even when data is encrypted or hashed data controllers need to be mindful of encryption security vulnerabilities.

Also, existing blockchain platforms and frameworks could potentially be used to perform automatic compliance checking in a transparent manner. For instance, the financial services sector have setup an Interbank Information Network (IIN)[11], which uses JPMorgan's private permissioned blockchain 'Quorum' (built on top of the Ethereum blockchain platform), to support global payments and compliance checking.

In the context of the SPECIAL project Thomson Reuters are currently assessing the suitability of Ethereum smart contracts, to support automated usage policy compliance checking based on the SPECIAL compliance checking algorithm. Preliminary results indicate that the public Ethereum platform struggles with complex policies, resulting in smart contract timeout issues. Going forward the consortium will investigate potential optimisations, including assessing the suitability of alternative platforms built on top of both the Ethereum and the Hyperledger Fabric platforms.

Additionally, other platforms offered by *Hyperledger* could potentially also be of benefit in the context of the SPECIAL project. 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.

[11]https://www.euromoney.com/article/b1bfkrj10f1vts/banks-rush-to-join-jpmorgans-blockchain-based-interbank-information-network

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.

2 Decentralised application platforms

In 2011, the World Economic Forum produced a report entitled *Personal Data: The Emergence of a New Asset Class*[12], which described the results of a multi-stakeholder project to envisage what a personal data ecosystem of the future might look like. Among the key take homes was the need to take steps to *innovate around user-centricity and trust* and to *focus on interoperability and open standards*. In this section we introduce two alternative initiatives in this in the form of the digi.me[13] and SOLID[14] platforms and discuss the intersection between said platforms and the work carried out in the context of the SPECIAL project.

2.1 The Digi.me Platform

The overall mission of Digi.me is *a decentralised world where data is controlled by people for their own benefit*. They do so by providing tools that enable individuals to retrieve and integrate personal data from third parties, such as Facebook, Instagram, Twitter etc. The personal data is stored in an encrypted format on the storage platform of the users choice, whereby the user can control who has access to their personal data. Not only does Digi.me, allow you to search and analyse your personal data but also provides a software development kit (SDK) and Applications Programming Interfaces (APIs) that can be used by developers to create mobile applications that make innovative use of personal data, with the consent of the users.

2.1.1 Digi.me and SPECIAL

Digi.me provides data subjects with the ability to retrieve and integrate personal data from a variety of sources, while at the same time offering companies the ability to develop innovative mobile apps over this data. Considering these apps process personal data, SPECIAL could potentially be used to increase trust in the apps, via a SPECIAL transparency and compliance mobile app. The SPECIAL app would act as a monitor of Digi.me applications, ensuring that all data processing and sharing performed by the app complies with the permissions specified by the end user.

2.2 The Solid Platform

An alternative decentralised platform known as Solid[15] 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.

[12]http://www3.weforum.org/docs/WEF_ITTC_PersonalDataNewAsset_Report_2011.pdf
[13]https://digi.me/
[14]https://solid.mit.edu/
[15]https://solid.mit.edu/

Identity & Profiles Web Identity and Discovery (WebID)[16], 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.

Authentication & Access Control WebID-TLS[17] 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[18] 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.

2.2.1 SOLID and SPECIAL

The SOLID platform by design affords individuals more control over how their personal data is used. It does so by changing the paradigm such that data subjects maintain their personal data in a personal data store of their choice and applications are designed with work with distributed personal data stores. Limitations of the existing architecture relate to performance and control. More specifically in order for such an architecture to work in practice, there is a need for caching of data either by the client (the application), the server (each of the personal data stores) or both. The need for caching also brings with it the need to move beyond simple access control towards usage control where data subjects can stipulate how their data is used. The SPECIAL team is looking into investigating if the existing usage control policies, transparency and compliance checking can be embedded into the fabric of the SOLID architecture.

3 Summary of results and future outlook

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 distributed ledger technology and decentralised application platforms. 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*. 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.

[16]https://www.w3.org/2005/Incubator/webid/spec/identity/
[17]https://www.w3.org/2005/Incubator/webid/spec/tls/
[18]https://www.w3.org/TR/ldp/

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*.

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.

Chapter 3

Leveraging the Big Data Europe Infrastructure

In deliverables *D1.7 Policy, transparency and compliance guidelines V2*, *D1.8 Technical Requirements V2*, and *D2.7 Transparency Framework V2* 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 [5], which provides the foundational basis for large-scale integration and processing of data that is necessary to deliver such services. First we provide background information on the SPECIAL usage policies, event log and supporting vocabularies. Following on from this we describe two alternative architectures developed int he context of the SPECIAL project.

1 RDF Vocabularies for Usage Policies and Events

The vocabularies described in this section are based on the SPECIAL usage policy language[1] and log vocabulary[2], which were derived from in-depth legal analysis of use cases that require the processing and sharing of personal data for improved information and communication technology and financial services. SPECIAL usage policies can be used to denote the following information at different levels of granularity:

- *Data* describes the personal data collected from the data subject.
- *Processing* describes the operations that are performed on the personal data.
- *Purpose* represents the objective of such processing.
- *Storage* specifies where data are stored and for how long.
- *Recipients* specifies with whom the data is shared.

[1]https://www.specialprivacy.eu/images/documents/SPECIAL_D2.1_M12_V1.0.pdf
[2]https://aic.ai.wu.ac.at/qadlod/policyLog/

In this paper we use the standard namespace prefixes for both `rdf` and `rdfs`, and adopt the SPECIAL vocabulary prefixes represented in *Listing* 3.1.

Listing 3.1: SPECIAL Namespace Prefixes

```
PREFIX spl: <http://www.specialprivacy.eu/langs/usage-policy#>
PREFIX splog: <http://www.specialprivacy.eu/langs/splog#>
PREFIX svd: <http://www.specialprivacy.eu/vocabs/duration#>
PREFIX svl: <http://www.specialprivacy.eu/vocabs/locations#>.
```

1.1 Usage policies.

Using the SPECIAL usage policy language it is possible to specify basic usage policies as OWL classes of objects, as denoted in *Listing* 3.2 (represented using the OWL functional syntax for conciseness). Whereby the permission to perform *SomeProcessing* of *SomeDataCategory* for *SomePurpose* has been given to *SomeRecipient* in compliance with *SomeStorage* restrictions.

Listing 3.2: Structure of a Usage Control Policy

```
ObjectIntersectionOf(
   ObjectSomeValuesFrom(spl:hasData (*@\textit{SomeDataCategory}@*))
   ObjectSomeValuesFrom(spl:hasProcessing (*@\textit{SomeProcessing}@*))
   ObjectSomeValuesFrom(spl:hasPurpose (*@\textit{SomePurpose}@*))
   ObjectSomeValuesFrom(spl:hasStorage (*@\textit{SomeStorage}@*))
   ObjectSomeValuesFrom(spl:hasRecipient (*@\textit{SomeRecipient}@*)))
```

1.2 Data processing and sharing events.

The SPECIAL policy log vocabulary is used to represent data processing and sharing events. The event log extract represented in *Listing* 3.3 (represented using turtle), relates to a new processing event corresponding to a data subject identified as `befit:Sue` on the `03.01.2018` at `13:20` (i.e., validity time). The event was recorded few seconds later (i.e., transaction time). The actual data captured can be traced via the `splog:eventContent` property, which is detailed in *Listing* 3.4, and usually stored in a separate knowledge base. While a hash of the content is stored in the event log.

Listing 3.3: A new event for Sue's BeFit device

```
befit:entry3918 a splog:ProcessingEvent;
splog:dataSubject befit:Sue;
dct:description "Store location in our database in Europe"@en;
splog:transactionTime "2018-01-10T13:20:50Z"^^xsd:dateTimeStamp;
splog:validityTime "2018-01-10T13:20:00Z"^^xsd:dateTimeStamp;
splog:eventContent befit:content3918;
splog:inmutableRecord befit:iRec3918.
```

Listing 3.4: The content of a new event for Sue's BeFit device

```
befit:content3918 a splog:LogEntryContent;
    spl:hasData svd:Location;
    spl:hasProcessing befit:SensorGathering;
    spl:hasPurpose befit:HealthTracking;
    spl:hasStorage [spl:haslocation svl:OurServers];
    spl:hasRecipient [a svr:Ours].
```

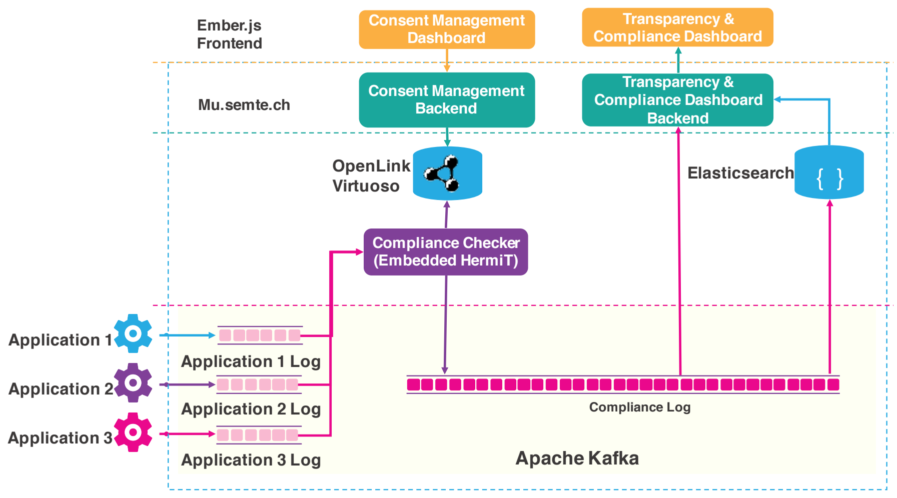


Figure 3.1: A Scalable Consent, Transparency and Compliance Architecture

1.3 Compliance checking.

In order to verify that data processing and sharing events comply with the corresponding usage policies specified by data subjects, we use OWL reasoning to decide whether the authorized operations specified by a data subject through their given consent, subsume the specific data processing records in the transparency log.

2 SPECIAL-K: our Apache Kafka based implementation

In this section, we describe the SPECIAL[3] consent, transparency and compliance system, which can be used not only to record consent but also to provide transparency to data subjects concerning the use of their personal data.

2.1 Background on Kafka

Apache Kafka[4] is an open-source stream-processing platform managed by the Apache Software Foundation, which provides publish and subscribe functionality to streams of data that are persisted in a fault-tolerant durable manner. Messages, which contain a key, value and timestamp are stored in Kafka topics (i.e. a unique identifier for the stream content), which can be partitioned across a cluster of machines.

2.2 SPECIAL-K: leveraging Kafka for transparency and compliance

The SPECIAL demo system architecture, which is depicted in *Figure* 3.1, enables transparency and compliance checking based on usage policies and events expressed using the aforementioned vocabularies.

[3]https://www.specialprivacy.eu/
[4]https://kafka.apache.org/

Kafka and Zookeeper. Data processing and sharing event logs are stored in the Kafka[5] distributed streaming platform, which in turn relies on Zookeeper[6] for configuration, naming, synchronization, and providing group services. Each application log is represented using a distinct Kafka topic, while a separate compliance topic is used to store the enriched log after compliance checks have been completed.

Virtuoso Triple Store Based on our current use case requirements, we assume that consent updates are infrequent and as such usage policies and the respective vocabularies are represented in a Virtuoso triple store.

Compliance Checker. The compliance checker, which includes an embedded HermiT[7] reasoner uses the consent saved in Virtuoso together with the application logs provided by Kafka to check that data processing and sharing complies with the relevant usage control policies. The results of this check are saved onto a new Kafka topic.

Elasticsearch. As logs can be serialized using JSON-LD, it is possible to benefit from the faceting browsing capabilities of Elasticsearch[8] and the out of the box visualization capabilities provided by Kibana.

Consent and Transparency & Compliance Backends. Interaction between the various architectural components is managed by mu.semte.ch[9] an open source micro-services framework for building RDF enabled applications.

Consent and Transparency & Compliance Dashboards. Users interact with the system via the consent management and the transparency and compliance dashboards. The former supports granting and revoking consent for processing/sharing. While, latter provides the data subject with transparency with respect to data processing and sharing events in a digestible manner.

3 SPIRIT: our Apache Spark based implementation

The work described in this section has been performed in close collaboration with the BDE members, Jens Lehmann[10] and Patrick Westphal[11]. 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.

3.1 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.

[5]https://kafka.apache.org/
[6]https://zookeeper.apache.org/
[7]http://www.hermit-reasoner.com/
[8]https://www.elastic.co/products/elasticsearch
[9]https://mu.semte.ch/
[10]Smart Data Analytics Group, University of Bonn, DE. jens.lehmann@cs.uni-bonn.de
[11]Institute for Applied Informatics (InfAI), University of Leipzig, DE. patrick.westphal@informatik.uni-leipzig.de

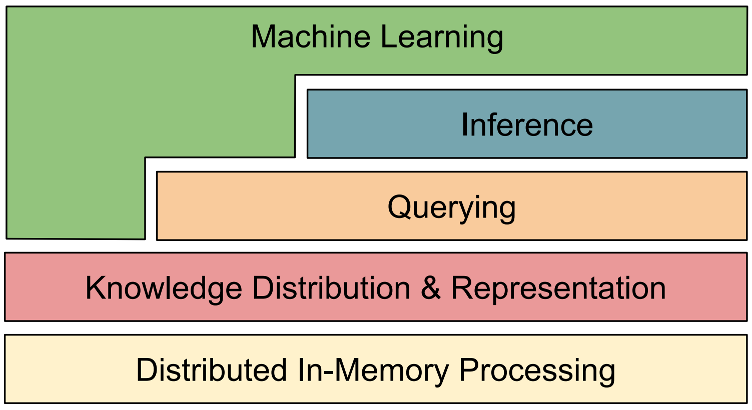

Machine Learning
Inference
Querying
Knowledge Distribution & Representation
Distributed In-Memory Processing

Figure 3.2: The SANSA Stack Layers

In terms of actual general purpose big data processing frameworks, Apache Hadoop[12], Apache Spark[13], and Apache Flink[14] 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[15] is an open source[16] semantic data processing stack that supports distributed computations 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 3.2) 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 *D1.7 Policy, transparency and compliance guidelines V2* and *D2.7 Transparency Framework V2*. In terms of data structures and programming interfaces SANSA follows the common and accepted representations of Apache Jena[17] and the OWL API[18]. 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[19]. 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

[12]Apache Hadoop, `http://hadoop.apache.org/`
[13]Apache Spark, `http://spark.apache.org/`
[14]Apache Flink, `http://flink.apache.org/`
[15]SANSA Stack home page, `http://sansa-stack.net`
[16]SANSA Stack on GitHub, `https://github.com/SANSA-Stack`
[17]Apache Jena, `http://jena.apache.org/`
[18]OWL API, `https://owlcs.github.io/owlapi/`
[19]SPARQL, `https://www.w3.org/TR/2013/REC-sparql11-query-20130321/`

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[20] 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, 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 [31], 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 [29] for e.g. link prediction, graph clustering and association rule mining techniques.

3.2 SPIRIT: leveraging the SANSA stack for transparency and compliance

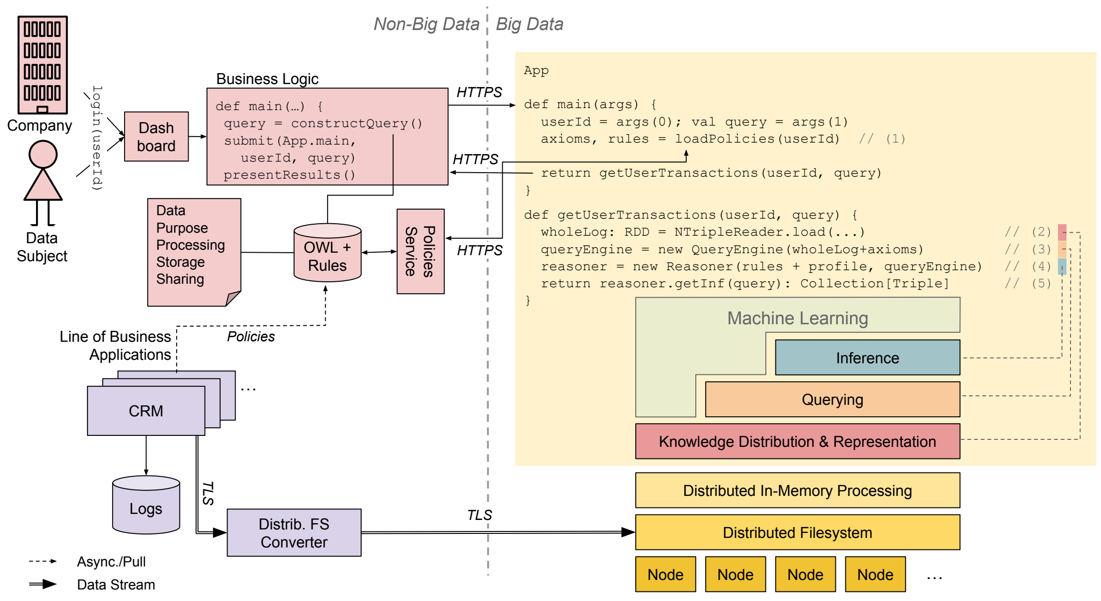
In the following, we present technical details of our transparency and compliance checking approach. Figure 3.3 sketches the transparency and compliance architecture and shows the components of the proposed architecture. We keep the notion of "Line of Business" applications as presented in deliverable *D2.7 Transparency Framework V2*.

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.

3.2.1 Transaction logs

When it comes to the ledger storing all data sharing and processing events (presented in deliverable *D2.7 Transparency Framework V2*), 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)*[21] fulfills

[20]RDFS, `https://www.w3.org/TR/2014/REC-rdf-schema-20140225/`
[21]HDFS, `http://hadoop.apache.org/`



```
def main(…) {
 query = constructQuery()
 submit(App.main,
   userId, query)
 presentResults()
```

```
App

def main(args) {
  userId = args(0); val query = args(1)
  axioms, rules = loadPolicies(userId)  // (1)

  return getUserTransactions(userId, query)
}

def getUserTransactions(userId, query) {
  wholeLog: RDD = NTripleReader.load(...)                  // (2)
  queryEngine = new QueryEngine(wholeLog+axioms)           // (3)
  reasoner = new Reasoner(rules + profile, queryEngine)    // (4)
  return reasoner.getInf(query): Collection[Triple]        // (5)
}
```

Figure 3.3: SPIRIT architecture exemplifying the transparency use case

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

3.2.2 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[23], OpenId Connect[24] or SAML2.0[25]. 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.

[22]Apache Flume, `http://flume.apache.org/`
[23]OAuth2, https://oauth.net/2/
[24]OpenId Connect, http://openid.net/connect/
[25]SAML2.0, https://wiki.oasis-open.org/security/FrontPage

3.2.3 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 Figure 3.3. 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 3.3 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 *D2.7 Transparency Framework V2*) 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 3.3):

- 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 3.3, 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.

3.3 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.

Knowledge distribution, representation & querying In *D1.7 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 [33]. Additionally different partitioning, indexing and materialisation strategies based on access policies and requests are required, in order to improve performance.

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.

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 introduced two consent, transparency and compliance architectures, the kafka based SPECIAL-K architecture and the spark based SPIRIT architecture. Ongoing work includes both the optimization of the existing prototypes and the benchmarking of these two alternative architectures against each other. Future work also includes the extension of these architectures to cater for compressed and encrypted data and privacy preserving machine learning.

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 show 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. D1.7 and [14]):

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 [18] proved that no deterministic protocol exists, in which there is no participation of a third party. Further research is discussed in [19], 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 [11]. 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 [26, 27] in the context of certified email, and in the setting of efficient fair exchange protocols for *generic items* in [3, 4, 9].

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 [24] 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 [6] 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 [23], 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 [7] 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;
- $A \rightarrow B : X$ means that A sends message X to B.

Now we are ready to recall Micali's fair protocol for certified email [27]. 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. $A \rightarrow B : Z$, where $Z = E^R_{TTP}(A \| B \| M)$
2. $B \rightarrow A : [Z]_B$
3. $A \rightarrow B : (M \| R)$.

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

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 $E_{TTP}^{R'}(A \| B \| M')$ 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 [30] (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 [30] 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 5, 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 $*URI$. After step 5, B can terminates the protocol with no problems.

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.

6 Formal verification of the protocol

The protocol has been validated by formally verifying that it actually enjoys the properties defined above, and therefore, it is fair.

The verification process of an exchange protocol relies on the creation of a model which describes the behaviour of the protocol itself. Once such a model is deployed, in the form of a finite state machine (FSM, for short), it is possible to express the desired fairness properties with an appropriate logic language and then apply the technique of model checking on the FSM. Clearly, the model checking of the FSM will succeed if and only if the protocol enjoys the required properties.

The formalism accepted by model checking tools is low-level, compared to the notation introduced above (calld AB notation), since the latter is not precise enough. The AB notation, indeed, is a user-friendly specification and therefore too abstract. For this reason we adopted the specification language HLPSL, developed in the european project Automated Validation of Internet Security Protocols and Applications (Avispa)[2]. The main feature of HLPSL is the

[2]http://www.avispa-project.org/

description of the behaviour of each party of the protocol in the form of roles. In particular, it allows to define operators for send and receive messages using channels, and to combine atomic messages in order to compose messages of complex type. A more detailed description of HLPSL syntax is provided in the Appendix.

An important feature of HLPSL is the presence of an *intruder*, that is an additional role whose goal is to break the protocol. We adopt *Dolev-Yao*'s intruder model, where the attacker has full control of communication channels. Hence, the intruder is able to intercept and then replay, suppress or forward every message, except those exchanged with the TTP. The intruder also gains knowledge by decomposing and analysing messages, though it cannot break cryptography.

HLPSL is supported by several tools, such as OFMC[10], CL-Atse[32], SATMC[2] and TA4SP[13]. However, these tools do not provide resilient communication channels. This feature is necessary to prove fairness properties, since no protocol can guarantee fairness if the intruder is able to suppress the messages towards one party[21]. Hence, the communication channels between parties and TTP are assumed to be resilient. As model checker, therefore, we used the tool TPMC[12], presented in the 21st European Conference on Modelling and Simulation. It allows to define both resilient and unreliable channels. TPMC is based on THLPSL, and, as opposed to HLPSL, also provides specification of temporal aspects. Moreover, the tool includes a translator from HLPSL specification to compositions of parallel automata, which can then be model checked by the UPPAAL engine. The language used to express the fairness properties supported by UPPAAL is a fragment of CTL[17] (see in Appendix).

The version of the protocol we verified is a refinement of the one presented above, where the signature on some messages that are not necessary to ensure the satisfiability of the properties are removed.[3] More specifically, Step 7, 10 and 11 of the protocol do not includes the signature anymore. The entire validated 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. $A \to B : Z_1$, 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. $B \to A : [Z_1]_B$

5. $A \to B : Z_2$, where $Z_2 = [A \| B \| URI \| E_{TTP}(k) \| t_0]_A$

6. $B \to A : [Z_2]_B$

7. $A \to B : E_B(k)$; 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. $B \to TTP : [Z_2]_B$

[3]In practice, however, it is useful to keep those signatures since they make it possible to discover attacks earlier.

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

The fairness properties mentioned above has been verified by means of the following conditions:

1. if A has the receipt $R_2 = [Z_2]_B$, then A has obtain the receipt $R_1 = [Z_1]_B$;
2. if B has the session key k, then A can obtain the receipt $R_2 = [Z_2]_B$;
3. if A has the receipt $R_2 = [Z_2]_B$, then B can obtain the session key k.

It is easy to see that whenever B has the session key k and, therefore, has access to the dataset D, than the first two conditions imply that A obtains the two receipts $R_1 = [Z_1]_B$ and $R_2 = [Z_2]_B$, as required by Property 1.

Property 2 requires that if a problem occurs before Step 5, then A has at most one of the two receipts. Due to the first condition, A cannot have the receipt $R_2 = [Z_2]_B$, and, as consequence of the second condition, B cannot obtain the session key k. After Step 5, three cases may arise: (1) B can abort the protocol, thus neither A obtains $R_2 = [Z_2]_B$, nor B obtains k; (2) B sends to A the receipt $R_2 = [Z_2]_B$, hence, by the third condition, B obtains the session key k; (3) B sends to TTP the message $[Z_2]_B$. In latter case, as a consequence of the fact that the communication channels between the TTP and the parties are resilient, B obtains k and A obtains the receipt $R_2 = [Z_2]_B$. Observe that this case also implies Property 3 and 4.

The secrecy constraint of Property 5 has been verified during the creation of the FSM, since the tool TPMC in this process generates for each party the maximal knowledge it can obtain by means of derivation rules. Hence, in the corresponding FSM, it turns out that neither the TTP nor the intruder own the symmetric key k.

Property 6 and 7 are implicitly verified adopting the model Dolev-Yao, since this type of intruder can perform both man-in-the-middle and replay attacks.

Finally, Properties 8 and 9 are not security properties and do not require the verification of logical formulas. Concerning 8, note that the fairness properties of the protocol do not guarantee that the dataset D is meaningful. However, they ensure that, at the end of the protocol, B has both of the messages sent by A, by means of which it can prove that A transferred "bad" data (if D actually consists of rubbish). Concerning Property 9, it is well known that the TTP may become a bottleneck. To certify that this problem is mitigated in the proposed protocol, Property 9 ensures that the workload of TTP is independent from the size of the dataset D. This property can be confirmed by analysing the implementation of the TTP role in the HLPSL language, where one can observe that TTP never needs to process D to complete its tasks.

The protocol can abort due to a time-out either as consequence of message suppression by the intruder or due to problems of the parties. These cases are simulated by means of the non-deterministic choices in the control flow of the specification of the parties. Intuitively, at every step the parties can choose either to proceed with the exchange or to stop the communication (thereby modelling hardware/software faults, party misbehavior, and failure of the cryptographic verification steps). The non-determinism has also been used to model the case in which at Step 3 the dataset D addressed by the URI does not match the hash value $h(*URI)$, as well as the case at Step 8 in which the session key k cannot decrypt the dataset D.

We successfully validated the protocol with one session and two interleaved sessions (where the intruder may try to use the messages of one session to attack the other). The validation with three sessions required more computational resources than those available, as the size of the FSM grows exponentially with the number of sessions. We can only report that the verification of the protocol with three sessions did not find any errors before running out of memory.

Appendix

Technical details of the formal protocol verification

The specification language used to describe the input protocol of TPCM is HLPSL[4]. Therefore, in this section, we first describe its main features and then show how the protocol is reformulated in this language.

HLPSL is a role-based language, containing two type of roles: *basic* and *composition*. A basic role is a module that describes the behaviour of a party, consisting of its initial knowledge and the rules expressing how the knowledge can change. A composition role, instead, is the parallel instantiation of basic roles (which is useful for checking multiple parallel sessions, for example).

The knowledge of a party is represented by the content of the set of parameters, constants and local variables of its role. Each data is typed and the types provided are the following: nat (natural numbers), bool (binary flags), agent (party names), symmetric_key and public_-key (cryptographic keys), channel (communication channels), text (atomic messages), message (complex messages), function (hash functions), role_instance (instance identifiers).

The type channel has additional parameters to specify (1) the model of intruder (DY for the Dolev-Yao model) and (2) the reliability of the communication, that can be operational (delivery guaranteed within a bounded time interval), resilient (delivery eventually guaranteed), and unreliable (delivery not guaranteed).

Examples of complex messages are: A.B with A and B messages, H(A) with H function and A messages, A_K or A_inv(K) with A messages and K symmetric or public key. Obviously, a text is also a message.

The knowledge of a basic role identifies a state and its actions are transitions describing changes of the role knowledge. The new value that a variable A takes in a transition is represented by a primed variable, e.g., A'. The transition schema is

```
label. left-predicate =|> right-predicate
```

The label uniquely identifies the transition. The left-side predicate, instead, specifies when the transition is enabled by means of boolean expressions and it can include message delivery. Finally the right-side predicate describes the action of the role and it can include send operations.

We can now describe the formulation of the protocol in HLPSL. The proposed fair protocol has four roles: the two parties (Alice and Bob) and the TTP are basic roles, the main role, here called environment, is a composite role.

Alice has the following role:

```
role Alice(
```

[4]http://www.avispa-project.org/

```
  A, B : agent,
  SND, RCV : channel(dy, 0, inf, unreliable),
  RCVTTP : channel(dy, 0, 0, resilient),
  KA, KB, KTTP : public_key,
  RI : role_instance)

  played_by A def=

  local  State : nat,
  URI,HASHDATA,T : text,
  R1,R2,R3,KT : message,
  K : symmetric_key

  init State = 0

  transition
  alice1. State = 0 =|> State' = 1 /\ SND({A.B.URI.T.HASHDATA}_inv(KA))
  alice2. State = 1 /\ RCV(R1') /\ R1' = {{A.B.URI.T.HASHDATA}_(KA)}_(KB) =|> State' = 2
  alice3. State = 2 =|> State' = 3 /\ KT' = {K}_(KTTP) /\ SND({A.B.URI.T.KT}_inv(KA) )
  alice4. State = 3 /\ RCV(R2') /\ R2' = {{A.B.URI.T.KT}_(KA)}_(KB) =|> State' = 4
  alice5. State = 3 /\ RCVTTP(R3') /\ R3' = {{A.B.URI.T.KT}_(KA)}_(KB) =|> State' = 5
  alice6. State = 4 =|> State' = 5 /\ SND({K}_(KB) )
  alice7. State = 1 =|> State' = 5
  alice8. State = 2 =|> State' = 5
  alice9. State = 4 =|> State' = 5
end role
```

In detail, the parameters A and B are the name of parties, SND and RCV are the two unreliable comunication channels between A and B, SCVTTP is the resilient channel on which A receives messages from the TTP, finally KA, KB and KTTP are public keys and RI is the identifier of the role instance. Local variable are used for information independent of the instantiation, in particular State represents the current state. Alice has six states: (0) is the initial state; (1) identifies the state in which A has sent Z_1 to B and is expecting the receipt $[Z_1]_B$ from B; in state (2) A has obtained receipt $[Z_1]_B$; (3) identifies the state in which A has sent Z_2 to B and is expecting the receipt $[Z_2]_B$ from B; in state (4) A has obtained receipt $[Z_2]_B$; finally (5) is the final state. There are also nine transitions, corresponding to the steps reported in Section 5: (1) represents Step 2; (2) represents Step 4; (3) represents Step 5; (4) represents Step 6; (5) represents Step 11; (6) represents Step 7; finally (7), (8) and (9) allow for time-out simulation.

The role of Bob is the following:

```
role Bob(
  A, B : agent,
  SND, RCV : channel(dy, 0, inf,unreliable),
  SNDTTP, RCVTTP : channel(dy, 0, 0, resilient),
  KA, KB, KTTP: public_key,
  RI : role_instance)

  played_by B def=

  local State : nat,
  URI,HASHDATA,T : text,
  K : symmetric_key,
  R1,R2,S1,S2,KT : message
  init State = 0

  transition
  bob1. State = 0 /\ RCV(R1’) /\ R1’ = {A.B.URI’.T’.HASHDATA’}_(KA) =|> State’ = 1
  bob2. State = 1 =|> State’= 2 /\ S1’ = {R1}_inv(KB) /\ SND(S1)
  bob3. State = 1 =|> State’= 3
  bob4. State = 2 /\ RCV(R2’) /\ R2’ = {A.B.URI.T.KT’}_(KA) =|> State’ = 4
  bob5. State = 4 =|> State’= 5 /\ S2’ = {R2}_inv(KB) /\ SND(S2)
  bob6. State = 5 =|> State’= 7 /\ SNDTTP(S2)
  bob7. State = 5 /\ RCV({K’}_inv(KB)) /\ KT = {K’}_(KTTP) =|> State’ = 6
  bob8. State = 6 =|> State’= 3
  bob9. State = 6 =|> State’= 7 /\ SNDTTP(S2)
  bob10. State = 7 /\ RCVTTP({K’}_inv(KB)) /\ KT = {K’}_(KTTP) =|> State’= 3
  bob11. State = 0 =|> State’= 3
  bob12. State = 2 =|> State’= 3
  bob13. State = 4 =|> State’= 3
  bob14. State = 5 =|> State’= 3
end role
```

In detail, in addition to the parameters of Alice, Bob has also the send resilient channel to communicate with the TTP. Bob has seven states: (0) is the initial state in which is expecting the message Z_1 from A; (1) identifies the state in which B has received Z_1; in state (2) B has sent the receipt $[Z_1]_B$ to A and is expecting the message Z_2 from A; (3) is the final state; in state (4) B has received Z_2; in state (5) B has sent the receipt $[Z_2]_B$ to A and is expecting the session key from A; (6) identifies the state in which B has received the session key from A; finally (7) identifies the state in which B has sent to TTP the message Z_2. There are also fourteen transitions, corresponding to the steps reported in Section 5: (1) represents Step 2; (2) represents Step 4; (3) represents the abort of Step 3 in case of mismatch; (4) represents Step 5; (5) represents Step 6; (6) represents Step 9; (7) represents Step 7; (8) represents the succeed of the verification at Step 8; (9) represents Step 9 as consequence of the failure of the verification at Step 8; (10) represents Step 10; finally (11), (12), (13) and (14) allow for time-out simulation.

TTP has the following role:

```
role TServer(
  A, B, TTP : agent,
  SNDTTPA, SNDTTPB, RCVTTPB : channel(dy, 0, 0, resilient),
  KA, KB, KTTP : public_key,
  RI : role_instance)

  played_by TTP def=

  local State : nat,
  URI,T : text,
  K : symmetric_key,
  M : message

  init State = 0

  transition
  sttp1. State = 0 /\ RCVTTPB(M’) /\ M’ = {{A.B.URI’.T’.{K’}_inv(KTTP)}_(KA)}_(KB) =|> State’ = 1
  sttp2. State = 1 =|> State’ = 2 /\ SNDTTPB({K}_(KB))
  sttp3. State = 2 =|> State’ = 3 /\ SNDTTPA(M)
end role
```

In detail, TTP has the same parameters as Alice and Bob, except for the communication channels. Indeed, SCVTTP is the resilient channel on which TTP sends messages to A, SNDTTPB and RCVTTPB are the two resilient channel of comunication with B. TTP has three states: (0) is the initial state in which is expecting the message $[Z_1]_B$ from B; (1) identifies the state in which TTP has received $[Z_1]_B$; in state (2) TTP has sent the session key to B; finally in state (3) TTP has sent the receipt $[Z_2]_B$ to A. There are three transition for the TTP: (1) represents Step 9; (2) represents Step 10; and (3) represents Step 11.

Finally, the composite role has no parameters and declares the variable used to instantiate the basic roles. It also defines the initial knowledge of the intruder. The following code instantiates one session, therefore, it declares one copy of each basic role:

```
role Env( ) def=
   const a, b :agent,
   ka, kb, kttp : public_key,
   ki : symmetric_key,
   sa, ra, sb, rb: channel(dy,0,inf, unreliable),
   sttpa, sttpb, rttpb : channel(dy,0,0, resilient)

   knowledge(i)={a, b, ka, kb, kttp, ki}

   composition
      Alice(a, b, sa, ra, rttpa, ka, kb, kttp, 0) /\
        Bob(a, b, sb, rb, sttpb, rttpb, ka, kb, kttp, 1) /\
          TServer(a, b, ttp, rttpa, rttpb, sttpb, ka, kb, kttp, 2)
end role
```

The composition of two session is, instead, the following:

```
   composition
      Alice(a, b, sa, ra, rttpa, ka, kb, kttp, 0) /\
        Bob(a, b, sb, rb, sttpb, rttpb, ka, kb, kttp, 1) /\
          TServer(a, b, ttp, rttpa, rttpb, sttpb, ka, kb, kttp, 2)
      Alice(a, b, sa, ra, rttpa, ka, kb, kttp, 3) /\
        Bob(a, b, sb, rb, sttpb, rttpb, ka, kb, kttp, 4) /\
          TServer(a, b, ttp, rttpa, rttpb, sttpb, ka, kb, kttp, 5)
```

The language used to express the fairness formulas is a fragment of CTL[17], since nested formulas are not allowed. Given a formula φ it is possible to express reachability properties $E\Diamond\varphi$ (φ can be eventually satisfied), invariants $A\Box\varphi$ (φ is always satisfied), safety properties $E\Box\varphi$ (potentially φ is always satisfied) and liveness properties $A\Diamond\varphi$ (always φ is eventually satisfied).

Before describing the formulas used in the verification process, we need to discuss how the resiliency of the communication channels has been modelled. When a protocol in HLPSL has a resilent channel, the resulting FSM, generated by TPMC, has an additional variable I, that is an array of resilient messages $irsl$. Every time a message is sent through a resilient channel, the corresponding message $irsl$ in I is flagged. Once it is delivered, its flag is reset. In this way it is possible to identify which states are conform to the resiliency constraint.

We can now recall the conditions introduced in Section 5:

1. if A has the receipt $R_2 = [Z_2]_B$, then A has obtained the receipt $R_1 = [Z_1]_B$;
2. if B has the session key k, then A can obtain the receipt $R_2 = [Z_2]_B$;
3. if A has the receipt $R_2 = [Z_2]_B$, then B can obtain the session key k.

The form of these conditions is φ imply ψ and can be expressed in CTL as $A\Box(\varphi$ imply $A\Diamond\psi)$ or succinctly as φ –> ψ. The corresponding formula of the first condition with CTL, in the succinct form, is the following:

```
((a_Alice_0.N[11]==b_Bob_0.N[11] and a_Alice_0.N[11]!=-1)
or
(a_Alice_0.N[12]==b_Bob_0.N[11] and a_Alice_0.N[12]!=-1))
-->
((forall(i:irsl) I[i]==0)
imply
(b_Bob_0.N[8]==a_Alice_0.N[7] and b_Bob_0.N[8]!=-1))
```

In the above formula, φ represents the predicate "A has the receipt R_2" and is composed of the OR of two formulas: the first one expresses the condition that A has received R_2 form B and the second one that A received R_2 from TTP. On the right-hand side of the formula, instead, ψ represents the predicate "A has obtained the receipt R_1" in the form ψ_1 imply ψ_2, where ψ_1 requires that the messages sent through the resilient channels have been delivered, while ψ_2 expresses the condition that A has obtained the receipt R_1.

The formula of the second condition is the following:

```
((b_Bob_0.N[12]==a_Alice_0.N[9] and b_Bob_0.N[12]!=-1)
and
(a_Alice_0.N[10]==b_Bob_0.N[9] and a_Alice_0.N[10]!=-1))
-->
((forall(i:irsl) I[i]==0)
imply
((a_Alice_0.N[11]==b_Bob_0.N[11] and a_Alice_0.N[11]!=-1)
or
(a_Alice_0.N[12]==b_Bob_0.N[11] and a_Alice_0.N[12]!=-1)))
```

As before, φ represents the predicate "B has the session key k" and is composed of the AND of two formulas: the first one expresses that B has the session key k, and the second that this key match with key contained in the receipt R_2. On the right-hand side of the formula, instead, ψ represents the predicate "A can obtain the receipt R_2" in the form ψ_1 imply ψ_2, where ψ_1 requires that the messages sent through the resilient channels have been delivered, while ψ_2 expresses the condition that A obtains the receipt R_2 from B or from TTP.

Finally, the formula of the third condition is the following:

```
(((a_Alice_0.N[11]==b_Bob_0.N[11] and a_Alice_0.N[11]!=-1)
or
(a_Alice_0.N[12]==b_Bob_0.N[11] and a_Alice_0.N[12]!=-1))
and
(a_Alice_0.N[10]==b_Bob_0.N[9] and a_Alice_0.N[10]!=-1))
-->
((forall(i:irsl) I[i]==0)
imply
(b_Bob_0.N[12]==a_Alice_0.N[9] and b_Bob_0.N[12]!=-1))
```

The antecedent of the operator "–>" represents the predicate "A has the receipt R_2" and is composed of the AND of two formulas: the first one expresses that A has received the receipt R_2 from B or from TTP and the second that the key contained in the R_2 match with the encription key of the dataset. On the right-hand side of the formula, instead, ψ represents the predicate "B can obtain the session key k" in the form ψ_1 imply ψ_2, where ψ_1 requires that the messages sent through the resilient channels have been delivered, while ψ_2 expresses the condition that B obtains the session key k.

Bibliography

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

[2] A. Armando, R. Carbone, and L. Compagna. SATMC: a sat-based model checker for security protocols, business processes, and security apis. *STTT*, 18(2):187–204, 2016. doi: 10.1007/s10009-015-0385-y. URL `https://doi.org/10.1007/s10009-015-0385-y`.

[3] 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`.

[4] 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`.

[5] 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.

[6] 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`.

[7] 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`.

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

[9] 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`.

[10] D. A. Basin, S. Mödersheim, and L. Viganò. OFMC: A symbolic model checker for security protocols. *Int. J. Inf. Sec.*, 4(3):181–208, 2005. doi: 10.1007/s10207-004-0055-7. URL `https://doi.org/10.1007/s10207-004-0055-7`.

[11] 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`.

[12] M. Benerecetti, N. Cuomo, and A. Peron. TPMC: A model checker for time-sensitive security protocols. *JCP*, 4(5):366–377, 2009. doi: 10.4304/jcp.4.5.366-377. URL `https://doi.org/10.4304/jcp.4.5.366-377`.

[13] Y. Boichut, P.-C. Héam, and O. Kouchnarenko. Automatic verification of security protocols using approximations. Technical report, INRIA, 2005.

[14] 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.

[15] P. A. Bonatti. Fast compliance checking in an OWL2 fragment. In J. Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.*, pages 1746–1752. ijcai.org, 2018. ISBN 978-0-9992411-2-7. doi: 10.24963/ijcai.2018/241. URL `https://doi.org/10.24963/ijcai.2018/241`.

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

[17] E. C. E. Emerson. Design and synthesis of synchronisation skeletons using branching time temporal logic. *Logic of Programs*, 131:52–71, 1981.

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

[19] 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`.

[20] B. C. Grau and B. Motik. Reasoning over ontologies with hidden content: The import-by-query approach. *J. Artif. Intell. Res.*, 45:197–255, 2012. doi: 10.1613/jair.3579. URL `https://doi.org/10.1613/jair.3579`.

[21] F. C. G. H. Pagnia. On the impossibility of fair exchangewithout a trusted third party. Technical report, TUD, 1999.

[22] Y. Kazakov, M. Krötzsch, and F. Simancik. The incredible ELK - from polynomial procedures to efficient reasoning with ontologies. *J. Autom. Reasoning*, 53(1):1–61, 2014. doi: 10.1007/s10817-013-9296-3. URL `https://doi.org/10.1007/s10817-013-9296-3`.

[23] 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`.

[24] A. Küpçü 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`.

[25] T. Lyons, l. Courcelas, and K. Timsit. blockchain and the gdpr. 2018. URL `https://www.eublockchainforum.eu/reports`.

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

[27] 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`.

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

[29] 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.

[30] 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`.

[31] 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.

[32] M. Turuani. The CL-Atse protocol analyser. In *Term Rewriting and Applications, 17th International Conference, RTA 2006, Seattle, WA, USA, August 12-14, 2006, Proceedings*, pages 277–286, 2006.

[33] 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.