Heiko Roßnagel, C. H. Schunck, S. Mödersheim (Hrsg.): Open Identity Summit 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 107

Adapting the TPL Trust Policy Language for a Self-Sovereign Identity World

Lukas Alber[1], Stefan More[2], Sebastian Mödersheim[3], Anders Schlichtkrull[4]

Abstract: Trust policies enable the automated processing of trust decisions for electronic transactions. We consider the Trust Policy Language TPL of the LIGHT^{est} project [Mö19] that was designed for businesses and organizations to formulate their trust policies. Using TPL, organizations can decide if and how they want to rely on existing trust schemes like Europe's eIDAS or trust scheme translations endorsed by them. While the LIGHT^{est} project is geared towards classical approaches like PKI-based trust infrastructures and X.509 certificates, novel concepts are on the rise: one example is the self-sovereign identity (SSI) model that enables users better control of their credentials, offers more privacy, and supports decentralized solutions. Since SSI is based on distributed ledger (DL) technology, it is a question of how TPL can be adapted so that organizations can continue to enjoy the benefits of flexible policy descriptions with automated evaluation at a very high level of reliability.

Our contribution is a first step towards integrating SSI and the interaction with a DL into a Trust Policy Language. We discuss this on a more conceptual level and also show required TPL modifications. We demonstrate that we can integrate SSI concepts into TPL without changing the syntax and semantics of TPL itself and have to add new formats and introduce a new built-in predicate for interacting with the DL. Another advantage of this is that the "business logic" aspect of a policy does not need to change, enable re-use of existing policies with the new trust model.

Keywords: Trust policies, Accountability, Security, LIGHTest, eIDAS, SSI

1 Introduction

Automated trust decisions are an essential component of many business and government processes. When Alice sends an electronic transaction to Bob, the latter must verify Alice's signature on the transaction, in particular, that it is the person claimed here and it is legally binding. That, in turn, relies on trust in the organization that certified Alice in terms of a credential or in an organization that translates this trust in some way. Bob's prerogative is to determine his policy, i.e., which issuers or trust schemes to trust and what requirements to ask from Alice.

[1] Graz University of Technology, Inffeldgasse 16a, Graz, Austria lukas.alber@iaik.tugraz.at
[2] Graz University of Technology, Inffeldgasse 16a, Graz, Austria stefan.more@iaik.tugraz.at
[3] Technical University of Denmark, DTU Compute, Richard Petersens Plads, Bygning 324, 2800 Kongens Lyngby, Denmark samo@dtu.dk
[4] Aalborg University Copenhagen, Department of Computer Science, A.C. Meyers Vænge 15, 2450 Copenhagen SV, Denmark andsch@cs.aau.dk

The trust verification system introduced by the LIGHT^{est} project [BL16] enables this: a high level of assurance, a rich and flexible Trust Policy Language (TPL), and automated policy verification. In this context, the term *transaction* is used in a quite liberal way: it can include, for instance, a bid in an auction house or a simple login at an online system.

The Trust Policy Language was introduced by Mödersheim et al. [Mö19] inspired by existing concepts of trust policy systems [BFG10; DD10; GN08; He00; LFG99]. Using a policy language is obviously more flexible and declarative than rules hard-coded in an implementation. TPL is in particular based on logic-programming and uses a Prolog-style syntax and semantics, allowing to separate "business logic" from lower-level aspects of the evaluation like interfacing with trust servers.

Centralized trust and identity management approaches represent an attractive target for criminals and cyber war [Fr20]. Furthermore, storing identity data in centralized data silos increases the likelihood and impact of data breaches [Be17; Be20; Th17]. For this reason, in recent years, more decentralized trust management architectures caught the research community's attention.

Our goal is to integrate support for such modern identity solutions with TPL and trust policies in general. We focus on two concepts: distributed ledgers (DL) and self-sovereign identity (SSI). DLs decentralize storage and governance helps to mitigate issues like a single point of failure, split-world attacks, and the loss of identity retention [Ko20]. SSI describes the concept of storing identities decentralized at the owner, also called holder, instead of centrally at an identity provider. The SSI community furthermore introduced several concepts regarding credentials [SLC19] and identities [Re21; SS20] while ensuring interoperability, privacy, and decentralized storage by combining SSI with DLs and other promising technologies (e.g., zero-knowledge proofs).

In this paper, we present a vision of bringing LIGHT^{est}'s world of automated trust decisions together with novel concepts of SSI and DLs. Our contribution is to show how this integration can be made without changing the syntax and semantics of TPL itself. That has several advantages. First, the tools based on TPL do not need substantial modification: the logical evaluation stays the same, and only new modules for the interfacing with the DL and parsers for new document formats are needed. Indeed, the connection of the theorem prover RP_X that provides LIGHT^{est} trust decisions with an independent verification does not require any changes. Second, a policy's "business logic" does not have to change either. That makes it possible to formulate policies accepting classical eIDAS credentials and modern SSI-credentials in a uniform way, making the lower level only a "technology-choice".

The rest of this paper is structured as follows: In Sect. 2 we give an overview of DL and DL-based Trust Management, SSI, and TPL. We discuss our position on extending TPL for the DL/SSI world in Sect. 3 and show the extension by example in Sect. 4. We discuss the support of additional SSI concepts and conclude the paper in Sect. 5 with an outlook into a concrete implementation.

2 Background

Distributed Ledger (DL)-based Trust Management A distributed ledger (DL) is a decentralized data storage model. The data is stored redundantly at several distributed nodes maintained by different entities, improving resilience. Each node preserves independent control, and they agree on a common state by running a consensus protocol [Xi20]. A DL is often referred to as "blockchain", although it indicates a subset of the ledger technology. Access-wise, DLs support a large spectrum of models, mostly grouped and described by the terms public, private, permissionless, and permissioned [Zh18].

The **Self-Sovereign Identity (SSI)** community aims to replace centralized or federated identity concepts [ZZS14] with a decentralized model, i.e., keeping the identity in the holder's sole possession [Ab17]. In the SSI model, a user creates a **Decentralized Identifier (DID)** for its identity [Re21; SS20] and publishes it by defining a DID document (DIDoc) containing the DID, a corresponding public key, and other application dependent information. That is often done using a DL.

The vision of SSI is that any party can certify attributes of any other party, e.g., a higher education institution can accredit graduation to a student, or an interior ministry can attest someone's date of birth. For such certifications it is common to use the **Verifiable Credentials (VCs)** data model [SLC19]. This specification defines a generic way of packaging claims and the corresponding issuing authority in a signed JSON document.

Trust Policy Language (TPL) In a previous publication [Mö19], we introduced TPL to enable service providers to flexibly define and run an automated process for deciding when to accept a transaction, e.g., based on whether the signatures and certificates that come with the transaction are sufficient for the service providers to consider the transaction's sender trustworthy. TPL was initially created in the context of the LIGHT^{est} project [BL16; Ro17; Wa19].

A TPL policy is a list of Horn clauses in the syntax of Prolog. Each Horn clause is in the form $p(t)$:- $q_1(u_1), ..., q_n(u_n)$. meaning: if all the $q_i(u_i)$ are true, then also $p(t)$ is true. We refer to the Horn clauses as rules. A set of rules of the form $p(t)$:- $q_1(u_1), ..., q_n(u_n)$. for the same p defines the *predicate* p. Rules can be evaluated in the same way as Prolog would: To see if a query $p(s)$ succeeds, find a suitable rule; e.g. the above $p(t)$:- $q_1(u_1), ..., q_n(u_n)$. if s and t can be unified. Then apply the resulting unifier to all $q_i(u_i)$ and evaluate them; if the evaluation for the subqueries evaluates to success, then we say that the original query also evaluated to a success. If that is not the case, then try again with the next suitable rule if any such exists. The subqueries are evaluated in the same way by recursion, except for the case where a q_i is a so-called built-in predicate, i.e., a predicate that interacts with, e.g., servers or formats. In that case, the interaction is performed, which either results in a success or a failure. We notice that the rules define the policy in a positive way: the transaction is rejected

if and only if in every applicable rule at least one of the conditions is not met. For a policy where the above procedure takes too long, we reject the transaction based on a timeout.

TPL features built-in predicates which essentially wrap context-specific discovery and verification logic. In the LIGHT^{est} context that includes all the interactions with Europe's eIDAS and comparable trust schemes. The Automated trust discovery and the verification of trust status information use DNSSEC [Ar05] for security and authenticity protection. Therefore, the application of TPL in LIGHT^{est} focused on centralized PKI, although means for a trust translation between different trust schemes have been presented. For a more in-depth description see our previous publication on the topic [Mö19]).

3 Concept

In this section, we introduce our vision of an SSI extension to the TPL system. We give an overview of the process flow typical in the SSI world to create and register an identity, acquire credentials of identity attributes, provide them with legal value, and use these credentials to authenticate at service providers (SPs). At the same time, we show how this flow integrates into the TPL infrastructure. Further, we discuss the policy system's functionalities and show the extensions needed to support the described SSI model. Finally, we conclude the chapter with a discussion on the consequences concerning accountability of integrating TPL with SSI.

3.1 Step by Step Flow

To derive qualified identity credentials for our purpose in SSI, we utilize the process introduced by Abraham et al. [Ab20]. A holder (user holding a credential) needs to generate a decentralized identifier (DID), and a corresponding DID document, let the DID document get registered at a DL, and acquire legal identity credentials. Only then the holder can attempt to authenticate herself to a service provider (cf. Fig. 1).

To **generate a DID**, the holder first generates a public/private key pair. From the public key, they derives a decentralized identifier (DID) that can refer to them as the holder in a privacy-friendly way. The holder can then choose what credentials she wants to obtain for this DID, e.g., one certifying only their birth date.

To **acquire credentials** for their identity attributes, the holder first authenticates at an Identity Provider (IdP) such as their government's E-ID system. Additionally, the holder needs to prove the ownership of their DID to the IdP. That is done by initiating a challenge-response protocol with the IdP using the holder's DID keys. The IdP then uses the public key from the transmitted DID document to verify the signature that resulted from the challenge-response. After the holder has successfully proven the ownership, the IdP system generates various identity credentials. Each of those credentials contains one of the holder's identity attributes.

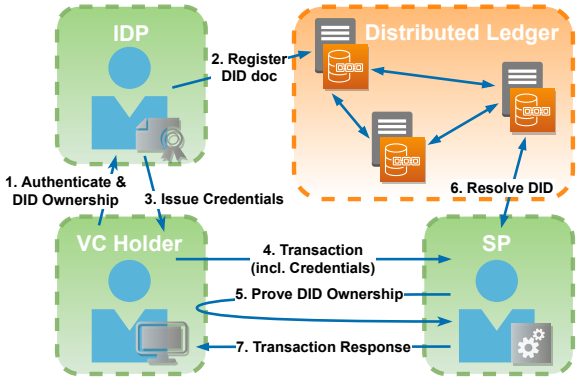

Fig. 1: Architectural overview of our concept. After retrieving credentials from the identity provider (IdP), the VC Holder sends an electronic transaction (including the verifiable credentials) to the service provider (SP) and proves ownership of their DID. The service provider then uses an Automated Trust Verifier (ATV) to interpret the TPL policy, resolve the DID document using a DL, and forms an automated trust decision about the transaction.

The VCs are bound to the holder by adding their DID as a credential subject. The IdP system signs the credentials and sends them to the holder. The IdP also adds the holder's DID documents to the DL, enabling retrieval by all other nodes.

Using SSI, a holder can now **authenticate to a service provider** with the registered DID and the obtained identity credentials. First, the holder provides one or more identity credentials to the service provider, which the service provider can use for authentication. The holder also provides additional transaction information, signed with the secret key corresponding to their DID. The holder also proves their DID ownership to the service providerby including a challenge provided by the SP in the signed transaction. To verify the signature on the transaction, the service provider discovers and retrieves the holder's DID document from the DL and uses the contained public key. To ensure a qualified issuer issued the holder's identity credentials, the service provider needs to **authenticate the issuer**. In our case, the issuer is the IdP. That ensures that the IdP is indeed a qualified issuer and that it indeed signed the credentials. If this authentication succeeds, the service provider trusts the attributes contained in the credentials.

In LIGHT^{est} this authentication uses a trust infrastructure, e.g., the eIDAS trust scheme. To that end, the service provider uses an automated trust verification system (ATV). The ATV uses TPL to interpret the holder's policy that enables the definition of custom trust rules. TPL's built-in predicates allow specifying queries to servers that the ATV must perform.

Adding SSI Support: To support the SSI model, we introduce the new built-in predicate `resolveDID`. With this predicate, users can create policies that use information stored on the

DL (cf. Sect. 3.3). Further, we add support for several SSI data-structures by introducing new TPL *formats* for each of them (cf. Sect. 3.2).

We show that all necessary modifications can be made within the extensible built-in predicate and formats system of TPL. Hence, no syntax changes are needed, and formal properties of TPL and the accountability of policy evaluations are not weakened, as shown in Sect. 3.4.

3.2 New TPL Formats

In TPL, the concept of formats is used to connect policies with parsers that extract values from complex data formats and ensure compliance with data schemas [Mö19]. These values can be concrete numbers, constants, or of some format themselves. The `extract` predicate is used in a policy to extract values, e.g. we write `extract(Form, certificate, Certificate)` to extract a certificate from a form into the variable `Certificate`. The special key `format` is used to specify what format we expect, e.g. we write `extract(Certificate, format, x509)` if we require the certificate to be an x509 certificate. Consequently, to make additional data used in SSI accessible in TPL, we introduce two new formats:

`ssi_credential` is the format for Verifiable Credential (VC). The format contains the VC's fields and supports verification of the VC's signature. For example, if we have a VC with a `birthday` attribute, we can directly call `extract(Credential, format, ssi_credential)`, `extract(Credential, date_of_birth, Birthdate)` to extract the birthdate to the variable `Birthday`. This is only an example: different formats for different *VC contexts* can exist.

`ssi_diddoc` is the format for DID Documents. It contains names from the name-value of the DID Document. For example, if we want to access a corresponding public key, we specify `extract(DIDDoc, format, ssi_diddoc)`, `extract(DIDDoc, pk, PK)`. Similar to the `ssi_credential` format, for each context different a DID document format may exist, supporting different versions of the DID specification (such as `w3id.org/did/v1`).

3.3 New TPL Predicate

Built-in predicates wrap functionality that is executed by the ATV and not inside TPL, such as lookups and signature verifications. Built-in predicates may return their result in the variables that have been passed to the call (output parameters). For our extension to SSI, we introduce one additional built-in predicate:

`resolveDID(DID subject, min_blockage, out: DID document)` takes three arguments: The first argument is the DID to resolve. The second argument specifies the minimal age of the block. On call, the ATV will try to look up the DID document at the DL and, on success, return the result to the last argument's variable. If a ledger-based system (like in this paper) is used, the block age parameter can be seen as a choice of assurance level regarding the

DID document because a young block could be dropped as the blockchain grows, while an older block is more established and is unlikely to drop out. That means the higher number we choose, the more sure we will be that the block is not dropped.

3.4 Verification and Accountability

In [SM20] we investigated how we can formally verify aspects of trust decisions and make them accountable: for instance, in case a business transaction enters a legal dispute, it is valuable to have a precise argument available consisting of all data, especially involved credentials, and the policy rules upon which a decision was made.

Obviously, there are several problems that such a package cannot solve. First, the author of a trust policy may have made a mistake so that the trust policy does not truly reflect the business or organization's needs. Second, there may be problems in the implementation, like an overflow problem in a document parser. Third, especially when we look at an electronic document decades in the future, today's cryptographic algorithms like signature schemes may be broken. Fourth, when retrieving trust information from a server, it may be later impossible to prove that this server indeed gave a particular answer at the time of the policy decision. Here distributed ledger technology can give an advantage (and partially also with respect to the third problem) because it has interesting archival properties since as long as all but the few latest blocks are undisputed, we can reconstruct the state at every given point in time. Moreover, due to the distributed nature, we do not rely on a single party (that could, e.g., be hacked or out of business). However, with the distributed ledger technology come also different problems, most notably, a policy decision is made with respect to a block that has not yet "aged" sufficiently and is ultimately not included in what becomes the accepted chain of blocks. For this reason, we add the `min_blockage` parameter to the new `resolveDID` predicate to specify which block age is required for relying on an entry.

A problem we can solve concerns the correct implementation of trust policy's semantics in the decisions of the automated trust verifier ATV. Suppose, due to a bug, the ATV accepts a transaction that should be rejected according to the policy semantics. Building on the paper by Schlichtkrull; Mödersheim [SM20] results in a solution to this problem: the ATV records a *proof certificate*, i.e., a package consisting of a set of relevant facts about the transition, the state of the world at the given time, and the policy; then one can *check* if the acceptance indeed logically follows from the policy using an independent and verified software tool named RP_X introduced by Schlichtkrull et al. [SBT19; Sc20]. It is extremely unlikely that a logical mistake in the ATV decision process gets erroneously accepted by this "independent pair of eyes". In particular, we have no semantic gap, as the semantics of TPL rules can directly be formulated as first-order sentences, and thus in the native language of RP_X. Since this logical deduction is independent of the concrete technology (like formats and cryptography) that implements the facts, it works immediately with the extensions of TPL for SSI and DL.

4 Age Verification Example

We now give an example of the concept introduced in Sect. 3. To illustrate the concepts, we use a social online platform for teenagers. In this fictional platform, teenagers can join without revealing their legal identity, but they need to be in their teens (older than 12, not older than 18) to ensure only teenagers participate in the discussions. Since other identity attributes of the teenagers are not relevant, it is sufficient to provide an *age credential*. This age credential only contains the date of birth of the teenager and no further information.

The age credential is a VC containing the user's DID as subject and the date of birth encoded in the credential's `credentialSubject` field. To add (legal) value to the credential, a qualified issuer must issue it. A government authority or a similar trusted institution can be a legitimate issuer for such claims.

```
accept(Form) :-
  extract(Form, format, registrationFormat),

  extract(Form, birth_credential, Credential),
  extract(Credential, format, ssi_credential),
  extract(Credential, date_of_birth, Birthdate),
  calculateAge(Birthdate, Age),  Age >= 13, Age <= 18,

  extract(Credential, dIDsubject, DIDsubject),
  extract(Credential, dIDissuer, DIDissuer),

  get_DIDdoc(DIDsubject, PKu, DIDDocSubject),
  verify_signature(Form, PKu),
  get_DIDdoc(DIDissuer, PKi, DIDDocIssuer),
  verify_signature(Credential, PKi),

  check_issuer(DIDDocIssuer).

get_DIDdoc(DID, PK, DIDDoc) :-
  resolveDID(DID, 3, DIDDoc),
  extract(DIDDoc, format, ssi_diddoc),
  extract(DIDDoc, pk, PK), verify_signature(DIDDoc, PK).

check_issuer(DIDDocIssuer) :-
  extract(DIDDocIssuer, trustScheme, TrustSchemeClaim),
  trustscheme(TrustSchemeClaim, trustedTrustScheme),
  trustlist(TrustSchemeClaim, TrustListEntry),
  extract(TrustListEntry, pubKey, PKi),
  verify_signature(DIDDocIssuer, PKi).
```

List. 1: TPL policy sketch for our exemplary age verification use case

4.1 Example Policy

We show an example policy in List. 1 corresponding to the age verification use case (cf. Sect. 4). A user sends a transaction containing a registration request `registrationForm` to the discussion platform. The discussion platform uses the given policy and an automated verification tool to assess if the user is in the right age span to be admissible to the platform.

The input parameter passed to the TPL interpreter (`Form`) contains the registration request, signature, and credential of the user. After ensuring the incoming transaction has the correct format, it checks the user's age by extracting the date of birth from the birth credential. Then it uses the predicate `calculateAge` to derive the user's age before it verifies if the age is within the specified range. We omit a concrete explanation of the `calculateAge` predicate since it is of no interest to this discussion.

Next, the built-in predicate `extract` is used to retrieve the DID of the sender (credential subject) and of the credential's issuer. These two DIDs are then used with the new built-in predicate `resolveDID` (cf. Sect. 3.3) to retrieve the DID documents of the two entities. This step also takes the minimum age of the DID document into account, as specified by the second parameter. Each DID document contains a public key corresponding to DID, which is first used to verify the DID document itself. Further, the sender's key is used to verify the transaction, and the issuer's key to verify the credential. If all those checks succeed, there is a valid trust chain between the issuer and the registration request. The interpreter proceeds to authenticate the issuer itself. In our example, we authenticate the issuer in `check_SSI` using a common LIGHT^{est} authentication flow [Wa19] and a trust scheme `trustedTrustScheme`, showing that both centralized and decentralized world can be used together in one policy.

5 Future Work

5.1 Towards an implementation

We have shown that TPL's extensibility supports novel trust and identity management concepts like SSI. To give more insights into this adaption's consequences and explore further extensions or modifications to the TPL language, we propose implementing the stated concepts following an agile approach. Doing so supports the verification of the stated concepts while at the same time keeping up with the ongoing development of SSI concepts to ensure compatibility. Given that the first version of TPL is currently interlocked with the LIGHT^{est} toolchain, we propose a more stand-alone implementation, enabling other projects to use the TPL interpreter. Nevertheless, LIGHT^{est} provides powerful concepts which are essential to TPL. It is unavoidable and reasonable to keep building on these concepts. Finding the middle ground might be challenging.

5.2 Outlook

In this section we discuss further possibilities and future work in the intersection of distributed trust management and trust policies.

Issuer accreditation While authentication of a credential is performed by resolving and retrieving the signer's DID document and verifying the credential's signature, a DL is not always used to authenticate the signer or their respective issuer. On the one hand, if a consortium of qualified entities operates the ledger, the fact alone that a DID is registered on the DL provides legal value to the DID. The same is true for (identity) credentials registered on such a "qualified ledger" – as only qualified entities can add credentials to the DL, the registration alone acts as a certification of the credential's content. Thus, authentication of issuers can be achieved by supporting the respective discovery means using a DL.

On the other hand, if the used ledger is a public ledger such as the commonly used "Ethereum mainnet", other means are needed to define which entities represent qualified trust (service) providers or otherwise relevant entities. In the end, it depends on the service provider's local rules, laws, and other regulations which issuers are trusted by them. So it makes sense to anchor these rules in a trust policy. For instance, the service provider could define all qualified trust service providers as defined by their existing and trusted trust scheme to act as SSI (credential) issuers. In the example shown in Sect. 4, we sketch out how such a definition of a trusted scheme could be integrated into a TPL trust policy. To extend this (centralized) trust scheme-based authentication framework, we propose to support decentralized frameworks such as those realized by a smart contract-based web of trust [Mo21].

Privacy-preserving features In our example (cf. Sect. 4), even the date of birth provides more information than needed. The only relevant information is the 1-bit of information whether a person is in the defined age-range. Thus, more privacy could be added by supporting, e.g., range proofs, which we intend to do in a later version of TPL.

Acknowledgments

This work was supported by the European Union's Horizon 2020 Framework Programme for Research and Innovation under grant agreements No. 871473 (KRAKEN), No. 959072 (mGov4EU), and No. 830929 (CyberSec4Europe), as well as the Sapere-Aude project "Composec: Secure Composition of Distributed Systems", grant 4184-00334B of the Danish Council for Independent Research.

References

[Ab17] Abraham, A.: Whitepaper: Self-Sovereign Identity, tech. rep., 2017, visited on: 10/16/2020.

[Ab20] Abraham, A.; More, S.; Rabensteiner, C.; Hörandner, F.: Revocable and Offline-Verifiable Self-Sovereign Identities. In: TrustCom. IEEE, pp. 1020–1027, 2020.

[Ar05] Arends, R.; Austein, R.; Larson, M.; Massey, D.; Rose, S.: DNS Security Introduction and Requirements. RFC 4033/, pp. 1–21, 2005.

[Be17] Berghel, H.: Equifax and the Latest Round of Identity Theft Roulette. Computer 50/12, pp. 72–76, 2017.

[Be20] Berghel, H.: The Equifax Hack Revisited and Repurposed. Computer 53/5, pp. 85–90, 2020.

[BFG10] Becker, M. Y.; Fournet, C.; Gordon, A. D.: SecPAL: Design and semantics of a decentralized authorization language. J. Comput. Secur. 18/4, pp. 619–665, 2010.

[BL16] Bruegger, B. P.; Lipp, P.: LIGHT^{est} - A Lightweight Infrastructure for Global Heterogeneous Trust Management. In: Open Identity Summit. Vol. P-264. LNI, GI, pp. 15–26, 2016.

[DD10] Dong, C.; Dulay, N.: Shinren: Non-monotonic Trust Management for Distributed Systems. In: IFIPTM. Vol. 321. IFIP Advances in Information and Communication Technology, Springer, pp. 125–140, 2010.

[Fr20] Fritsch, L.: Identity Management as a target in cyberwar. In: Open Identity Summit. Vol. P-305. LNI, Gesellschaft für Informatik e.V., pp. 61–70, 2020.

[GN08] Gurevich, Y.; Neeman, I.: DKAL: Distributed-Knowledge Authorization Language. In: CSF. IEEE, pp. 149–162, 2008.

[He00] Herzberg, A.; Mass, Y.; Mihaeli, J.; Naor, D.; Ravid, Y.: Access Control Meets Public Key Infrastructure, Or: Assigning Roles to Strangers. In: IEEE S&P. IEEE, pp. 2–14, 2000.

[Ko20] Koa, C.-G.; Heng, S.-H.; Tan, S.-Y.; Chin, J.-J.: Review of Blockchain-Based Public Key Infrastructure. In: Cryptology and Information Security Conference 2020. P. 20, 2020.

[LFG99] Li, N.; Feigenbaum, J.; Grosof, B. N.: A Logic-based Knowledge Representation for Authorization with Delegation. In: CSFW. IEEE, pp. 162–174, 1999.

[Mö19] Mödersheim, S.; Schlichtkrull, A.; Wagner, G.; More, S.; Alber, L.: TPL: A Trust Policy Language. In: IFIPTM. Vol. 563. IFIP Advances in Information and Communication Technology, Springer, pp. 209–223, 2019.

[Mo21] More, S.; Grassberger, P.; Hörandner, F.; Abraham, A.; Klausner, L. D.: Trust Me If You Can: Trusted Transformation Between (JSON) Schemas to Support Global Authentication of Education Credentials. In: IFIP SEC. IFIP Advances in Information and Communication Technology, In press, Springer, 2021.

[Re21] Reed, D.; Sporny, M.; Longley, D.; Allen, C.; Grant, R.; Sabadello, M.: Decentralized Identifiers (DIDs) v1.0, W3C Working Draft, W3C, Jan. 20, 2021, URL: https://www.w3.org/TR/2021/WD-did-core-20210128/.

[Ro17] Roßnagel, H.: A Mechanism for Discovery and Verification of Trust Scheme Memberships: The Lightest Reference Architecture. In: Open Identity Summit. Vol. P-277. LNI, Gesellschaft für Informatik, Bonn, pp. 81–92, 2017.

[SBT19] Schlichtkrull, A.; Blanchette, J. C.; Traytel, D.: A verified prover based on ordered resolution. In: CPP. ACM, pp. 152–165, 2019.

[Sc20] Schlichtkrull, A.; Blanchette, J.; Traytel, D.; Waldmann, U.: Formalizing Bachmair and Ganzinger's Ordered Resolution Prover. J. Autom. Reason. 64/7, pp. 1169–1195, 2020.

[SLC19] Sporny, M.; Longley, D.; Chadwick, D.: Verifiable Credentials Data Model 1.0, W3C Recommendation, W3C, Nov. 19, 2019, URL: https://www.w3.org/TR/2019/REC-vc-data-model-20191119/.

[SM20] Schlichtkrull, A.; Mödersheim, S.: Accountable Trust Decisions: A Semantic Approach. In: Open Identity Summit. Vol. P-305. LNI, Gesellschaft für Informatik e.V., pp. 71–82, 2020.

[SS20] Sporny, M.; Steele, O.: DID Specification Registries, W3C Note, W3C, June 2020.

[Th17] Thomas, K.; Li, F.; Zand, A.; Barrett, J.; Ranieri, J.; Invernizzi, L.; Markov, Y.; Comanescu, O.; Eranti, V.; Moscicki, A.; Margolis, D.; Paxson, V.; Bursztein, E.: Data Breaches, Phishing, or Malware?: Understanding the Risks of Stolen Credentials. In: CCS. ACM, pp. 1421–1434, 2017.

[Wa19] Wagner, G.; Wagner, S.; More, S.; Hoffmann, M.: DNS-based Trust Scheme Publication and Discovery. In: Open Identity Summit. Vol. P-293. LNI, GI, pp. 49–58, 2019.

[Xi20] Xiao, Y.; Zhang, N.; Lou, W.; Hou, Y. T.: A Survey of Distributed Consensus Protocols for Blockchain Networks. IEEE Commun. Surv. Tutorials 22/2, pp. 1432–1465, 2020.

[Zh18] Zheng, Z.; Xie, S.; Dai, H.-N.; Chen, X.; Wang, H.: Blockchain challenges and opportunities: a survey. Int. J. Web Grid Serv. 14/4, pp. 352–375, 2018.

[ZZS14] Zwattendorfer, B.; Zefferer, T.; Stranacher, K.: An Overview of Cloud Identity Management-Models. In: WEBIST (1). SciTePress, pp. 82–92, 2014.