H. Roßnagel et al. (Eds.): Open Identity Summit 2019,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2019 107

GTPL: A Graphical Trust Policy Language

Sebastian Alexander Mödersheim[1] Bihang Ni[2]

Abstract: We present GTPL, a Graphical Trust Policy Language, as an easy-to-use interface for the Trust Policy Language TPL proposed by the LIGHTest project. GTPL uses a simple graphical representation where the central graphical metaphor is to consider the input like certificates or documents as *forms* and the policy author describes "what to look for" in these forms by putting constrains on the form's fields. GTPL closes the gap between languages on a logical-technical level such as TPL that require expertise to use, and interfaces like the LIGHTest Graphical-Layer that allow only for very basic patterns.

Keywords: Trust policy; graphical language

1 Introduction

Trust is a quite important element in identity management and electronic transactions: to be sure about the identity of a partner, one usually relies on some form of certificate; this in turn is only meaningful if one trusts the issuer of the certificate. There is hence a trust policy (at least implicitly) as to which entities one accepts as certificate authorities. The most basic form of a trust policy is simply a *list* of trusted entities. For instance, most web browsers ship with a list of certificate authorities (and their public keys) so all certificates issued by one of these authorities are immediately accepted.

While this may be sufficient for web browsers, for electronic business transactions one may want to formulate more complex policies, for instance where entities may have different trust levels. With increased complexity it becomes apparent that one wants a form of *language* to formulate such policies. Such a trust policy language is a formal language in the sense that it has to have a precise syntax (what constitutes a valid specification in the language?) and semantics (is the policy satisfied for a given problem instance?). Especially this semantic question, i.e., defining and implementing an automatic policy decision procedure, indicates a large similarity between trust policies and access control policies [Bl99; He00; Ya03]. Some access control policy languages like SecPal and DKAL [BFG10; GN08] are based on simple fragments of first-order or modal logics that both allow for using common logical concepts

[1] Technical University of Denmark, Department of Applied Mathematics and Computer Science, Richard Petersens Plads, Building 324, Room 180 2800 Kgs. Lyngby, Denmark, samo@dtu.dk
[2] Technical University of Denmark, Department of Applied Mathematics and Computer Science, Richard Petersens Plads, Building 324, Room 180 2800 Kgs. Lyngby, Denmark, bnia@dtu.dk

(like variables, connectives, or rule matching) and for rather immediate implementations of the decision procedures. Similarly, both the trust policy languages of [He00] and of the LIGHTest project [MS18] are based on a logic-programming approach. In fact, both [He00] and [MS18] call their trust policy language TPL.

While these languages are very declarative to use for logicians and programmers, they are not so suitable for users without a solid technical background. This is crucial since the trust policies are most relevant in business settings where the decision makers do not necessarily have such a background while at the same time they should be able to understand in full detail the policy they are authoring. For this reason, the LIGHTest project offers a simple graphical interface, aimed at novice users, where users can select entities that are trusted, and entities that are not [Th18]. The aim of the graphical trust policy language GTPL that we present in this paper is to fill the gap between on the one side very simplistic interfaces for policy languages that lack expressiveness and on the other side very technical languages that are very expressive (e.g. Turing-complete) but basically require programming skills. The target user of GTPL has a great knowledge of their business domain, but not necessarily a technical-logical background.

The central graphical metaphor of GTPL is that of a *paper form*. For instance, when applying for admission to a university, one needs to fill out a form provided by the university that has different *fields* with a predefined meaning, e.g. name, address, date of birth, and one has to attach to the form a number of documents such as a high-school diploma. An office clerk who processes the application will follow a policy for checking the documents, e.g. whether the attached diplomas indeed qualify the applicant for this study line, whether the grades are good enough, and whether the information such as name and address in the different documents indeed matches. One could thus describe this checking process as *constraints* on the fields of the forms, e.g., that certain fields match each other and that certain values are in acceptable range. Hence, one could specify a policy quite formally by putting these constraints directly into an empty form, essentially specifying *what to look for*. Such a policy is not only easy to write, it is also possible to read very quickly, as it literally gives the "overview" over what matters. Moreover, in contrast to a textual representation, it is less likely that the policy author accidentally forgets to specify a constraint on some field, since the entire form is in view.

The contributions of this paper include the definition of GTPL as a graphical language for trust policies that consists of a small number of language constructs. The language is parameterized over the concept of a form as a list of fields, and can thus be used with forms from any business domain. Moreover, we have implemented GTPL as a graphical policy editor that includes a translator to the LIGHTest TPL. This makes GTPL a formal language with a precise *semantics* (through the semantics of TPL) and it immediately makes the policies usable in LIGHTest and its automated trust verifier [BL16]. While GTPL is closely related to LIGHTest, the idea of specifying policies by constraints on fields of forms is general: We see this as a contribution towards language design that abstracts from irrelevant technical details and allows users to focuses on the business logic.

We introduce GTPL in the following by a concrete example of trust policies for an auction house that wants to allow online bids. This allows us to introduce all constructs of GTPL step by step as a collection of policy rules, where each rule describes one sufficient condition for the auction house to accept an online bid. We then summarize the general concepts of GTPL in a textual syntax. This syntax both describes the data structures of the GTPL editor and is the basis for the semantics by translation to LIGHTest TPL. Due to lack of space, we only summarize this translation, the formal details are found in a technical report [MS18].

2 A Running Example

We introduce the Graphical Trust Policy Language GTPL by using an example of a classical auction house who likes to extend their traditional business to electronic bidding and formulate trust policies for that matter. This should be based on a trust infrastructure like LIGHTest [BL16], and we will introduce the relevant concepts of LIGHTest along the way.

The auctions may easily range up to thousands of Euros for a single item, which gives of course the classical problem of ensuring that the successful bidder indeed pays the sum they have bid. On the one hand, the auction house does not want to put any entrance barrier for new customers who just "stumbled" upon an item by an Internet search, on the another hand they want to avoid that, for instance, somebody practically anonymously bids on an item just to get the price up and then not paying if that bid was the highest.

This is a classical *trust* problem. The classical (non-electronic) solutions are that customers have to bring references from other auction houses or a bank statement, or be present at the auction in person, proving their identity before the auction starts. The point of trust infrastructures like LIGHTest is to facilitate these aspects in the digital world so one can benefit from the large potential of digitalization without losing the security and trust guarantees of the classical non-digital world. This example allows us to illustrate GTPL with realistic policies that an auction house may want to choose.

2.1 Bidding Forms

Auction houses typically allow customers to bid via standard (non-digital) mail if they cannot be physically present at the auction house. The bidder would tell the auction house a maximum bid for a particular item, and the auction house could accordingly act as if the bidder was present at the auction and place bids for the customer up to the maximum bid. For this purpose, each auction house would have their own bidding form: a paper sheet bearing the name of the auction house and the particular auction, like "The Auction House 2018". The form contains fields to fill in, such as the personal information of the bidder and a list of items (the lot numbers and the maximum bid) and finally a field where the bidder must sign the form. This signed form is then mailed to the auction house.

```
<AUCTIONHOUSEFORMAT
 auctionID="AUCTION18">
<bidder>...</bidder>
<address>
  <street>...</street>
  <city>...</city>
  <country>...</country>
</address>
<bid lotno="..."
     amount="...">
<signature>
jmj7l5rSw0yVbvlWAYkKYBwk
</signature>
<Certificate>
...
</Certificate>
</AUCTIONHOUSEFORMAT>
```

Fig. 1: Example form in XML

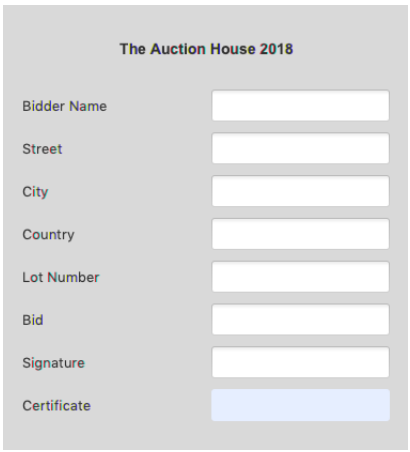

Fig. 2: Graphical representation of the form in GTPL

The first step of digitization for auction houses was providing online auction catalogs, where customers can click on items and place a bid. This would basically lead to an electronic version of the classical paper bidding form, and it is sent to the auction house using https or simply email. Such an electronic bidding form could look like the XML snipped in Fig. 1—for simplicity we consider bidding only on a single item. We have here already included a field for the digital signature, which is actually still optional in many of today's online bidding solutions. If used, it would be a digital signature on a hash of the document.

We consider the XML-based form in Fig. 1 as *concrete syntax*, since there are many different ways to convey the same informations, but the essence, the *abstract syntax*, is that it is a list of attribute-value pairs, as shown in Fig. 2, where every value is a blank box to be filled, and every attribute is an annotation like "Bidder Name" that declares the meaning of the corresponding value. As a first approximation let us say that the content of the blank boxes will be a string. Moreover, the form carries a title, identifying the meaning of the form as an entirety, so that nobody accidentally considers this form, say, as a passport. This title corresponds in the non-electronic world to having the name of the auction house and the particular auction printed on the paper bidding form. (This is crucial also to prevent a hacker to *replay* a bidding form at the next auction.) Both the signature and the certificate fields are special fields, while the other fields are generic and carry no built-in meaning for GTPL.

It is thus possible to connect a variety of forms to GTPL: one simply needs to define a parser for the new form (and connection to signature verification for the used signature scheme), and add it to the library of GTPL *formats*. This library of formats shall satisfy the conditions of [MK14], namely being unambiguous (every concrete input string can be parsed in only one way) and pairwise disjoint (no string can be parsed for two different formats).

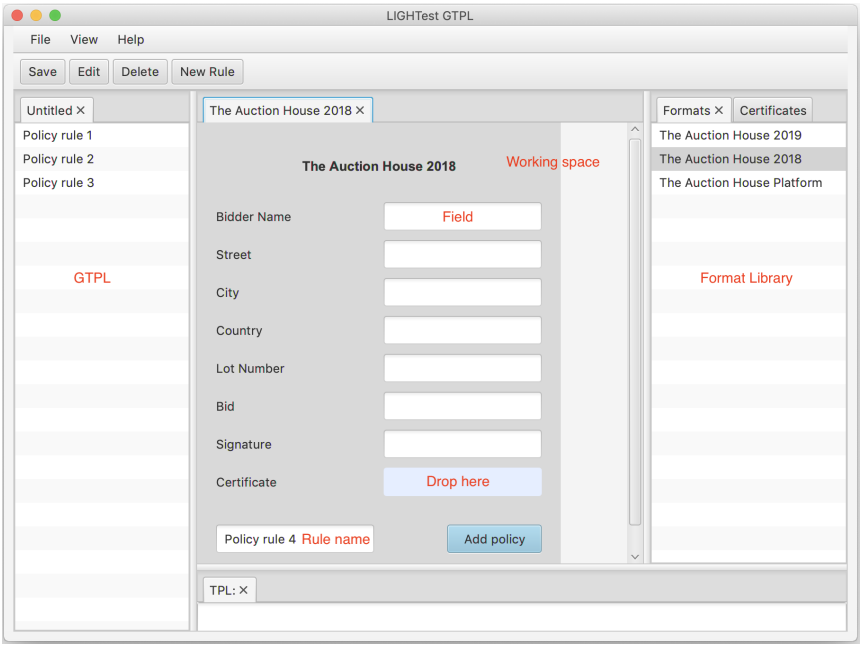

Fig. 3: The layout of the GTPL application (the annotations in red are not part of the GUI).

3 The Layout of the GTPL Application

Fig. 3 gives the overview of the GTPL application. The left part (labeled "GTPL") is the list of policy rules defined so far, the middle part (labeled "Working space") is the policy rule currently under edit, and the right part (labeled "Format Library") is a palette of formats and certificates that are currently available. The normal workflow to create a new policy rule is to select a format or certificate from the library and drag it to the center workspace, to open a new blank form. The figure shows this for the format "The Auction House 2018". One can now make constraints on this form, give it a name (in the "Rule name" field) and then click the "Add policy" button. Then it will appear in the list of the rules on the left; from there it can be selected later for editing (or deleting). Finally one can store the rules in GTPL format or export them to LIGHTest TPL for use in the LIGHTest architecture.

3.1 The Hello World of GTPL

The basic idea is that we can use this simple graphical representation of the "empty" form as a basis for describing trust policies. For instance, let us define as a first example policy rule in Fig. 4 where the auction house wants to accept any bids up to 100 Euro; note that the currency is implicit in the format (it may explicitly write that in the syntax of the field). The

Fig. 4: A first graphical policy rule

policy is specified by entering into the bid field the text `<=100` and leaving blank all other fields. This means that this policy rule has only one requirement, that the bid is a number up tp 100. In particular we do not require a signature, i.e., for bids below 100 Euro, this auction house is not worried about trust.

The basic concept of filling a constraint into a field is to either demand a particular value, e.g. entering `Denmark` for the country field, or a comparison with a value like `<=100`. The latter requires that the corresponding field of that format is defined to have an ordinal *type*. Such ordinal types can also be defined as part of the library, e.g., we could have a type *rating* with ordered elements `standard` < `gold` < `premium`. Moreover, we allow that a field could be any value in a list, e.g., the policy author can specify a list of countries `CL` and constrain the country field of the form to be any value of the list by writing `in CL`.

3.2 Checking Signatures

As a second policy rule, the auction house accepts any bid up to 1500 Euro if it is signed by an eIDAS qualified signature [En18]. To that end, we use that the Signature field has a distinguished meaning: we specify in this field the public key with respect to which the signature must verify. It is part of the format definition which part of the document is signed. The graphical convention is that the signature includes *all fields above the signature field*, e.g., in the auction house form the signature comprises all fields except the certificate field.[3]

In a signature field we practically never want to specify a particular fixed public key, but specify that it relates to a given certificate. If we use again the metaphor of a paper form, then a certificate would be an *attachment* to the main form, i.e., the person submitting the form provides additional relevant information that itself has some structure. Thus, such

[3] This should also include the kind of form it is, i.e., the format name; this is implied by the standard requirements on the disjointness of formats [MK14].

Fig. 5: A graphical policy rule with eIDAS qualified signature

attachments can be regarded as forms themselves, e.g., a certificate may be in the X.509-format. Further, we want to "bind" attachments to the main document in a suitable way; one could have for this a special container format (like Associated Signature Containers [In16]) or simply directly have another field in the form for such attachments, like the "Certificate" field in our example. The value of such an attachment field must then also be a format.

To embed this concept into our graphical language, we have adapted the notion of a *sub-form*, i.e., a field in a form can host an entire form itself. Fields of this type are highlighted blue in the GTPL application, indicating that the policy author can drag a form from the library onto this blue field. Fig. 5 shows the result of dragging a certificate format for eIDAS certificates to the certificate field, inserting a blank subform for entering constraints. Observe that in Fig. 5 we have now specified the *variable* `PK` both on the signature of the main form and in the `pubKey` field of the certificate. This means that the signature of the main form must verify with the public key we can extract from the eIDAS certificate. The eIDAS certificate is itself signed with yet another key `PkIss`—this is also a variable.[4] This illustrates the fact that a certificate is itself a signed document and without verifying the signature of the certificate it does not mean much. In standard PKIs one may have an arbitrary long sequence

[4] Recall that the scope of the signature field of the auction house format spans all fields above it; similarly, the scope of the signature field of the certificate are all fields of the subform above the signature field (here, all fields).

of certificates until one reaches the certificate from an already trusted organization. Here, instead, we want to formalize that the certificate is part of a particular trust scheme, eIDAS in this in this example, i.e., the issuer is member of a particular trust list headed by the EU.

There are several possible ways to organize and implement the check of this trust membership claim; e.g., LIGHTest suggests to include a pointer to the particular entry in the trust list, so one does not need to download the entire trust list. Essential to the policy author are only two aspects. First, one identifies the desired trust scheme, here `[eIDAS_qualified]`. We require that such trust lists are defined as part of the library of formats and certificates, in particular which URL is the relevant authority for a particular trust scheme. Second, if the lookup of the trust list entry is successful (otherwise the policy is not satisfied), one may specify constraints on the trust list entry that may contain a number attributes like a trust level. This entry is depicted graphically in GTPL to the right of the trust list—again as a form. In the example we assume that the entry contains a public key and specify that it has to be the same variable `PkIss` that we also have in the field of the eIDAS certificate's signature. This means, the eIDAS certificate must verify against the public key from the trust list entry, i.e., the certificate was indeed issued by a member of the eIDAS qualified trust scheme.

This completely explicit handling of trust list entries allows to specify quite complex policies when the trust list entry contains more information, while for simple Boolean trust lists (i.e., just checking that the entity is on the trust list like in this example), this is a bit overkill. Therefore we plan to allow here also a simplified notation as syntactic sugar, namely one could just specify `[eIDAs_qualified]` in the `Certificate` field, i.e., keeping the subform of the eIDAS certificate implicit.

3.3 Allowing Trust Translation

LIGHTest facilitates also the specification of trust translations, e.g. the authority of a trust scheme can specify that they regard another trust scheme as equivalent, for instance the European Union may declare that they regard some foreign trust scheme as equivalent to eIDAS. It is of course the decision of each policy author whether they want to accept trust translation in the first place. For our auction house example, we could imagine the following policy: we do accept certificates with foreign trust schemes that eIDAS considers equivalent, but set a lower limit on the bid in this case. Fig. 6 shows just that: we have replaced `[eIDAS_qualified]` with `=[eIDAS_qualified]`, meaning we do allow eIDAS, or one that eIDAS considers equivalent, and thereby allowed trust translation, but we have capped the bid to `<=1000` in this case. Also in this case the certificate is not an eIDAS certificate, but a generic certificate.

Fig. 6: A graphical policy rule with eIDAS equivalent signature

3.4 Putting it all together

We have specified a number of policy rules. They are collected all in the left tab of the GTPL application (cf. Fig. 3). All these graphical trust policy rules are put together by *disjunction*, i.e., in our example, a bid is accepted if *any* of the rules match. The order of the rules in the left-hand tab of the interface only determines in which order they are checked, so it makes sense to put most common cases first, and the rarer cases later.

4 GTPL Syntax and Semantics

While GTPL is a graphical language, it is also formal in the sense that is has a precise syntax and semantics. This is crucial for trust decisions and thus for all machinery that works on GTPL specifications. The abstract syntax of GTPL is defined in terms of Java data structures; for reading convenience, we use an EBNF-style notation in Fig. 7. Let us briefly review each item with an intuitive semantics. The formal semantics is defined by translation to the LIGHTest TPL; the formal definitions of TPL and the translation are found in [MS18].

At the top level, GTPL is a list of forms, where each form means one rule of the policy, like in figures 4–6. The meaning of the full policy is the *disjunction* of the rules, i.e., the

$$
\begin{array}{rcl}
\mathit{GTPL} & ::= & \mathit{Form}^{\star} \\
\mathit{Form} & ::= & \mathit{Formatname}(\mathit{AttVal}^{\star}) \\
\mathit{AttVal} & ::= & (\mathit{Attributename}, \mathit{Value}) \\
\mathit{Value} & ::= & \mathit{BLANK} \mid \mathit{Constant} \mid \mathit{Variable} \mid \mathit{op}\ \mathit{Constant} \mid \mathit{op}\ \mathit{Variable} \\
 & \mid & \mathit{in}\ \mathit{Listname} \mid \mathit{Form} \mid =^{?}\ [\mathit{Trustlist}]\ \mathit{Form}^{?} \\
\mathit{op} & ::= & < \mid > \mid <= \mid >=
\end{array}
$$

where *Formatname*, *Attributename*, *Variable*, *Trustlist* and *Listname* are alphanumeric identifiers and *Constant* is either a sequence of digits or printable ASCII characters in quotes

Fig. 7: Syntax in GTPL in a textual/data structure form; terminal symbols are set in blue.

policy is fulfilled, if at least one rule is. A form consists of a formatname (like "Auction House 2018") and a list of attribute-value pairs. The meaning of this policy is that firstly the given input must be parsable as the given format indicated by formatname, and secondly the conjunction of the constraints specified by the attribute-value pairs must be satisfied.

An attribute-value pair consists of an attribute name and value, of course. Here, the attribute name (like "Country") indicates one of the fields of the form, and the value gives a constraint on the value of this field. The first possibility is *BLANK* meaning that the policy author left the field blank, and thus there is no constraint on this field. Second, it can be a constant (either numeric or an ASCII string in quotes), meaning the value must be just that. Third, it can be a variable (like `PK` in the examples). The meaning of a variable is that the value can be arbitrary, but all fields where the same variable is specified (in the present rule) must have the same value. The fourth and fifth possibilities are a comparison operator followed by a concrete value or a variable. This is can only be used on fields where an ordering is defined (e.g. numerics, levels, dates). The sixth possibility is to specify membership in a user-defined list (e.g. the country must be one in a given list of countries). The seventh possibility is a form itself. This can be only used on fields that are highlighted blue, i.e., that allow for a subform as a value, e.g. the certificate field in the examples. The meaning is simply that in this case the condition specified for the subform are checked as expected.

The last possibility of a value has several options, and can only be used for the `trustlist` field of certificates. The most basic form is to specify only a trustlist (like `[eIDAS_qualified]`). The meaning is that this trustlist field is a URL that points to the entry of a trust list. The constraints we specify here are (a) that trust list of the URL indeed belongs to the specified trust list (like eIDAS), (b) that the trust list entry indeed exists and (c) that it contains a public key that verifies the signature of the given certificate. One option for this trust list specification is to specify also a form (the fact that this is optional is specified by the question mark). If specified, the meaning is that the returned trust list entry must meet the constraints expressed by the given form. This allows for trust schemes where the trust list entry contains further entries, e.g., a trust level that can then be constrained as part of the policy. Finally, one can also put an equal sign in front of the trust list specification and thereby allow trust translation. (See [MS18] for more technical details of trust translation.)

5 Conclusion

We have introduced a graphical trust policy language to describe trust schemes. The central metaphor of this graphical language is to treat all input documents like paper forms that consist of a number of fields and the policy author can take a blank form and write constraints onto the fields. We believe this is a quite intuitive way of specifying it, because the policy authors have a good knowledge of the business domain they work in, e.g., the owner of an auction house understands the bidding form of the auction house and a university clerk understands the application form of the university. It is then easy to say what the requirements are based on the fields of the form, and seeing all the fields together also minimizes the risk of forgetting something: sweeping with one's eye over the field of the form, one typically remembers what conditions must be checked about this field. One may compare this with instructing a new employee: telling them literally "what to look for" in order to make the decision to accept or to deny.

In fact, GTPL has started with the question how LIGHTest experts would like to specify policies, i.e., to extract the essential logical elements of the more technical TPL specifications in a succinct form. This is thus close to typical mathematical efforts to abstract, generalize and thereby simplify matters. We see in this the key contribution of this paper, to identify a very simple but expressive set of concepts to specify policies with. We believe, however, that this language can be further improved and developed, especially with the help of systematic user testing and participation.

One of the most closely related graphical policy languages is a graphical editor for XACML [NUG15] that is based on the Scratch approach for teaching programming to children [Re09]. One of the most interesting ideas of Scratch is the graphical metaphor of puzzle pieces, so that constructs can only be combined in meaningful ways. Since with the forms we already have the overall structure of a policy rule, this is was not directly necessary for GTLP, but indeed this metaphor could be helpful in future versions, namely for types of credentials and forms as well as for specifying conditions (which is still textual at present). Indeed within the LIGHTest project, there is also work in progress to define a natural-language layer for policy specifications [Th18], also based on Scratch, to be close to natural language. This implies giving the user less boundaries, but also less structure. It is certainly interesting to see if these two languages could benefit from each others ideas. For future work we also intend to look at the specification of trust translation schemes themselves, as well as trust with delegation schemes.

Acknowledgement This work was supported by the EU H2020 project no. 700321 "LIGHTest: Lightweight Infrastructure for Global Heterogeneous Trust management in support of an open Ecosystem of Trust schemes" (lightest.eu).

Bibliography

[BFG10] Becker, M.; Fournet, C.; Gordon, A.: SecPAL: Design and Semantics of a Decentralized Authorization Language. Journal of Computer Security 18/4, pp. 619–665, 2010.

[BL16] Bruegger, B. P.; Lipp, P.: LIGHTest–A Lightweight Infrastructure for Global Heterogeneous Trust Management. In: Open Identity Summit 2016. 2016.

[Bl99] Blaze, M.; Feigenbaum, J.; Ioannidis, J.; Keromytis, A. D.: The KeyNote Trust-Management System Version 2, IEEE RFC 2704, 1999.

[En18] Engelbertz, N.; Erinola, N.; Herring, D.; Somorovsky, J.; Mladenov, V.; Schwenk, J.: Security Analysis of eIDAS - The Cross-Country Authentication Scheme in Europe. In: 12th USENIX Workshop on Offensive Technologies, WOOT 2018. 2018.

[GN08] Gurevich, Y.; Neeman, I.: DKAL: Distributed-Knowledge Authorization Language. In: Proceedings of the 21st IEEE Computer Security Foundations Symposium, CSF 2008. 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: 2000 IEEE Symposium on Security and Privacy. Pp. 2–14, 2000.

[In16] Institute, E. T. S.: Electronic Signatures and Infrastructures (ESI); Associated Signature Containers (ASiC), tech. rep. ETSI EN 319 162-1 V1.1.1, 2016.

[MK14] Mödersheim, S.; Katsoris, G.: A Sound Abstraction of the Parsing Problem. In: IEEE 27th Computer Security Foundations Symposium, CSF 2014. Pp. 259–273, 2014.

[MS18] Mödersheim, S.; Schlichtkrull, A.: The LIGHTest Foundation, tech. rep. DTU TR-2018-6, Available at `http://orbit.dtu.dk/ws/files/160744642/tr18_06_Modersheim_A.pdf`, 2018.

[NUG15] Nergaard, H.; Ulltveit-Moe, N.; Gjøsæter, T.: A Scratch-based Graphical Policy Editor for XACML. In: Information Systems Security and Privacy, ESEO. Pp. 182–190, 2015.

[Re09] Resnick, M.; Maloney, J.; Monroy-Hernández, A.; Rusk, N.; Eastmond, E.; Brennan, K.; Millner, A.; Rosenbaum, E.; Silver, J. S.; Silverman, B.; Kafai, Y. B.: Scratch: programming for all. Commun. ACM 52/11, pp. 60–67, 2009.

[Th18] The LIGHTest project: Deliverable D6.2: Requirements and Design of a Conceptual Framework for Trust Policies, Available at `https://www.lightest.eu/static/deliverables/D6.2.pdf`, 2018.

[Ya03] Yao, W.: Fidelis: A Policy-Driven Trust Management Framework. In: First International Conference on Trust Management, iTrust 2003. Pp. 301–317, 2003.