Exploring and Conceptualising Attestation*

Ian Oliver[1][0000−0002−8319−2612], John Howse[2][0000−0002−2329−2726], Gem Stapleton[2][0000−0002−6567−6752], Zohreh Shams[3][0000−0002−0143−798X], and Mateja Jamnik[3][0000−0003−2772−2532]

[1] Nokia, Helsinki, Finland
ian.oliver@nokia-bell-labs.com
[2] Centre for Secure, Intelligent and Usable Systems, University of Brighton, UK
{John.Howse,g.e.stapleton}@brighton.ac.uk
[3] University of Cambridge, Cambridge, UK
zs315@cam.ac.uk Mateja.Jamnik@cl.cam.ac.uk

Abstract. When formalising the rules of trust in the remote attestation of TPM-based computer systems it is paramount that the rules are precisely understood, supporting unambiguous communication of information about system requirements between engineers. We present a diagrammatic approach to modelling rules of trust using an extended version of concept diagrams. Within the context of our proof-of-concept Network Function Virtualisation and Attestation environment, these rules allow different level of trust to be explored and, importantly, allow us to identify when a computer system should not be trusted. To ensure that the modelling approach can be applied to general systems, we include generic patterns for extending our domain model and rules of trust. Consequently, through the use of a formal, yet accessible, diagrammatic notation, domain experts can define rules of trust for their systems.

Keywords: attestation · trust · networks · specification · diagrams.

1 Introduction

A major problem in the security of critical telecommunications infrastructure is knowing when it is appropriate to trust an element: a server machine, an edge node (e.g., base station) and more commonly IoT and UE (user equipment, e.g., mobile phones) devices. This paper presents a formal, diagrammatic approach to modelling the data structure related to trust in systems, allowing rules to be defined that capture system integrity. In particular, the rules, based upon the Trusted Computing Group's[4] Trusted Processor Module (TPM) Technology, allow machines to be remotely attested so establishing whether a given system is trusted [17]. Attestation is a method for authenticating trustworthiness of your system to a remote party. Of course, the idea of using models, and in particular

* Stapleton, Shams, and Jamnik were funded by a Leverhulme Trust Research Project Grant (RPG- 2016-082) for the project entitled Accessible Reasoning with Diagrams. Oliver was supported by the EU-funded SCOTT and Secredas projects.

[4] https://trustedcomputinggroup.org.

diagrammatic representations of those models, is not new, UML being a canonical example. But, the use of diagrammatic models to represent constraints of the complexity that arise in the attestation of machines in networks is not common, with B [1] or Z [14] sometimes being notations of choice. The reason for the lack of diagrammatic models arises due to expressiveness limitations that are considered inherent in diagrammatic languages [4]. However, recent research has pushed the boundaries of what can be expressed diagrammatically [8] in ways effective for humans to understand [2, 7, 10–12].

In this paper we present a diagrammatic method for formally defining the data structure over which rules are used to test the integrity of systems, using trusted computing principles. Our focus is the particular application area of the remote attestation of telecommunications cloud infrastructures known as Network Function Virtualisation (NFV) [5, 9], as has been defined by the European Telecommunications Standards Institute (ETSI). The contributions made in this paper are as follows:

1. We diagrammatically define specific rules of trust, considering the system to be static, allowing the integrity of systems to be established at the level of individual rules. We present patterns so that rules for additional kinds of data associated with machines can be included in the attestation framework.
2. We explore how sets of rules of trust can be considered in combination to define different levels of trust, allowing us to attest elements. The attestation rules are partially ordered, thus network routing provision can select a more trusted environment for running sensitive and critical workloads.
3. We argue, by appealing to known results, that our diagrammatic approach to modelling is more effective than competing textual and symbolic methods. This suggests that people are likely to produce more understandable and, thus, fit-for-purpose, rules of trust. This, in turn, will lead to more robust network provision.

Our diagrammatic approach to modelling the rules of trust can also be considered desirable as compared to existing approaches, such as database access, which hide the actual functioning of the rules themselves. Given a formal specification of the rules, such as those we propose in this paper, an implementation of them in a network can be provided. We conclude the paper by discussing how one might extend this approach to define trust in the temporal case, where information known by machines changes over time.

2 Context: Trusted Computing in NFV

Based around a hardware root of trust, trusted computing provides a mechanism for measuring, cryptographically, critical system components which include BIOS, Operating System, critical system files and so forth. These measurements can be provided, in a secure manner, to an external remote attestation server. Once the measurements are provided, they can be used to determine whether the machine that provided the measurements can, for example, be trusted for use or workload placement.

This process – of remotely providing measurements and checking them against expected values – is used as a security mechanism to ensure that systems providing telecommunication services remain in a known (trusted) state. When measurements do not match, changes can be monitored and acted upon. The integrity of systems, using this approach, can be provided by utilising the Trusted Platform Module (TPM) and remote attestation in a novel manner by reasoning over the latest data as well as historical records in the context of system events such as restarting a machine or patching. Identifying system integrity in telecommunication systems is clearly important in both the static and temporal cases. The primary focus of this paper is on the static case, with the temporal case left as future work.

2.1 Network Function Virtualisation

Telecommunications infrastructure is increasingly being virtualised – this is known as Network Function Virtualisation (NFV). Such an environment consists of a set of elements – servers, IoT, Edge, UE elements. Server machines are generally known as the NFV Infrastructure (NFVI) and run the core telecommunications workloads. This workload is in the form of Virtualised Network Functions (VNFs) which communicate amongst themselves and with the Edge elements. Ensuring the integrity by knowing what exactly is running in terms of whether system components are exactly as expected (i.e., non-tampered, no rootkits, known BIOS/firmware/kernel/VNF configurations, etc.) is critical to the security of such a system.

We are constructing an attestation environment that encompasses the NFVI, VNFs and ultimately Edge, IoT and UE elements. This environment monitors, attests and stores known good configurations and acts as arbiter to other infrastructure components in questions of whether elements or structures of elements are trusted. Failure of trust can have significant consequences for overall network integrity, workload placement, and data integrity, for example.

Traditional attestation tends to focus on whether an element is trusted or not against a single policy. NFV is a much more complex and dynamic environment where the trust decision can vary depending upon circumstances, for example, not all elements have the same notion of trust. In this respect we break the monolithic trust decision into a larger number of simpler rules which can be combined with varying policies. This allows finer grained analysis and trust history to be combined to provide a more context-specific answer to whether a given element is trusted.

2.2 Integrity Measurements and Attestation

An important question arises from the prior discussions: how do we know whether the measurements we obtain from elements are correct and, thus, the element is trustworthy. We can make an *integrity measurement* by requesting a *quote* from a given (element of) a machine for a given kind of measure. As one example, we might ask a machine to quote its core root of trust and BIOS measurement as well as its operating system kernel configuration. In this case, a TPM device will send a reply comprising a data structure with these measurements alongside a signature and some meta-data which is based on the TPM's attestation key; this key is unique to each TPM device.

We now include an example to illustrate modelling trust in networks. Suppose that we have a quote for a machine which includes its CRTM, BIOS configuration and DRTM measurement as stored in the TPM's SHA-256 PCR bank. Additionally, we can supply the handle in the TPM of a signing key, such as the attestation key, as well as a hashing algorithm for the quote. Thus, the quote itself is returned and the quote taken against the attestation private key, which (using significantly truncated values) may look something like:

```
tpm2 quote -k 0x81010003 -L sha256:0,1,2,3,17
quoted: ff544347801...eca71a signature: 1400...6334
```

Importantly, the quoted value is constructed from a number of data items: the hash of the requested PCR, the attested value, the TPM's clock, the firmware version, the reset and restart counts (which correspond to the number of boot and suspend events associated with the machine), as well as clock integrity measures and a TCG-defined type and a magic number. These quoted values must be compared to the expected values. The attested value itself *must* match the given known expected value. By comparing the other items of meta-data along with signature verification, using the attestation public key, we are able to reach a decision on the machine's integrity status. If expected values do not match actual values the integrity of the machine is uncertain and it should not be trusted.

With TPM a number of concepts of trust are defined (at least on x86 platforms): CRTM – the core root of trust measurement for a system, the SRTM – a measurement over BIOS and initial boot loader configurations (with other aspects as necessary), DRTM – a measurement relying upon certain sandboxing features of x86 processors to measure operating system components prior to launch. We can continue to build policies over measurements to be taken over the file system at run-time (e.g., Linux IMA and EMA). Furthermore the TPM specification is open to further measures such as geographical trust.

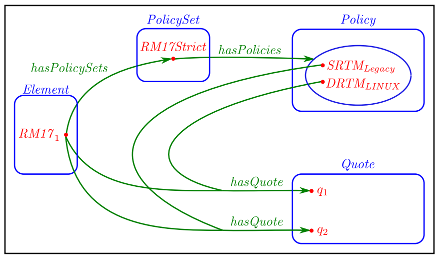

Fig. 1. An example situation depicted using an extended concept diagram.

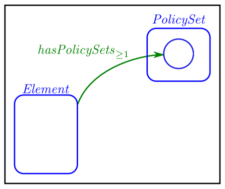

Fig. 2. Elements and their policy sets.

3 Modelling the Data Structure

We will now propose a formal model of the data structure discussed in the previous section. To reiterate, the decision to trust a given system depends on the latest quote for a given NFVI element (machine) matching certain properties which are expressed as a policy. In the data structure, there are four main classes: *Element*, *Quote*, *Policy*, and *PolicySet*. Elements and quotes have been defined earlier. A policy, however, is a combination of measures. For example, a policy might be for a given CRTM, partial SRTM measures, a DRTM measure, or user defined measures, say. A policy set groups these policies into logical sets which enable ease of measurement management as machines/elements are updated or need to comply with varying situations. One novelty in our approach is moving away from static, monolithic measurement to finer grained policies; for example, it is perfectly possible to have two machines with the same CRTM/SRTM measures and different DRTM; an example is two different Linux kernels on the same hardware and the same configuration. We want to capture the following information:

1. Every element is associated with at least one policy set.
2. A policy set is (essentially) a non-empty set of policies.
3. Given an element and one of its associated policies, there is exactly one quote. Thus, given an element and policy pair, there is either no quote (when the policy is not associated with the element) or exactly one quote (when the policy is associated with the element).

We assume that Element, Policy, PolicySet and Quote are pairwise disjoint.

Our approach to defining this data structure is to use an extended version of a diagrammatic logic, called *concept diagrams* [8], which has a formal syntax and semantics and they form a second-order dyadic logic [15]. Concept diagrams have been empirically evaluated against both a stylised textual notation (Manchester OWL Syntax) and a symbolic notation (description logic [3]), with the results suggesting that people can interpret concept diagrams significantly more effectively (measured via accuracy and time performance) [2]. In addition, they form a richly expressive diagrammatic logic, capable of expressing a wide range of constraints including, with some minor extensions to allow the expression of ternary relations, the rules that are needed to identify whether we can trust elements [16].

An example of what this data structure looks like can be seen in the extended concept diagram in figure 1. The four classes are represented by four rectangles. We can see that $RM17_1$ is an *Element*. In addition, $RM17_1$ is associated with one policy set called *RM17Strict*. In turn, we can see that this policy set includes two policies, $SRTM_{Legacy}$ and $DRTM_{Linux}$. We can see that $RM17_1$ along with the policy $SRTM_{Legacy}$ returns the quote q_2 and, with the policy $DRTM_{Linux}$ returns the quote q_1. Note that concept diagrams as in [8] do not include arrows with two sources. Here, these are used to represent ternary relations.

In figure 2, the classes *Element* and *PolicySet* are (again) represented by non-overlapping rectangles; their non-overlapping nature asserts that no two

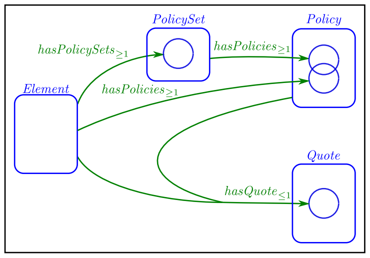

Fig. 3. The basic structure.

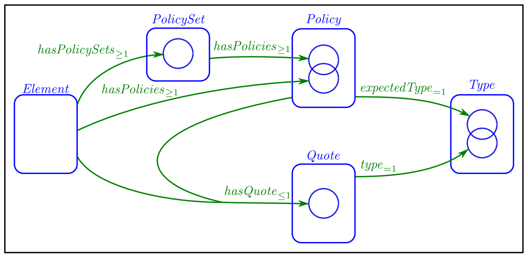

Fig. 4. Adding in information about types.

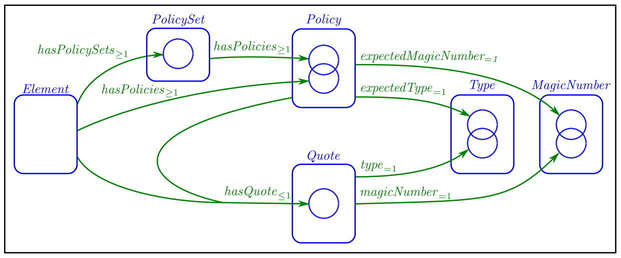

Fig. 5. Adding in information about magic numbers.

things can be in both classes – they are disjoint. Now, the fact that elements are associated with at least one policy set is expressed by the arrow. Here, the arrow, labelled *hasPolicySets*, shows that there is an association (binary relation) between *Element* and *PolicySet*. The annotation $_{\geq 1}$ tells us that every element is associated, under the *hasPolicySet* relation, to at least one policy set.

Figure 2 is extended in figure 3 to express further constraints. For instance, the arrow from *PolicySet* to *Policy* also has the annotation $_{\geq 1}$, which informs us that policy sets are associated with at least one policy under the *hasPolicies* relation. It is helpful to define a further relation, that allows us to navigate directly from elements to policies, indicated here by the *hasPolicies* arrow sourced on *Element* and targeting *Policy*. There is also an arrow, labelled *hasQuote*, which is sourced on *Element* and *Policy* and targets an anonymous subset of *Quote*. This expresses that given an element and a policy, we 'return' a set of quotes. The arrow annotation, $_{\leq 1}$, tells us that the returned set of quotes contains at most one quote. There is a required constraint that this diagram does not capture: the *hasPolicies* relation from element to quote is the composition of the relations *hasPoliciesSet*, from elements to policy sets, and *hasPolicies*, from policy sets to policies. We will return to this missing constraint later.

Our task now is to capture information about the values that policies expect to obtain from quotes. We will define two cases, from which we will extract a basic pattern that can be used to extend our model when policies expect other kinds of values, as well as the two we consider, namely: *Type* and *MagicNumber*. *Type* and *MagicNumber* are defined by the TCG and are present in the whole

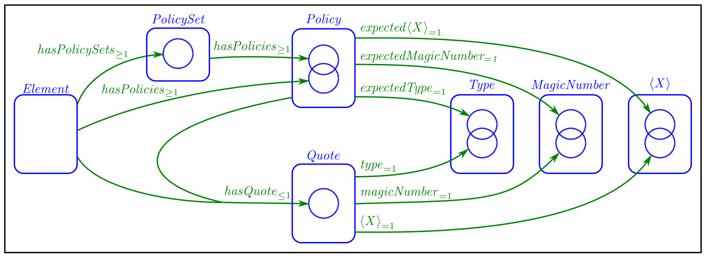

Fig. 6. Extending the model using patterns.

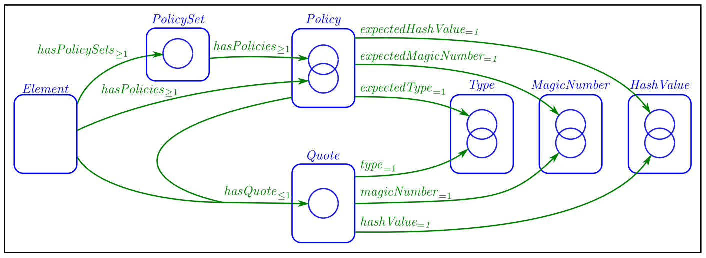

Fig. 7. Extending the model to include additional classes for measures.

quote structure. In this case *Type* is a value which encodes the type of the datastructure, that is, a TPM2 Quote, and magic number is a fixed value stating that the quote is from a TPM. We will then show how the pattern can be applied to a third case, *HashValue*, which is a single value calculated from recursive extension of the individual hash values of a PCR in a policy.

Figure 4 shows how to extend the basic data structure in figure 3 to include the class *Type* and the two new binary relations, *expectedType* and *type*. We ultimately want to check that the value expected by a policy, given an element and a quote, matches the value provided by the quote for a type. Each policy expects a unique value for *Type*. In figure 4, we have an arrow from *Policy* to *Type*, indicating that each policy is associated, under the relation *expectedType*, to a unique *Type* (indicated by the $_{=1}$ annotation). Similarly, each quote is associated with a unique thing in the class *Type*. When modelling rules that capture trust, we will require that the expected type for the policy matches the (actual) type for the quote in order for an element to be trusted.

Information about magic numbers is added to the model in figure 5, extending figure 4. We can see that the extension has a very similar structure to the previous case: add a new rectangle for the new class, with two further arrows, one from *Policy* and one from *Quote*. We adopt the naming convention that the arrow from *Quote* to the new rectangle, *MagicNumber* is labelled *magicNumber*. The other new arrow has the same label, but with a leading capital letter and then prefixed with *expected*, giving the label *expectedMagicNumber*. This leads to a general pattern for extending the basic data structure to include new kinds of classes for

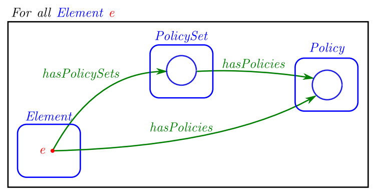

Fig. 8. Composing binary relations.

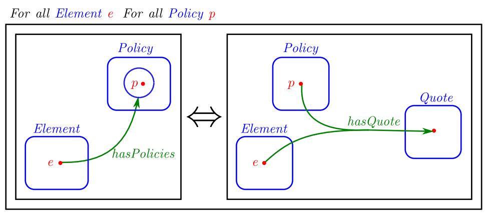

Fig. 9. Binary and ternary relations.

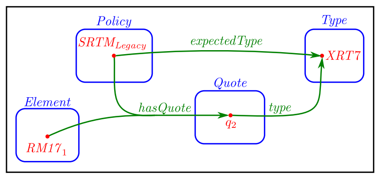

Fig. 10. A trusted element.

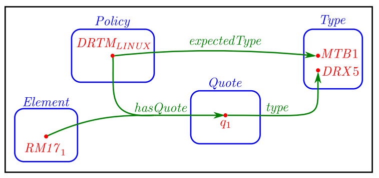

Fig. 11. An untrusted element.

which measurements can be obtained for quotes, shown in figure 6. The extension of figure 5 to include *HashValue*, shown in figure 7, is then straightforward.

Returning now to the missing constraint identified above, we show how to define the composition with reference to figure 8. We want to express that, for each element, the set of policies it 'has direct access to' under the *hasPolicy* relation are exactly those associated with its policy sets. Since we need to talk about all elements, the diagram in figure 8 includes a *quantification expression* above the bounding box. In the diagram, we can then see that if we navigate from (any element) e to its set of policies via *PolicySet*, under first the *hasPolicySets* relation and then the *hasPolicies* relation, we get to exactly the same set of policies if we go directly from e to policies under the *hasPolicies* relation.

To conclude this section, we now focus on the relationship between the *hasPolicy* and *hasQuote* relations, shown in figure 9. This diagram also uses a quantification expression, allowing us to talk about all elements and all policies. In addition, it incorporates a standard logical connective, $\Leftrightarrow$, to represent bi-implication. Reading the diagram, inside the bounding rectangles, we see:

1. if e is an element, p is a policy and e *hasPolicy* p then there is a unique quote (here represented by the unlabelled dot) where element e and policy p have the quote.
2. if e is an element, p is a policy and there is some unique quote where e and p have that quote then element e has policy p.

Using this data structure and its associated constraints, we can now proceed to define rules of trust.

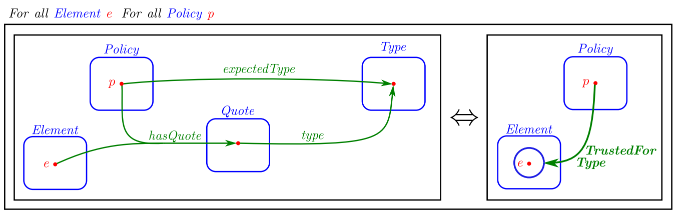

Fig. 12. Defining a rule of trust for the *Type* class.

4 Rules over Elements and Policies

Intuitively, we want expected measures to be the same as actual measures in order to trust an element. Looking at the data structure visualised in figure 7, we can see that we must navigate from elements and policies to quotes in order to get access to (actual) measures for particular classes, such as *Type*. Essentially, rules of trust are binary relations, from policies to elements: given any policy, it can either trust an element or not.

Figure 10 shows an instance where the element $RM17_1$ is trusted by policy $SRTM_{Legacy}$. The quote provided is associated with type *XRT7* which is the value expected by $SRTM_{Legacy}$. By contrast, figure 11 shows an example where a different policy does not trust $RM17_1$. Here, $RM17_1$'s quote, given the policy $DRTM_{Linux}$, is q_1, which returns *DRX5* as the type. But, the policy $DRTM_{Linux}$ expects the value $MTB1$. These values for *Type* do not match, so $DRTM_{Linux}$ does not trust $RM17_1$. This example illustrates that some policies can trust an element whilst other do not.

We define our first rule of trust, in figure 12, for the *Type* class. Again, this diagram uses a quantification expression: we need access to all elements, e, and all policies, p. Figure 9, expresses that if e has the policy p then there is a unique quote, q, for that element and policy pair. Whenever we have a (unique) quote, q, we can get access to q via the *hasQuote* relation. At that point, we can obtain both the expected value, t_1, of *Type* for the policy *and* the actual value t_2, of *Type* for the quote. Only when $t_1 = t_2$ does p trust e for the *Type* class. In the diagram of figure 12, the left-hand rectangle constructs the relationships between e, p, and their quote: the two arrows *expectedType* and *type* hit the same dot (which represents an anonymous individual), expressing that the expected and actual measures are the same. The policy p *trusts* e given the expected and actual measures for the class *Type*. Likewise, if p trusts e for *Type* then the expected and actual measures for *Type* must match. Essentially, the diagram in figure 12 is defining the relation *trustedForType*: given p, the set of elements p is related to under *trustedForType* is precisely the set of things that p trusts for *Type*.

Having defined what it means to be trusted by a policy for the class *Type*, we can readily deduce what it means to not be trusted. Suppose that policy p does not trust element e, as shown in figure 13. This figure depicts that e is not trusted by p since it is outside the (curve that represents) the set of things that

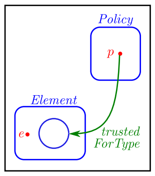

Fig. 13. A policy does not trust an element.

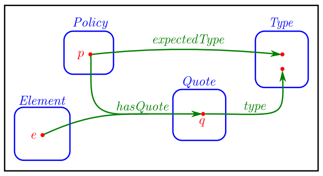

Fig. 14. Measures do not match.

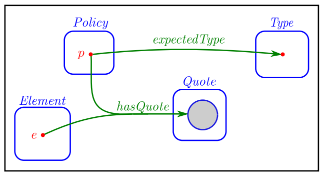

Fig. 15. There is no quote.

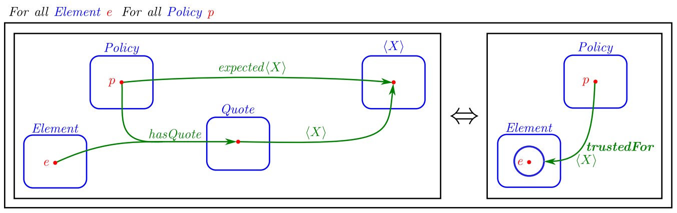

Fig. 16. Pattern for defining a rule of trust.

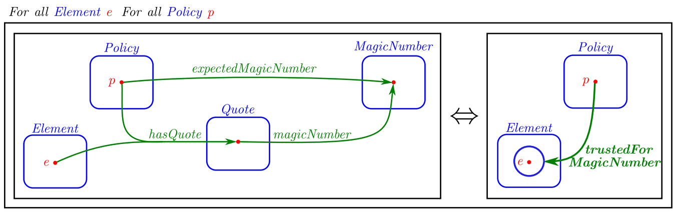

Fig. 17. Defining a rule of trust for the *MagicNumber* class.

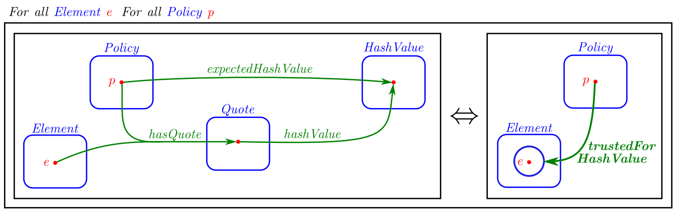

Fig. 18. Defining a rule of trust over the *HashValue* class.

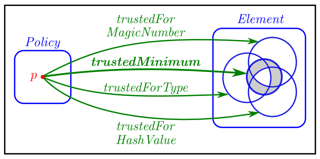

Fig. 19. A minimum level of trust.

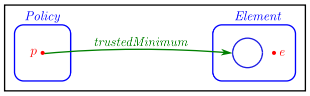

Fig. 20. A minimum level of trust does not hold.

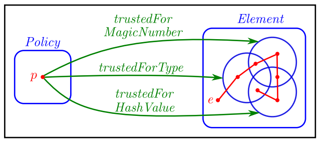

Fig. 21. A non-trusted element.

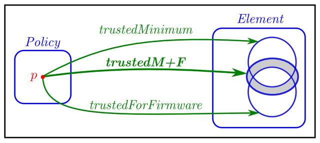

Fig. 22. A higher level of trust.

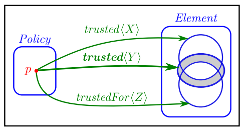

Fig. 23. A simple attestation rule pattern: adding one rule of trust.

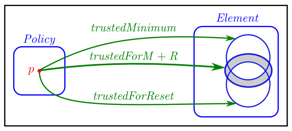

Fig. 24. Extending *trustedMinimum* to include *Reset*.

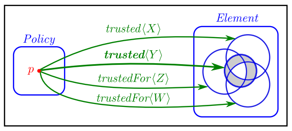

Fig. 25. A more complex attestation rule pattern: adding two rules of trust.

p trusts for *Type*. How can this situation arise? Well, the obvious case is shown in figure 14, where the expected value for the policy does not match that given by the quote. Perhaps a less obvious case is covered in figure 15, where there is no quote for e and p. The construction here builds the set of quote for p and q using an unlabelled curve. The shading inside the curve is used to denote the emptiness of the corresponding set: there are no quotes for e and p. Given the data structure in figure 4 along with figures 8 and 9, we can deduce that p is not a policy of q in this case. So, the only way that an element can fail to be trusted *for one of its policies* is when the measures do not match.

Returning our attention to defining rules of trust, we can readily extract a pattern from figure 12. For any given rule, we want the expected measures and actual measures to have the same value, for a given class; the generic pattern is shown in figure 16. This pattern is used to create figures 17 and 18, defining rules of trust over the *MagicNumber* and *HashValue* classes. Many more rules of trust can be defined for other kinds of things in the TPM quote field. These things include Firmware, Clock and Safe (which denotes clock integrity), amongst others.

5 Attestation Rules Defined from Rules of Trust

We can use individual rules of trust to define different levels of trust, giving rise to attestation rules over sets of classes. As a minimum, we require all three rules of trust to pass to provide a *minimum* level of trust for an element given a policy. This minimum level of trust, for the purposes of attestation, is defined in figure 19. The diagram expresses that for each policy, p, the set of things in which p has a minimum level of trust is exactly the set of things that p trusts under the three previously defined rules of trust. Notice the use of shading here: the region inside the target of the *trustedMinimum* arrow that is not inside the targets of *all* the other arrows is shaded, so no elements can be in (the set represented by) that region. Given *trustedMinimum*, elements are attestable by policies.

Suppose, for a given element e and policy p we have the situation shown in figure 20: p does not have a minimum level of trust in e. This can arise when p fails to trust e for at least one of the rules of trust used to define *trustedMinimum*. There are seven ways in which this can happen, illustrated in figure 21: e must lie in one of the regions that contain a node of the tree labelled e. The tree thus represents disjunctive information, showing that e fails all of the rules of trust (represented the node outside the three unlabelled curves), or exactly one of the rules of trust (represented by the three nodes each inside exactly one unlabelled curve), or any pair of the rules of trust (represented by the other three nodes). It is possible to prove (using sound diagrammatic inference rules) that these are the possibilities for e given the lack of trust by p.

Further attestation rules can be defined, exploiting different sets of rules of trust, to give stronger levels of trust. This is useful since, in some situations, having a basic level of trust in an element is sufficient for the purposes of attestation, yet at other times a stronger level of trust is required. To illustrate, TPM includes Firmware as one of its quote fields. Assuming the pattern in figure 16 is used to define the *trustedForFirmware* rule of trust, we can define a higher level of trust, which we call *trustedM+F*. This is shown in figure 22, where we can see that the set of elements that policies trust under *trustedM+F* is exactly the set of elements trusted under both *trustedMinimum* and *trustedForFirmware*.

From this, we can readily extract a pattern, given in figure 23, for defining higher levels of attestation rules of trust. This pattern takes one attestation rule and extends it using one rule of trust. Suitable names should be substituted for $\langle X \rangle$, $\langle Y \rangle$ and $\langle Z \rangle$, as in figure 24 which defines an attestation rule, *trustedForM+R*, for the *Reset* TPM quote field, assuming that *trustedForReset* has been defined. The pattern readily generalises to allow attestation rules to be extended by two or more rules of trust at the same time. The pattern for extending by two rules of trust is given in figure 25. It is easy to see that by defining attestation rules in this way, a partial order over them is derivable. This partial order allows the level of trust for an element, given a policy, to be ascertained.

6 Human Cognition and the Benefits of Diagrams

A major motivation for using diagrams to model systems, in our case rules of trust in the context of attestation, is the potential benefits they bring from the perspective of human cognition. It is important for models to be properly understood by all stakeholders. Focusing specifically on concept diagrams, there is empirical evidence to suggest that they are superior modes of communication compared to symbolic notations (specifically description logic) and textual notations (specifically the Manchester OWL syntax (MOS)) [2, 7]. Here we give some explanation as to why these diagrams have cognitive benefits by appealing to *well-matchedness* [6].

A notation is said to be well-matched when its syntactic relations, that convey semantics, 'match' the semantic concepts being conveyed. In concept diagrams, spatial properties, such as the disjointness of the interiors of closed curves, are used to express information. Disjoint curve interiors assert that the sets are disjoint, for example: this is a well-matched feature. Likewise, the region inside the overlap of curves represents the intersection of the represented sets and so on. Arrows are directed and are used to express directed relationships. Hence, concept diagrams are generally well-matched to their semantics.

Well-matched diagrams often have *free rides* which correspond to information that can readily be extracted from the diagram but which needs to be inferred from symbolic or textual representations [13]. For instance, from figure 21, by expressing that p is a policy, e is an element and the sets policy and element are disjoint, we can read off the diagram that p and e are distinct. An equivalent MOS representation of the information just described comprises the statements

`ClassAssertion( Policy p )`,
`ClassAssertion( Element e )`, and
`DisjointClasses( Policy Element )`

from which one needs to deduce that p and e are distinct individuals. The diagram possesses other free rides also, such as directly expressing the fact that all of the individuals to which p is related under *trustedForType* are not policies; this free ride follows from the containment of the unlabelled curve targeted by the arrow labelled *trustedForType* being inside the *Element* curve which, in turn, is disjoint from the *Policy* curve.

7 Conclusion and Future Work

We have presented a diagrammatic approach to modelling rules of trust, over a specified data structure, specifically for use in the context of the remote attestation of TPM-based computer systems. Using rules of trust, which determine the trustworthiness of system elements for given policies and specific classes, we were able to define attestation rules to decide on the trustworthiness of elements over sets of classes. If we detect a failure of trust, for a particular attestation rule, we know that one or more of the rules of trust have failed. Once a failure has been recognised, the next stage is to understand the root cause, with the attestation rules thus contributing to system diagnosis and forensics. The partial order over the attestation rules further permits the network routing provision to select a more trusted environment for running sensitive and critical workload.

The data structure, rules of trust, and attestation rules that we have provided, along with patterns that allow their extension to more complex environments, are suitable for a static view of trustworthiness. It is important to define rules of trust for the temporal case, which can include pairs of policies with expected values that change over time. Currently, there is no diagrammatic logic that is capable of formalising temporal constraints over the rich data structures, and corresponding rules that arise in the trust and attestation requirements of TPM-based computer systems, which has been shown to be more effective for people to use than competing symbolic and textual notations. This is an interesting avenue of future work, since it is desirable to exploit the established usability benefits of diagrams in the temporal case. As a result, we would expect a deeper, more accurate, understanding of the individual rules of trust and attestation to be gained by the designers of networks and systems in which these temporal rules are needed.

References

1. Abrial, J.: The B-Book: Assigning Programs to Meanings. CUP (1996)
2. Alharbi, E., Howse, J., Stapleton, G., Hamie, A., Touloumis, A.: Visual logics help people: An evaluation of diagrammatic, textual and symbolic notations. In: Visual Languages and Human-Centric Computing. pp. 255–259. IEEE (2017)
3. Baader, F., Horrocks, I., Sattler, U.: Description logics as ontology languages for the semantic web. In: Mechanizing Mathematical Reasoning. pp. 228–248. Springer (2005)
4. Chapman, P., Stapleton, G., Howse, J., Oliver, I.: Deriving sound inference rules for concept diagrams. In: IEEE Symposium on Visual Languages and Human-Centric Computing. pp. 87–94. IEEE (2011)
5. Dai, W., Jin, H., Zou, D., Xu, S., Zheng, W., Shi, L., Yang, L.: Tee: A virtual fdrtmg based execution environment for secure cloud-end computing. Future Generation Computer Systems **49**, 47–57 (2015)
6. Gurr, C.: Effective diagrammatic communication: Syntactic, semantic and pragmatic issues. Journal of Visual Languages and Computing **10**(4), 317–342 (1999)
7. Hou, T., Chapman, P., Blake, A.: Antipattern comprehension: An empirical evaluation. In: Formal Ontology in Information Systems. Frontiers in Artificial Intelligence, vol. 283, pp. 211–224. IOS Press (2016)
8. Howse, J., Stapleton, G., Taylor, K., Chapman, P.: Visualizing ontologies: A case study. In: International Semantic Web Conference. pp. 257–272. Springer (2011)
9. Oliver, I., Holtmanns, S., Miche, Y., Kalliola, A., Lal, S., Ravidas, S.: Experiences in trusted cloud computing. In: 11th International Conference on Network and System Security. pp. 19–30. Springer (2017)
10. Sato, Y., Masuda, S., Someya, Y., Tsujii, T., Watanabe, S.: An fMRI analysis of the efficacy of Euler diagrams in logical reasoning. In: IEEE Symposium on Visual Languages and Human-Centric Computing. pp. 143–151. IEEE (2015)
11. Sato, Y., Mineshima, K.: How diagrams can support syllogistic reasoning: an experimental study. Journal of Logic, Language and Information **24**, 409–455 (2015)
12. Shams, Z., Sato, Y., Jamnik, M., Stapleton, G.: Accessible reasoning with diagrams: from cognition to automation. In: Diagrams. Springer (2018)
13. Shimojima, A.: Semantic Properties of Diagrams and Their Cognitive Potentials. CSLI Publications, Stanford, CA, USA (2015)
14. Spivey, J.: The Z Notation: A Reference Manual. Prentice Hall (1989)
15. Stapleton, G., Howse, J., Chapman, P., Delaney, A., Burton, J., Oliver, I.: Formalizing concept diagrams. In: 19th International Conference on Distributed Multimedia Systems. pp. 182–187. Knowledge Systems Institute (2013)
16. Yang, Y., Shiwei, X., Hunguo, Z., Fan, Z.: Using first order logic to reason about TCG's TPM specification. In: International Forum on Information Technology and Applications. pp. 259–263. IEEE (2009)
17. Zimmer, V., Dasari, R., Brogan, S.: Tcg-based firmware: White paper by Intel Corporation and IBM Corporation Trusted Platforms. https://people.eecs.berkeley.edu/kubitron/cs194-24/hand-outs/SF09_EFIS001_UEFI_PI_TCG_White_Paper.pdf (2009), accessed May 2018.