Type Dependent Policy Language

Anna Rose Fritz

B.S. Chemical Engineering, University of Kansas, 2019

Submitted to the graduate degree program in Department of Computer Science and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Masters of Science.

Perry Alexander, Chairperson

Committee members

Alexandru Bardas

Andy Gill

Date defended: May 6th, 2021

The Thesis Committee for Anna Rose Fritz certifies
that this is the approved version of the following thesis :

Type Dependent Policy Language

Perry Alexander, Chairperson

Date approved: May 6th, 2021

Abstract

Remote attestation is the act of making trust decisions about a communicating party. During this process, an appraiser asks a target to execute an attestation protocol that generates and returns evidence. The appraiser can then make claims about the target by evaluating the evidence. Copland is a formally specified, executable language for representing attestation protocols. We introduce Copland centered negotiation as prerequisite to attestation to find a protocol that meets the target's needs for constrained disclosure and the appraiser's desire for comprehensive information. Negotiation begins when the appraiser sends a request, a Copland phrase, to the target. The target gathers all protocols that satisfy the request and then, using their privacy policy, can filter out the phrases that expose sensitive information. The target sends these phrases to the appraiser as a proposal. The appraiser then chooses the best phrase for attestation, based on situational requirements embodied in a selection function. Our focus is statically ensuring the target does not share sensitive information though terms in the proposal, meeting their need for constrained disclosure. To accomplish this, we realize two independent implementation of the privacy and selection policies using indexed types and subset types. In using indexed types, the policy check is accomplishes by indexing the term grammar with the type of evidence the term produces. The statically ensures that terms written in the language will satisfy the privacy policy criteria. In using the subset type, we statically limit the collection of terms to those that satisfy the privacy policy. This type abides by the rules of set comprehension to build a set such that all elements of the set satisfy the privacy policy. Combining our ideas for a dependently typed privacy policy and negotiation, we give the target the chance to suggest a term or terms for attestation that fits the appraiser's needs while not disclosing sensitive information.

Acknowledgements

First and foremost, I would like to thank Dr. Perry Alexander for encouraging me to go to graduate school and study computer science. I was first introduced to the charm of computer science when we began discussions about coding and proving late in my undergraduate career. Once I expressed further interested, he encouraged me to seek hard problems and to have fun while solving them. And with that, my graduate studies began. I am so grateful for the time and dedication he put into completing this work and to teaching me the things I know. It would not have been possible to do this without his efforts.

Next, I would like to thank everyone in the research lab who answered my many questions. When I first started research, it seemed like every day I would need help downloading an application or understanding a topic and they were always there smiling, ready to help. Through slack messages, zoom meetings, and the summer reading groups, I have learned so much from this group and would not be successful today without them.

I would also like to thank my teachers who taught me about things I never dreamed existed. I remember learning compilers and understanding how the pieces fit and my mind exploding. It seemed that every class was always something new and exciting. I know it takes a lot of dedication to make a course entertaining and I am lucky to have had professors that care.

Last but not least, I would like to thank my family and friends who have believed in me and encouraged me to continue believing in myself. Specifically, I want to thank my mom and dad who have provided constant love and support even when I was frustrated and didn't think I would succeed. They are two of the best people I know and I can only hope I am half as kind and considerate as they are.

They say it takes a village to raise a child. Well, I say it takes a village to help someone graduate. This truly could not be possible without so many people whose time shaped me and this work. I am so thankful to have them all.

Contents

List of Figures

Chapter 1

Introduction

As the internet grows, cyber attacks are becoming increasingly common and exceptionally clever. Even more formidable than the attacks themselves is the fact that users have little to no idea that they are under attack as many do not have an infrastructure for protection. Keeping user's safety at the forefront, Coker et al. (2011) set out to understand common characteristics of attacks to develop a verification scheme to protect users. They discovered most attacks were executed remotely, with standard computers, and were used to gain access to individual entities as opposed to a network of systems. They also discovered that the attacks are often able to obtain user's confidential information, such as bank passwords, by inserting malicious code. One way hinder these attacks and protect users would be to ensure a communicating peer's operating system is secure before making a trust decision (Loscocco et al., 2000). To do that, there would need to be an infrastructure that could attest to the state of the software executing on a remote machine so as to allow the user to make an informed trust decision before engaging with the remote party. With that, the concept of *remote attestation* was born as a way to ensure a communicating party should be trusted.

With the goal of making an informed trust decision about a communicating party over the network, we consider the formal definition of trust and the principles that guide development of attestation architectures, as laid by Coker et al. (2011). First, we define trust as having strong identity and observed good behavior. That is, trust is a relationship dependent on the situation and the identity of the communicating parties (Xiu & Liu, 2005). Taking into consideration the variety of situations and identities, the attestation architecture must be flexible because the same structure will be used to determine the trustworthiness of an enterprise of machines as well as an at home user's personal computer. Furthermore, the attestation architecture must operate with information that reflects the system's running state to obtain an accurate, authentic measurement. Finally, the attestation architecture must allow the communicating parties to maintain confidentiality and protect sensitive information that may compromise their system. Implementing a flexible framework, that reflects the running state of software, and considers communicating parties privacy standards, is not an inherently obvious task. Yet, these goals are formalized with the principles of remote attestation and are achieved with the attestation architecture. If following these principles, the remote attestation architecture could be critical to prohibiting cyber attacks in the future.

Currently, the steps to preform remote attestation begin with an appraiser asking a target to take a measurement which produces evidence. The appraiser then collects the evidence and evaluates it to make a trust decision. While implementing a correct, formally verified attestation architecture is critical, we must also consider a framework to obtain a sensible measurement. Before this work, a measurement was statically selected as a result of the context of communication between known systems. While this is useful to gain an understanding of the behavior of the attestation architecture and provide basic measurement operations, the framework neglects parties that must choose a measurement at runtime as they may not understand the system with which they are communicating. It also does not consider measurement operations that a communicating party does not want to perform because it could force them to expose sensitive information which may leave them vulnerable to attacks.

We introduce a framework, on top of the existing attestation work, that allows communicating parties to negotiate to determine an acceptable measurement for attestation. That is, we introduce the concept of *negotiation* as a back and forth communication routine resulting in a situationally dependent measurement that both parties find acceptable for attestation. To situate the work in the context of attestation, we reveal the negotiation framework in Figure 1.1. The routine begins when the appraiser sends a request causing the target to gather phrases that satisfy the request while ensuring the phrases selected do not expose sensitive information. These phrases are sent back to the appraiser where the appraiser can then select a best term for attestation. Following negotiation, the attestation framework runs as intended to verify the target is in a trusted state.

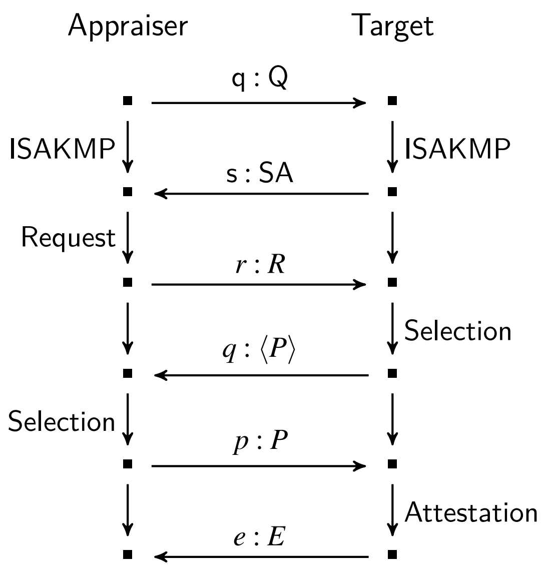

Figure 1.1: Negotiation process.

Within the negotiation framework, we realize the communicating parties privacy requirements in a type dependent fashion. That is, in order to protect a place's interests, we introduce the idea of a type dependent *privacy policy* that statically guarantees the target does not share a measurement that may expose sensitive information. We are not concerned with understanding what information should be remain confidential but rather developing a framework to represent the policy that is sound and logically correct. To motivate the use of dependent types, we would like to apply a more expressive type that ingrates programs and their specifications in one grammar. That is, by writing the structure in a type dependent manner, we can statically ensure evidence for attestation will only be shared if it satisfies a target's privacy requirements.

Taking this discussion into consideration, the objective of this thesis is to describe a negotiation framework that allows the target and appraiser to agree upon a term for attestation where the chosen measurement does not violate the target's privacy standards. To statically enforce the privacy standards within the negotiation framework, this thesis explores ways to use dependent typing to realize a situationally dependent privacy policy.

We begin this thesis by discussing the important background information in Section 2. In Section 3, we discuss the work surrounding the development of negotiation which lays a framework for determining the measurement that can be used to make a trust decision. We also introduce the ideas of the *privacy policy* and *selection policy* as a means to capture the context of the communication routine. We realize the privacy and selection policies in a type dependent fashion in Section 4. In Section 5, we provide an example to help the reader to better understand the integration and implementation of this framework. We conclude with Section 6 where we also discuss the future work surrounding negotiation and the type dependent policies.

Chapter 2

Background

2.1 Remote Attestation

As we established in the introduction, remote attestation is one way to make a trust decision about a communicating party over a network (Coker et al., 2011). The attestation process involves two parties: the target and the appraiser. The *appraiser* is the party requesting the attestation while the *target* or *attester* is the responding party. To begin the attestation sequence, the target receives a request for a specific piece of evidence in the form of an attestation protocol. The target executes the protocol to generate the requested evidence and then sends the result back to the appraiser. Upon receiving the evidence, the appraiser completes the *appraisal* to make a trust decisions about the target where appraisal is the target's decision making process. An overview of the process can be visualized in Figure 2.1 below.

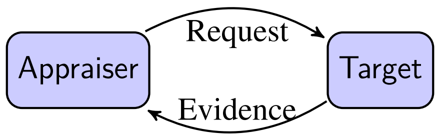

Figure 2.1: Remote attestation architecture showing an *appraiser* making an attestation request of a *target*.

To guide the design of attestation systems, Coker et al. (2011) defined the principles of attestation. These principles, described below, influence the design considerations when developing attestation systems.

1.Fresh Information

2.Comprehensive Information

3.Constrained Disclosure

4.Semantic Explicitness

5.Trustworthy Mechanism

The first principles state the desire for attestation systems to provide *fresh information* and *comprehensive information*. The former assures evidence reflects the running state while the latter assures the evidence generated by the target provides the appraiser with a complete and total understanding of the target's state. The principle of *constrained disclosure* protects the target and appraiser from sharing sensitive information by distinguishing what information should remain private. Evidence that is deemed sensitive may change situationally and is perhaps best implemented with an access control policy. Next, *semantic explicitness* guarantees attestation operates with uniform semantics. This means the language has logical inferences and target's identity is easily discernible. The last principle is to ensure that the mechanism used to determine trust is, in itself, trustworthy. This criteria ensures that the measurement tools will provide accurate results.

Taking into consideration the guiding principles, a language for remote attestation, Copland, was born (Ramsdell et al., 2019). Copland provides the syntax to write attestation protocols as well as an environment to execute those protocols. The protocols define measurements, where the act of an appraiser *measuring* a target is the idea of making claims about the target's current state (Pendergrass et al., 2017). The language also defines dispatching, sequencing, and evidence bundling operations where dispatching operations request evidence from specific places and sequencing operations are used to order measurements (Petz, 2020). Maintaining the correct measurement ordering and the correct evidence bundling are fundamentally important to ensuring attestation results are correct (Coker et al., 2008). As such, we emphasize that order is persevered through the Copland semantics. Additionally, Copland introduces metaevidence as a means to represent evidence properties and provide additional information regarding the situation. This means the measurement generated by the appraiser may ask for evidence, metaevidence, or a combination of both. In any case, the measurement must be complete and describe properties that are useful to make a trust decision.

The Copland grammar, as seen in Figure 2.2, is taken from Petz (2020) and Ramsdell et al. (2019) to describe attestation protocols. In this grammar, A describes the basic actions such as copy, signing, and hashing. A measurement preformed by an attestation service provider (ASP) would be represented with the syntax ($\mathsf{ASP}\ m\ \bar{a}\ p\ r$) where m is a number to identify the measurement, $\bar{a}$ is the measurement arguments, p is the place where the measurement is taken, and r is the measurement target. The nonterminal t is necessary to combine terms in the language. The $@_p t$ is an at operation to request a measurement of a different place p. Term may be executed in sequence, $(t_1 \rightarrow t_2)$, parallel, $(t \overset{\pi}{\prec} t)$, or any any order, $(t \overset{\pi}{\sim} t)$. After describing terms and the ordering operations, the evidence structure, E, is the final nonterminal in the Copland grammar. The E or evidence structure is the result of executing a Copland phrase or term. The evidence returned my be empty, ξ. It may also be a user space measurement of place p, $\mathsf{U}_P(E)$, or a kernel integrity measurement of place p from place p, $\mathsf{K}_P^P(E)$. Evidence gathered in sequence is denoted $(E ;; E)$ while evidence gathered in parallel is denoted $(E \parallel E)$.

$$\begin{array}{rcl} t & \leftarrow & A \mid @_p t \mid (t \rightarrow t) \mid (t \overset{\pi}{\prec} t) \mid (t \overset{\pi}{\sim} t) \\ A & \leftarrow & \mathsf{ASP}\ m\ \bar{a}\ p\ r \mid \mathsf{CPY} \mid \mathsf{SIG} \mid \mathsf{HSH} \mid \cdots \\ E & \leftarrow & \xi \mid \mathsf{U}_P(E) \mid \mathsf{K}_P^P(E) \mid [\![E]\!]_P \mid \#_P E \mid \mathsf{N}_P(E) \mid (E ;; E) \mid (E \parallel E) \mid \cdots \end{array}$$

Figure 2.2: Copland Phrase Grammar

To motivate this work, we consider an example where an appraiser wishes to gain a better understanding of the target's state with a measurement of its virus checker. There are many possible Copland phrases the appraiser could request. The simplest would be to ask for a candid measurement of the virus checker, as seen in Figure 2.3 number 1. This measurement would reveal state of the target's virus checker, vc. To enhance the measurement, as observed with number 2,

1. $@_p\,[(\text{ASP}\,vc\,\bar{a}\,p\,t)]$
2. $@_p\,\{n\}[(\text{ASP}\,vc\,\bar{a}\,p\,t) \to \text{SIG}]$
3. $@_p\,\{n\}[@_{ma}\,\{n\}[(\text{ASP}\,h\,\bar{b}\,p\,v) \to \text{SIG}] \to (\text{ASP}\,vc\,\bar{a}\,p\,t) \to \text{SIG}]$
4. $@_p\,\{n\}[@_q\,\{n\}[(\text{ASP}\,m\,\bar{c}\,q\,ss) \to \text{SIG}] \to (\text{ASP}\,vc\,\bar{a}\,p\,t) \to \text{SIG}]$

Figure 2.3: Possible phrases, written in Copland, for virus checking example.

the appraiser could ask for a nonce and the same measurement of the virus checker, but signed. Both the signature and the nonce would be considered metaevidence and exist to support the target's trustworthiness. Another option to provide a more comprehensive measurement, is for the appraiser to ask for measurement of the virus checker's operational environment before the virus checker measurement. This syntax can be visualized with number three in 2.3 where *ma* is a place that has access to $p's$ environment and *h* is the measurement of *p*'s operating system. In another situation, the appraiser may want a measurement of signature server instead of the operational environment. This is seen with number four in 2.3 where *q* is the signature file server. The benefit of having multiple phrases that may include the same measurement is the varied application of these measurements for different circumstances. That is, if the appraiser wanted the attestation routine to be quick, they may choose the first measurement. If they wanted attestation to provide the most detailed evidence, they may choose the third or fourth measurements. By providing many options, that all include the virus checker measurement, we allow for the context and situation to determine which one would be most beneficial.

2.2 ISAKMP

Network security is a broad, general phrase that can be summarized with the goals of maintaining confidentiality, integrity, and authenticity. Confidentiality states that malicious user should not be able to eavesdrop on the contents of messages while integrity means that an attacker cannot change the contents of messages. Authenticity ensures the sender is communicating with the intended receiver. In an attempt to meet all three networking goals and enhance network security, the networking concept of the Internet Security Association and Key Management Protocol

(ISAKMP) was born (Hajjeh et al., 2003).

When created in Request for Comments (RFC) 2408, ISAKMP was designed to establish a secure channel that protects subsequent communications through the negotiation, creation, modification, and deletion of *security associations* (SA) (Hecker, 2002). A security association is an agreement between the communicating parties that protects subsequent network traffic. It is akin to a tunnel and defines components such as the situation and domain of interpretation. That is, the *situation* describes the context of communication. Understanding the context is critical as it includes all security relevant information and allows for the peers to decide on the security requirements for the current session (Maughan et al., 1998). An example of something that may be defined in the situation is a hash algorithm used during communication (Maughan et al., 1998). Additionally, the *domain of interpretation* (DOI) is established as a result of ISAKMP to define the negotiated parameters. These parameters include the situation, security policies, syntax, naming schemes, payload formats, and additional exchange types (Maughan et al., 1998). It is important to emphasize, the DOI ensures a consistent naming scheme in order for the two parties to communicate effectively.

To offer flexibility, ISAKMP implements different exchange types and payload types as motivated by different contexts for communication. Exchange types define the ordering of messages in the communication routine, with five possible options (Qing & Adams, 2006). Choosing an option is a function of the situation where different options include difference exchanges that allow parties to share more or less information. Payloads are dependent on the situation and the exchange type but all have the same fixed header. The body of the payload changes situationally and can contain various pieces of information, thus providing flexibility.

The process of ISAKMP occurs in two phases where the first phase establishes protection for the second phase (Qing & Adams, 2006). The basic security association (SA) is established in the first phase which includes a label to identify the communication (Hajjeh et al., 2003). The first phase incurs a high operating cost and is therefore preformed intermittently. The second phase is preformed regularly and is designed to exchange key material and other security protocols to ensure greater security between the communicating parties (Ramalingam, 2017). As a result of the second phase, the two parties are authenticated.

At its core, ISAKMP outlines a negotiation mechanism between two parties that enables the parties to set up a tunnel for secure communication. Upon completion of the framework, the initiating party and responding party obtain a security association which situates the communication and provides a domain of interpretation. In looking at the process, the flexibility of ISAKMP allows the responder and initiator to agree on what is to be shared and the order in which it will be shared. This establishes a framework for our work.

2.3 Dependent Types

According to Cardelli (1996), type soundness is defined as the absence of type errors that are a result of ambiguous program implementation. Without type soundness, the program can be vulnerable to attacks designed to take advantage of these errors. For example, tasks like taking the front of an empty list or the predecessor of zero are undefined an often result in an error or incorrectly defined computation. In the predecessor of zero example, it may be defined such that the predecssor of zero is zero which avoids throwing an error but may result in future, unforeseen issues. However, with dependent types, we can constrain the domain to prove typing errors cannot occur. In essence, dependent types combat undefined operations to statically prohibit their computations.

In a dependently typed system, McBride (2002) states that types are first class objects. They can be used as an input to a function or can result from a function call. In reasoning about dependent types this way, they become programs with an added systematic constructs. McBride (2002) formalizes this discussion by stating that a key idea of dependent types is understanding a type family. The function `F:: T` $\rightarrow$ `Type` where `F` is a collection of types indexed in `T`. The application `F t` generates a type in the family. We can define families of types inductively which leads us to the idea that the behavior of one computation will effect subsequent computations.

There are many ways to implement dependent types to statically capture program properties.

To better understand this idea, we reason about an example using lists. Lists are common structures used in traditional programming that involve predefined operations over the list. Typically, a `front` function exists to capture and operate on the first element of a list. As long as a list is not empty, this is an acceptable thing to do. But, in the case that the list is empty, this would be an invalid operation and could leave the program vulnerable. One way to ensure this front function is never called on an empty list is to statically enforce that the list's size must be greater than zero before the front function can operate. To accomplish this, we implement a dependently typed front function using indexed types and subset types in the subsequent sections.

2.3.1 Indexed Types

Indexing is a common operation to help characterize and label objects, providing object with a context to ensure they are useful (Zenger, 1997). For example, phone numbers are simply numbers indexed with area codes. When looking to call businesses in a certain area, it is helpful to use the area code to exclusively seek objects that satisfy the location criteria. In essence, a filter is applied based on location. The same idea can apply to an indexed type system where the types can be written to capture properties about the system (Xi & Pfenning, 1999).

The goal of an index typed language is to eliminate the chance a program can reach defective edge cases. It is favorable because a program's properties can be enforced statically by avoiding the overhead that type checking induces (Zenger, 1997). The overhead is avoided because each object created in the indexed type system has an accompanying proof that it satisfies the necessary criteria. Proofs are computed statically and therefore do not incur any runtime cost (Fogarty et al., 2007).

We aim to showcase the practicality of indexed types with the example presented in Figure 4.10 which ultimately prohibits the front function from being called on an empty list. First we implement the list structure which has the type signature `nat -> Type` where the `nat` argument is the length of the list. It has two constructors, a nil and a cons constructor, where the `Inil` constructor has type `ilist 0`, meaning the nil list has length 0. The `Icons` constructor takes an element of type X and an existing list of length n, adds X to the list, and adds one to the length of the list. In this way, we build lists that are tagged with their lengths. In line 5, we write my_list to build a list using the ilist syntax. This list, having one element, will be of type ilist 1 thus accomplishing the goal of embedding the length. With the structure for lists, we can now write a front function that takes into consideration the length of the list before preforming the front operation. This front function is implemented in lines 11 through 15. It works by accepting a list of length n and completing a proof that n is greater than zero, before returning the first element of the list. For the Inil case, the proof that must be completed is zero_gtz. The precondition of the proof $0 > 0$ which is not true. Because the precondition is false, the proof can be completed. In the ICons case, the proof is trivial and can be replaced by an underscore and Coq can infer the proof satisfaction.

The definition of front produces a dependent pair. The first element of the dependent pair would be the value while the second element would be a proof that the length of the list is not equal to zero. If attempting to provide the front function with a nil list, the user will need to provide a proof that $0 > 0$ which is impossible so the function does not evaluate. A correct instance of the function is visible in line 20 where we want to compute the front of my_list. In doing so, we must provide the front function with value and proof. The value is the list, my_list and the proof, one_gtz, concludes that the length of the list is greater than zero. The result of computing this function is 3.

2.3.2 Subset Types

Subset types allow for a more mathematical reasoning about types (Sozeau, 2007). They provide terms in the language with more information by constraining the type's arguments. The subset can be envisioned, in a mathematical sense, as a predefined set with some filter applied over the elements of the set. Formally, a subset type has the form $\{x \in \mathrm{S} \mid \mathrm{P}\}$ where S is a set and P is an operation that can be used over the set. The subset includes all x for which the property P holds. If attempting to reason further about x, one should have confidence in x abiding by the rule of P

```
Inductive ilist {X:Type} : nat -> Type :=
| Inil : ilist 0
| Icons : forall n, X-> ilist n -> ilist (S n).

Definition my_list := (Icons 3 Inil).

Lemma zero_gtz : 0 > 0 -> False.
  intros; inversion H.
Qed.

Definition front {X:Type} n (l:ilist n) : n>0 -> X :=
match l with
| Inil => fun pf:0>0 => match zero_gtz pf with end
| Icons _ x _ => fun _ => x
end.

Lemma one_gtz : 1 > 0.
  Proof. auto. Qed.

Compute front (my_list) one_gtz.
(* = 3
  : nat*)
```

Figure 2.4: Indexed Type Example

and need not worry about P moving forward. In Coq (Coq development team, 2016), a subset type can be thought of as a dependent pair. That is, each instance of a subset type includes a value and a proof that the value satisfies the predicate (Chlipala, 2013). The proof and the value must be provided simultaneously (Sozeau, 2007).

To emphasize the similarities and difference in indexed types and subset types, we attempt a subset typed version of the front function in Figure 2.5. First, the inductive structure for a list is written in lines 1-3. Here, we define `Snil` to represent the empty list and `Scons` to expand the list. Next, we attempt to write a definition, `front'`, to return the front of a list while making it statically impossible to take the front of an empty list. In order to write this definition, a proof of the `False` case must be provided. So, in lines 5-7, the proof `nil_not_nil` is completed. While both the subset type and indexed type need a proof of the `False` case, the proof for the former is with the constructors while the latter is with the size of the list. Moving forward, in lines 9-13, `front'` is defined to accept a subset type as input where `l` is a list and the predicate is that the list is not empty. If successful, the function returns the first element of the list. If this function is called on `Snil`, then Coq will give an error and computation will cease. An example of calling the `front'` function is done in line 19 with a list of length one that houses the value three. When creating the list, because the subset type is used, the `exist` constructor is needed to create an instance of the type. A proof that the list does not equal `Snil` is also needed. Combining the list instance and the proof, the computation, seen in line 19, evaluates to return 3.

```
Inductive list (X:Type) : Type :=
| Snil
| Scons (x : X) (l : list X).

Lemma nil_not_nil {X:Type} : Snil X <> Snil X -> False.
  intros H; destruct H; reflexivity.
Qed.

Definition front' {X:Type} (s: {l: list X | l <> Snil _}) : X :=
match s with
| exist (Snil) pf => match nil_not_nil pf with end
| exist (Scons n _ ) _  => n
end.

Lemma three : (Scons 3 (Snil _)) <> Snil _.
  unfold not.  intros H.  inversion H.
Qed.

Eval compute in front' (exist _ (Scons 3 (Snil _)) three).
(* = 3
: nat*)
```

Figure 2.5: Subset Type Example

Chapter 3

Negotiation

3.1 Motivation

The five guiding principles, as presented previously in Section 2.1, guide the development of remote attestation systems and must be enforced for optimal design. Principles like semantic explicitness and trustworthy mechanism are independent of the situation and hold because of sound implementation. Yet, other principles such as constrained disclosure and comprehensive information are situationally dependent and rely on the determination of relevant measurement parameters and additional constraints that arise from the context before they can be realized. The natural approach to capturing the situational demands is to develop a communication routine that gives the target and appraiser a chance to mutually conclude a measurement that protects the target and is adequate for the trust decision. This is subjective as the target's level of privacy may change with different appraisers.

The idea of gathering comprehensive information and maintaining constrained disclosure may be contradictory. When selecting a term for attestation, the target would like to keep as much information private as possible. They attempt to find the minimal evidence that is acceptable for the situation to preserve confidentiality. On the other hand, the appraiser would like the target to provide the most detailed and comprehensive evidence. The two parties must come to an agreement on the piece of evidence that would be sufficient for attestation. We aim to meet the needs of both the target and appraiser through *negotiation*.

3.2 Protocol Negotiation

Guided by the principles of remote attestation laid out by Coker et al. (2011), we introduce the idea of *negotiation* as a prerequisite to the existing attestation framework. The goal of negotiation is to generate a phrase for attestation that satisfies the appraiser's desire for comprehensive information and the target's need for constrained disclosure. This back-and-forth communication routine that would result in one term, agreed upon by the target and appraiser, to be employed for attestation.

We formally define the the negotiation process below in Figure 3.1. We define A as the appraisers and T as the targets. Q is the type of the initiating message of the security association and SA is the type of the instantiated security association. R is the type of attestation requests that defines the needs of the appraiser. P is the type of protocols represented as Copland phrases. E is the type of evidence generated by protocol execution.

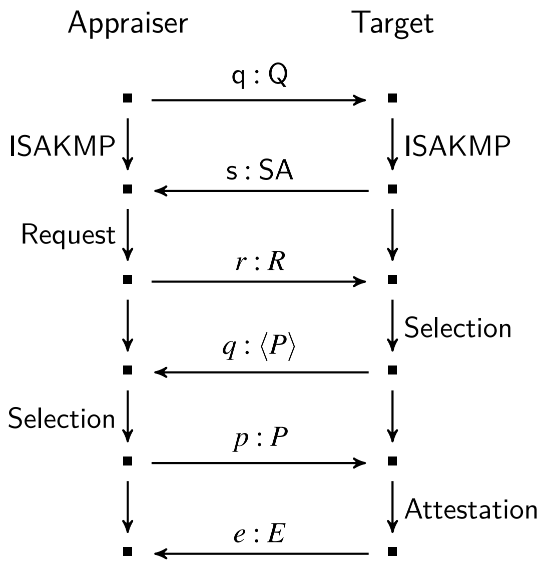

Figure 3.1: Negotiation process.

Negotiation begins with the establishment of a security association. This process is taken from ISAKMP and integrated to fit the needs of the situation to arrive at a controlled agreement. In essence, the appraiser initiates communication with $q : Q$ where ultimately the target responds with $s : SA$ to establish the security association. Intermediate steps of ISAKMP are omitted for clarity, as they are not a focus of this work, but it is assumed that ISAKMP runs correctly and generates the

SA. With it, the target and appraiser have a secure channel for subsequent communication, an idea of who they are communicating with, and an understanding of the security relevant parameters.

Following the establishment of the security association, the appraiser sends a request to the target. The request, $r : R$, illustrates the possible Copland phrase or phrases that the appraiser finds acceptable for attestation. The importance of the request is apparent when thinking about a target and appraiser's first encounter. In this case, the appraiser may be unaware of the target's measurement capabilities so they may include a variety of phrases in the request that entail measuring diverse aspects of the target's system. The negotiation framework then allows the target to decide what information they are willing to share. Inversely, the appraiser and target may have a well established relationship and the appriser could hold a cache of the negotiated phrase. In this case, the system could bypass negotiation and attestation would continue normally with the cached Copland phrase.

Once the target receives the request, selection is performed as seen with the downward arrow on the right in Figure 3.1. Selection entails the application of the *privacy policy* and *selection policy* to meet the principle of constrained disclosure. These policies are situationally dependent and realized upon establishment of the security association. To apply them, the appraiser evaluates the terms in the request to evidence. The policies are applied over the evidence to ensure the evidence does not expose sensitive information. With that, the target generates a list of acceptable terms for appraisal. This list of terms, $q : \langle P \rangle$, can be referred to as the proposal whose protocols fully or partially satisfy the request. The target chooses and orders $q : \langle P \rangle$ by examining evidence produced by executing protocols through Copland's evidence semantics. They then send $q : \langle P \rangle$ back to the appraiser for subsequent evaluations.

The appraiser receives the request, q, and must apply its own *selection policy* to achieve an ordering over terms. Although they share the same name, the realization of appraiser's selection policy is independent of the target's selection policy. The akin name is a result of selection's uniform structure but should not confuse the reader as the appraiser's selection process has different semantics. In the appraiser's context, selection implies and ordering over terms to arrive at the "best" term for attestation. A natural approach to quantifying the idea of "best" is to arrange the terms into a lattice where the "best" term is at the top of the lattice. The ordering operation would need to be situationally dependent and consider identity and timing requirements. While we have thought about how appraiser's selection policy would operate, we have yet to discern an ordering relationship between terms, leaving that to future work. Therefore, we have done minimal work implementing appraiser's selection policy.

Upon completion of the appraiser's selection policy, the communication routine resumes where attestation originally began. It is beyond the focus of this work to describe the steps of attestation but we do note, as seen in Figure 3.1, that one protocol is selected through the negotiation framework. This protocol is sent to the target where the target generates evidence, $e : E$, through attestation. The evidence is then returned to the appraiser as the final step in the diagram. This resulting evidence can be evaluated to make a trust decision.

It is important, when making trust decisions, that the mechanisms for gathering and evaluating measurements are trustworthy. In other words, we must verify that the semantics are correct. Before negotiation was introduced, a verification stack was developed to ensure a protocol evaluated to the correct evidence. It follows that the negotiation framework must be added to the verification stack to ensure the implementation aligns with the attestation framework and also produces a sound result. The verification stack can be seen in Figure 3.2 with the portion of negotiation added to the top as it occurs before attestation. The dotted arrows are mathematical relations while the solid arrows represent an evaluation rule. The portions of the stack that are in gray are beyond the scope of this work.

At the top of Figure 3.2, a *Request* becomes R which is mapped to an evidence lattice $(E, \preceq, \top, \bot)$ that produces the *Result*. The direct mathematical relation between requests and evidence lattices may be important to understand but left to future work. For now, we note that $\preceq$ is a partial order on evidence defining the selection policy, $\top$ defines the maxima, and $\bot$ defines the minima. Moving down from R, the request is evaluated, through negotiation, to obtain a proposal, $\langle P \rangle$ that produces corresponding elements of $\langle E \rangle$. $\langle E \rangle$ is the evidence allowed by the privacy and

$$\begin{tikzcd}
\text{Request} \arrow[r] & R \arrow[r, dashed] \arrow[d, "\text{Negotiation}"'] & (E, \preceq, \top, \bot) \arrow[r, dashed] & \text{Result} \\
& \langle P \rangle \arrow[r, dashed] \arrow[d, "\text{Selection}"'] & \langle E \rangle \arrow[u, "\text{Appraisal}"'] & \\
& \text{Protocol} \arrow[r, dashed] \arrow[d, "\text{copland\_compile}"'] & \text{Evidence Shape} \arrow[u, "\text{Abstraction}"'] & \\
& \text{CVM} \arrow[r, "\text{run\_cvm}"] \arrow[d, "\text{Synthesis}"'] & \text{Evidence} \arrow[u] & \\
& \text{CakeML/seL4} \arrow[r, dashed] \arrow[d, "\text{CakeML Compile}"'] & \text{Evidence} \arrow[u] & \\
& \text{Binary} \arrow[r] & \text{Evidence} \arrow[u] &
\end{tikzcd}$$

Figure 3.2: Verification stack.

selection policies and a sublattice of $(E, \preceq, \top, \bot)$. Moving down from $\langle P \rangle$, the appraiser applies their selection policy to obtain a single protocol for attestation. The selection policy is the same as the appraiser's selection policy discussed previously whereby the greatest element of $\langle P \rangle$ is $Protocol$. The protocol produces some known evidence shape where the evidence shape arises from the mapping of protocol to evidence. Abstracting the evidence shape results in one element in the evidence vector where other elements of the evidence vector are mapped from $\langle P \rangle$. The subsequent gray steps below the protocol are certified execution steps that are verified with respect to the Copland event semantics.

3.3 Policy

Two policies influence the production of a protocol, p, for attestation. The *privacy policy* is a collection of abstract goals enforced to protect a place's interest. It exists between either the target or appraiser and evidence. In some cases, the privacy policy implies the target has permission to send a protocol which can be evaluated to evidence. The privacy policy relation can be represented as $\pi_T : T \times E_T$. The appraiser may also employ a privacy policy $\pi_A : A \times E_A$ to prohibit sharing sensitive information in the request. The *selection policy* is a relation that maps concrete actions to abstract goals and can be represented with ψ. The privacy policy and selection policy are unique for each attestation system and can be conflated to a single policy.

In looking at the selection policy for the target and appraiser, they exhibit a similar structure as, in both cases, it is the realization of some property over terms. We know the appraiser's goal of negotiation is to find the "best" protocol for attestation. The realization of the ordering of protocols is their selection policy. In this case, ψ is a is a relation that maps the appraiser's desire for "best" to the concrete implementation of choosing the "best" protocol. It is left to future work to syntactically represent this relation. In the target's case, they must be cautious about what protocols they share in the proposal. Here, the *selection policy* maps the abstract goal of not violating the target's privacy standards to the concrete goals of not sharing certain protocols. This form of the selection policy can be observed as the relation $\{e : E_T | (t,e) \in \pi_T\}$ where evidence is filtered by the target's privacy policy.

We imagine many different attestation scenarios when constructing the privacy policy to ensure its flexibility for varying circumstances. First, we consider mutual attestation when an appraiser attempts to verify a target and then the communicating parties swap roles so that the appraiser becomes the target and the target becomes the appraiser. This situation forces both the target and the appriser to have privacy and selection policies that are dependent on their role and identity. This further emphasizes the need for a situationally dependent, flexible policy. Layered attestation is also a possibility where an appraiser requests evidence from a target and the target is an enterprise representing multiple machines. Here, only the target and appraiser would perform the steps of ISAKMP but the target would need to communicate with each machine in the enterprise to ensure no privacy policy is violated. It is also possible to have a cached negotiation whereby the appraiser would send the request and the target could quickly respond with the cached result. In this case, the privacy policy is only called during the initial communication routine before the results were cached. The target must ensure that its privacy policy has not changed between negotiations. If the privacy policy does change, the cache should be discarded. Observing these three negotiation scenarios emphasizes the importance of having a flexible, situationally dependent, privacy policy.

Chapter 4

Dependent Types for Policy

We aim to satisfy the target's goal of maintaining constrained disclosure by employing a dependently typed policy language to statically check if a term meets the requirements of the policy. We are not concerned with defining a correct policy, but rather capturing the policy behavior in a type dependent manner. Through the body of this section, the privacy and selection policies will be implemented in two flavors of dependent types: a subset type and indexed type. To reiterate, the target applies it's *privacy policy* to ensure the terms do not expose sensitive information and it's *selection policy* to realize terms for the proposal. Both the indexed type and the subset type accomplish the target's negotiation goals by statically selecting terms that satisfy the policies. By providing both implementations, we aim to disclose advantages and disadvantages of each way to decide which is favorable for negotiation.

The following structures, presented in Figure 4.1, are useful in both the subset type and indexed type implementations and discussed in this section for conciseness. First, the structure for `place` is presented to abstract measurement details such as identities and measurement locations. In traditional Copland, the idea of `place` is represented with a natural number that specifies a place where the measurement should occur. Yet, here it is an enumerated type to provide a greater level of abstraction for reasoning about policies. That is, there must be some way to capture measurement places, keys, and the identities as the policies are dependent on this information. We use the `place` grammar to abstract and reason about these situationally dependent parameters.

Also in Figure 4.1, the `class` grammar is presented as a means to realize the idea of constrained disclosure. That is, in implementation, each measurement will be given a label to classify information. The grammar has two options for labeling information: `red` and `green`. The `red` constructor

```
1           Inductive place : Type :=
2           | AA : place
3           | BB : place
4           | CC : place.
5
6           Inductive class : Type :=
7           | red : class
8           | green : class.
9
10          Definition eq_place_dec: forall x y:place, {x=y}+{x<>y}.
11          Proof. repeat decide equality. Defined.
12
13          Definition eq_class_dec: forall x y:class, {x=y}+{x<>y}.
14          Proof. repeat decide equality. Defined.
15
16          Definition good_encrypt := AA :: BB :: nil.
```

Figure 4.1: Common structures for negotiation

is used to state that the information is sensitive and should not be shared. The `green` constructor means that the term does not violate the privacy policy and its information can be shared without compromising the system. In practice, deciding if a blob is `red` or `green` will be a function of the situation and identities. Yet, an example of something that is always `red` is the private key because it should never leave its owner's machine. Inversely, the public key is exposed to everyone and would therefore be vacuously labeled `green`. However, asking the target for a measurement of arbitrary memory may be labeled `red` or `green`, depending on the context. In some cases, the target may want that portion of memory to remain private and so it would be labeled `red`. In other cases, the target may be willing to share that portion of memory so it changes the label to `green`.

A proof of decidability for places and classes is necessary for the privacy policy implementation. These proofs are completed in lines 10-14 of Figure 4.1. To prove two places or classes are equal or not equal, we say they are either a member of the set `{x=y}` or `{x<>y}`; they cannot be a member of both. We use the Coq proof tactic `decide equality` to solve these goals and guarantee set membership. The structure returns a proof of the places or classes equality or inequality that can be applied in the policy structure later.

The negotiation framework aligns closely with the Copland syntax shown in Figure 2.2 but it does not align exactly. First, the negotiation grammar support signing, sequence, and parallel operators for generating and ordering protocols in accordance with the Copland language. The `at` functionality is also supported to ask for measurements from specific places. The Copland syntax is embellished by defining a new constructor, called `Blob`, taken from the traditional computer science syntax meaning Binary Large OBject. The notion of a `Blob` is necessary to specify what component of the system should be measured where the blob is labeled `red` or `green` to distinguish sensitive information. The grammar is further embellished with an encryption operation. This allows the requestor to ask for some place to take a measurement and encrypt it with a specified key. In adding this operation, we must consider the case where a piece of information would be considered sensitive if it was decayable by a malicious intercepting party. We use a list to distinguish the places which the target finds acceptable to decrypt an encrypted measurement, that may be labeled `red`, with `good_encrypt`.

In the following sections, we use these structures to implement an indexed type and a subset type version of the target's privacy policy and selection function. We choose to utilize dependent types as we aim to statically meet the target's privacy requirements and cultivate a list of terms that are satisfactory for attestation. Both versions prove the same outcome but employ different means to generate the proposal making them interesting for compare and contrast.

4.1 Subset Type

The goal of representing the target's selection operation with subset types is to use sets and subsets to statically produce a subset of terms that meet the privacy policy criteria. In using subsets in Coq, each element of the set is a dependent pair where the first value is the term and the second is a proof that the term satisfies the privacy policy predicate. Starting with the set of all terms in the language, it is possible to define a subset where all terms in the subset satisfy the privacy policy predicate. Then, after the subset is created, if the user wants to use an element of the set for some other function, they must prove the predicate holds for the desired element. This solution bridges the gap between the mathematical version of subsets and type theory.

The subset type implementation of the target's selection operation begins by defining the `evidence` grammar. In Figure 4.2, the `evidence` structure is presented as type with seven constructors as there are seven operations in this language. The first constructor is `EBlob`, short for evidence blob, which tags the evidence type with its classification level. Again, the `red` label exposes sensitive information and `green` does not, thus accomplishing the goal of constrained disclosure. The next constructor is `EHash` which is short for evidence hash. This constructor does not have a class associated with it as any information that is hashed is safe to share and would be vacuously labeled `green`. `ECrypt` represents encryption and includes the evidence to encrypt as well as the place, or key, necessary for this operation. For encryption, keys are abstracted to `place` to simplify the syntax. `ESig` signs of a piece of evidence so it accepts a key for signing and evidence. Again, keys may be extracted to the place grammar. The `ESeq` and `EPar` constructors take in two pieces of evidence as a result of evaluating terms in sequence or parallel, respectively. Finally, the `EAt` constructor requests a measurement to be taken from a specific place. It requires the place where the measurement is collected as well as the evidence that was collected.

Given the `evidence` grammar, the `term` grammar is written in Figure 4.3 to map `evidence` to `Type`. This instantiates the relationship between terms and evidence. While the appraiser expects terms for attestation, privacy policy must operate over evidence to ensure no sensitive information is exposed. With this grammar, the relation between terms and evidence becomes apparent as each

```
Inductive evidence : Type :=
| EBlob : class -> evidence
| EHash : evidence
| ECrypt : evidence -> place -> evidence
| ESig : evidence -> place -> evidence
| ESeq : evidence -> evidence -> evidence
| EPar : evidence -> evidence -> evidence
| EAt : place -> evidence -> evidence.
```

Figure 4.2: Subset type evidence definition.

term is tagged with the evidence it produces. A constructor of interest is `THash` as it takes in a piece of evidence and returns the value `term EHash`. It disposes of the original evidence and is reduced to only the hash value. This is because hashing will never expose sensitive information so there is no concern the evidence will be visible to a malicious party. All other constructors expose the given evidence.

```
Inductive term : evidence -> Type :=
| TMeas : forall e, term e
| THash : forall e, term e -> term EHash
| TSig : forall e p, term e -> term (ESig e p)
| TCrypt : forall e p, term e -> term (ECrypt e p)
| TSeq : forall e f, term e -> term f -> term (ESeq e f)
| TPar : forall e f, term e -> term f -> term (EPar e f)
| TAt : forall e p, term e -> term (EAt p e).
```

Figure 4.3: Subset type term structure.

Using the grammars for `term` and `evidence`, the first minimal privacy policy is written, as seen in Figure 4.4, to meet the target's need for constrained disclosure. This privacy policy considers evidence as input and produces a proposition or a `True/False` claim to distinguish if the evidence exposes sensitive information. The prohibited cases, those that violate the privacy standards, evaluate to the proposition `False` making it impossible to provide a proof of their satisfaction for the subset type. In this first instantiation of a privacy policy, only the cases that violate the target's privacy policy, in all circumstances, are imposed. This forbids a measurement of any red blob and in some cases, the encryption operation, as we should refrain from sharing sensitive information the appraiser could decrypt. To ensure sensitive information is not decryptable by a malicious party, we hold the list of places that can receive decrypted information in the list `good_encrypt`. In implementation, as seen with line 8, the constructor for `ECrypt` states that if `ep`, the encryption key, is in `good_encrypt` then it is acceptable to share `red` information. If not, recurse on the evidence to ensure sensitive information is not exposed. The subsequent cases, `ESig`, `ESeq`, `EPar`, and `EAt` must recurse on the evidence to prevent exposing sensitive information.

```
1   Definition good_encrypt := AA :: BB :: nil.
2
3   Fixpoint privPolicy (e:evidence): Prop :=
4   match e with
5   | EHash => True
6   | EBlob red => False
7   | EBlob green => True
8   | ECrypt e' ep => if (in_dec (eq_place_dec) ep good_encrypt)
9                        then (match e' with
10                              | EBlob red => True
11                              | _ => privPolicy e'
12                            end)
13                         else privPolicy e'
14  | ESig e' _ => privPolicy e'
15  | ESeq l r => and (privPolicy l) (privPolicy r)
16  | EPar l r => and (privPolicy l) (privPolicy r)
17  | EAt p e' => privPolicy e'
18  end.
```

Figure 4.4: Subset type privacy policy definition.

Once the privacy policy is written, the selection function is implemented using the subset operations. The subset type is advantageous for two reasons. The first is that it is a static check which means it does not consume runtime resources. Secondly, each term in the subset type is a dependent pair which means the first element of the pair is the value and the second element is the proof that the value satisfies the privacy policy. This guarantees satisfaction of the privacy policy for terms in the proposal. In defining the selection function this way, as seen in Figure 4.5, we do not need to be concerned with providing a proof of a term's satisfaction until the selection function is applied.

```
Definition selectDep e (_:term e) := {t:term e | privPolicy e}.
```

Figure 4.5: Subset type selection function.

An example is presented in Figure 4.6 where measurement of a green blob is taken. We prove this term satisfies the privacy policy in the example `selectDep1` with a trivial proof. The Coq function, `proj1_sig`, is then used to retrieve the first element of the dependent pair, the value. If we wanted to return the proof, we could use the Coq function `proj2_sig`. After evaluating the phrase in line 6 of Figure 4.6 we obtain the result `term (EBlob green)` as expected. Lines 7 and 8 are comments in the code but are written to observe the return syntax that Coq produces after evaluating line 6. In practice, the program would statically observe that `selectDep1` satisfies the privacy policy and would dismiss the proof for subsequent computations.

```
1  Example selectDep1 : selectDep (TMeas (EBlob green)).
2  Proof.
3    unfold selectDep. exists (TMeas (EBlob green)). reflexivity.
4  Qed.
5
6  Check proj1_sig (selectDep1).
7  (* proj1_sig selectDep1
8        : term (EBlob green)*)
```

Figure 4.6: Subset type selection function example.

Ultimately, this type is a static way to ensure terms sent in the proposal satisfy the privacy policy. Even though it is successful, the implementation has its advantages and disadvantages. One can see that, when generating terms for the proposal in this type, one must have a proof for every term and remember the specific `proj1_sig` function in order to have access to only the term, not the accompanying proof. To make this easier, we could automate the proof by writing a generalized version that uses an existential variable for `exists` and then `auto` to solve the proof. This would be desirable as the proofs would not have to be altered even if the `evidence` or `term` grammar changes. This implementation may also be unfavorable as it requires a separate structure for selection, as is evident with `selectDep`. At the same time, the separation from the `term` syntax and the privacy policy makes it easier to write terms in the language. Overall, we remain optimistic about the subset type as we are able to accomplish the goal of statically enforcing the relationship between evidence and the privacy policy.

4.2 Indexed Type

Indexed types, like the subset types, are highly expressive forms of dependent types that can be used to provably ensure a program behaves in a certain way. In using indexed types, it is important to remember that the language design should not affect the standard programming operations or add additional constrains on the programmer (Fogarty et al., 2007). If used correctly, indexed types can be integrated seamlessly into the language and serve as a powerful way to statically enforce program properties. In the case of negotiation, an opportune place to apply indexed types is regarding the application of the privacy policy. That is, the `term` type can be constrained to include an index that calls the privacy policy to ensure the term does not expose sensitive information before the terms can be written.

The indexed type definition begins by writing an `evidence` structure that maps a `place` to a `Type`. The `place` input, as written in Figure 4.1, represents the place where the measurement occurs. This type differs from the subset type implementation of `evidence`, as seen in Figure 4.2, noting the absent `EAt` constructor. In the subset type case, evidence was a `Type`, without `place` as input, so in order to specify a place for measurement, the `EAt` constructor was needed. Here, because place is encoded in the type and each measurement couples the measurement's place and evidence, the `EAt` constructor can be omitted. We therefore have six constructors for the six operations in this language. Sans the `EAt` constructor, all constructors in the indexed type implementation accomplish the same measurements as they did in the subset type implementation.

```
Inductive evidence : place -> Type :=
| EBlob : forall p, class -> evidence p
| EHash : forall p, evidence p
| ECrypt : forall p q, evidence q -> place -> evidence p
| ESig : forall p q, evidence q -> place -> evidence p
| ESeq : forall p q r, evidence p -> evidence q -> evidence r
| EPar : forall p q r, evidence p -> evidence q -> evidence r
```

Figure 4.7: Indexed type evidence grammar.

The privacy policy operates over all cases in the `evidence` grammar. It accepts `e` where `e` is gathered from the target's place, `tp`, and returns the proposition `True` if the measurement does not expose sensitive information. As seen in the subset type implementation, a hash and a measurement of a `green` blob are always acceptable and do not expose sensitive information. A measurement of a `red` blob, where the information is accessible, is always prohibited and thus evaluates to `False`. The `ECrypt` constructor must take into consideration if `ep`, the encryption place, is in the list of acceptable place to encrypt with, `good_encrypt`. We use `ep` to abstractly represent the keys that can be used for encryption. It is possible appraiser's public key is not in the list `good_encrypt`. In that case, the target does not wish to share any information the appraiser could decrypt it with their private key. A check to ensure if `ep` is in the list of acceptable encryption keys, or places, must be completed before the function returns. If `ep` is not in the list `good_encrypt`, then the privacy policy recurses to ensure sensitive information is not encrypted. In the cases of `ESig, ESeq, EPar` we recurse on the arguments to ensure not sensitive information is exposed by sharing a `red` blob.

```
Definition good_encrypt := AA :: BB :: nil.

Fixpoint privPolicy tp (e:evidence tp): Prop :=
match e with
| EHash _ => True
| EBlob _ red => False
| EBlob _ green => True
| ESig _ e' _ => privPolicy e'
| ECrypt _ e' ep => if (in_dec (eq_place_dec) ep good_encrypt)
                        then (match e' with
                              | EBlob _  red => True
                              | _ => privPolicy e'
                        end)
                        else privPolicy e'
| ESeq _ l r => (privPolicy l) /\ (privPolicy r)
| EPar _ l r => (privPolicy l) /\ (privPolicy r)
end.
```

Figure 4.8: Indexed privacy policy definition.

The use of indexed types is realized in the `term` type. That is, in order to statically ensure that terms in the language are safe to share in the proposal, the privacy policy check can be encoded within the term grammar. This would imply that any time a term is written in the language, it must pass the privacy policy check before it is useful. This makes the outcome of the type dependent on the input and thus a dependent type. The encoding of the privacy policy in the `term` type can be observed in Figure 4.9 where the type accepts evidence as input and creates a Type. When writing a term, if the user only provides a piece of evidence, then the return value will include a lambda that requires a proof that the evidence satisfies the privacy policy. To return only the term, the term and proof must be simultaneously provided. Because this proof is impossible for the `False` cases, it will be impossible to write a useful term that does not satisfy the privacy policy. In examining the `term` constructors, all induce a privacy policy check sans the `THash` constructor. This is because a term that is hashed discloses no information so the hashing operation does not need a privacy policy check. However, all other operations may expose sensitive information prompting a privacy policy check for proof of satisfaction.

```
Inductive term p:(evidence p) -> Type :=
| TMeas : forall c,  privPolicy (EBlob p c) -> term (EBlob p c)
| THash : term (EHash p)
| TSig : forall ap e q,
    term e -> privPolicy ap (ESig p e q) -> term (ESig p e q)
| TCrypt : forall ap e q,
    term e -> privPolicy ap (ECrypt p e q) -> term (ECrypt p e q)
| TSeq : forall ap e f,
    term e -> privPolicy ap e -> term f -> privPolicy ap f -> term (ESeq p e f)
| TPar : forall ap e f,
    term e -> privPolicy ap e -> term f -> privPolicy ap f -> term (EPar p e f).
```

Figure 4.9: Indexed type term grammar.

To better understand the power of the indexed type, we present an example where the target measures a green blob, as seen in Figure 4.10. This mundane example is useful to understand the structure of this implementation as well as for comparison with the subset typed example presented in Figure 4.6. To begin, in line 1, the measurement for a green blob is written. This term states "measure a green blob at place AA." The return value of simply the measurement, without the proof, is a lambda that request a proof term to be instantiated before the term type can be written (lines 2 and 3). The proof is completed with `greenblob`. It is a short, simple proof that calls unfold and auto. This differs from the subset type proof which required a witness to the type within the proof. The complete measurement of the green blob is written in line 10 where the evidence and proof that the evidence satisfies the privacy policy are provided.

```
1   Compute TMeas AA green.
2   (*  = fun x : privPolicy (EBlob AA green) => TMeas AA green x
3      : privPolicy (EBlob AA green) -> term (EBlob AA green)*)
4
5   Lemma greenblob : forall p, privPolicy (EBlob p green).
6   Proof.
7       unfold privPolicy. auto.
8   Qed.
9
10  Compute TMeas AA green (greenblob AA).
11  (* term (EBlob AA green)*)
```

Figure 4.10: Indexed types example with measurement of a green blob.

In using the indexed type, we are able to ensure the privacy goals are met statically by constraining the type. Some believe that adding constraints to a type is favorable while other believe that the added systematic constructs confuse programmers and hinder the programs functionality. That is, in using the subset type where there are no systematic constraints to the term type, allows for separation between proofs and terms which may make it easier to reason about terms in the future. Yet, using the indexed type adds clarity and conciseness that may be missing from the subset type implementation. Specifically, there is no `selectDep` function that must be called to ensure a piece of evidence satisfies the privacy policy; it is called automatically when writing the term. This is favorable because we generate the same result, in both cases, but with fewer steps in the indexed type implementation. Furthermore, the proofs of privacy policy satisfaction are simpler in the indexed type making them easier to reuse and reason about. These benefits of the indexed type may be cause to use its implementation going forward, but the decision has yet to be ultimately made.

Chapter 5

Example

In traditional Copland, an attestation protocol is selected by the appraiser, and the target is asked to evaluate the protocol to evidence. Without the negotiation infrastructure, the target has no language in which they can provide useful feedback to communicate what phrase or phrases they would find acceptable for attestation. In this section, a complete example is presented to reenforce the necessity of the negotiation framework to meet the target's goal of constrained disclosure. In their paper, Petz (2020) present examples of Copland phrases that are useful to establish trust in remote machines. They begin with a simple example where a measurement of a virus checker is executed to gain an understanding of a target's state and then embellish the measurement to provide addition details. The four possible measurements we mentioned in Figure 2.3 are summarized with the list below. Each useful for proving a different level of information, we assume that these measurements are joined together and sent in the `request`.

1. Take a measurement of the virus checker.
2. Take a measurement of the virus checker and sign it.
3. Take a measurement of the signature server, sign it, and take a measurement of the virus checker and sign it.
4. Take a measurement of the target's operational environment, sign it, and take a measurement of the virus checker and sign it.

Figure 5.1: Measurements in the request.

Before dissecting and responding to the request, it is assumed that ISAKMP has run to completion, allowing the target and appraiser to develop a security association (SA) that provides a secure channel for subsequent communications while defining identities and a common vocabulary. The common vocabulary is established as result of the domain of interpretation which allows objects to be named similarly in both the target and appraiser's contexts. Futhermore, the SA defines a situation that may communicate the urgency of the response. For the purposes of this example, it is assumed that the target forbids sharing a measurement of its operational environment as this may reveal what version of the operating system the target is running, which could leave their system vulnerable to attacks. This privacy goal is realized after the establishment of the SA as it is a function of this specific negotiation routine between target and appraiser.

In order to adequately represent negotiation, we must remove a level of abstraction from the `class` type. Previously, a `class` type was implemented to label measurement objects as `red` or `green` where the former suggests the measurement violates the privacy policy and the latter suggests it does not. In the context of a functioning negotiation routine, the `red/green` labeling will be predetermined by the SA. Therefore, it is redundant to allow an object to be labeled as `red/green` as it is either shareable or private in the context of a specific situation and the measurement object is realized as such. It follows that the `red/green` labeling should be replaced with the specific measurement objects to be reasoned about in the privacy policy.

```
Inductive class : Type :=
| VC : class
| SS : class
| OP : class.

Inductive place : Type :=
| target : place
| appraiser : place
| t_pub_key : place
| t_priv_key : place
| a_pub_key : place.
```

Figure 5.2: Common structures for this example.

As seen in Figure 5.2, the `class` grammar includes `VC`, `SS`, and `OP` which represents the abstraction of the virus checker, signature server, and operational environments, respectively. For signing and encryption, keys are abstracted to the place grammar and are represented with `t_pub_key`, `t_priv_key`, and `a_pub_key` to denote the target's public key, the target's private key, and the appraiser's public key respectively. The target and the appraiser's places are represented abstractly with `target` and `appraiser` in the place grammar.

```
1     Definition good_encrypt := t_pub_key :: nil.
2
3     Fixpoint privPolicy (e:evidence): Prop :=
4     match e with
5     | EHash => True
6     | EBlob OP => False
7     | EBlob _ => True
8     | ECrypt e' ep => if (in_dec (eq_place_dec) ep good_encrypt)
9                       then (match e' with
10                            | EBlob red => True
11                            | _ => privPolicy e'
12                            end)
13                      else privPolicy e'
14    | ESig e' p => privPolicy e'
15    | ESeq l r => and (privPolicy l) (privPolicy r)
16    | EPar l r => and (privPolicy l) (privPolicy r)
17    | EAt p e' => privPolicy  e'
18    end.
19
20    Definition selectDep e (_ :term e) := {t:term e | privPolicy e}.
```

Figure 5.3: An example privacy policy written for the subset type definition.

The grammars for `terms` and `evidence` are the same as written in Chapter 4, but the privacy policy cannot be realized until the establishment of the SA as it is parameterized over the situation. The realization of the privacy policy for the subset type implementation can be visualized in Figure 5.3. The list `good_encrypt` recognizes the keys the target is willing to use for encryption with `t_priv_key` and `a_pub_key` omitted to ensure the target does not encrypt any sensitive information that could be decryptable. As we assumed previously, the target refuses to send a measurement of their operational environment so a measurement requested of the blob `OP` will return `False`. With the privacy policy defined, the selection function is written to be a subset of all terms that satisfy the privacy policy, as seen in line 20.

The first measurement for this example is written in Figure 5.4, specifically in line 2. Recall this is a measurement of the virus checker explicitly where the request is that place `target` measures the blob `vc`. With the `Compute` function in line 4, we can observe and understand the effect of calling the selection function on the term. Explicitly stated in line 5, the `Compute` returns a dependent pair where the first element is a term in the language and the second requires a proof that it satisfies the predicate. While this is expected for the type, this subset holds no value for subsequent computations because the term has not been proven `True`. The proof of the `True` case is implemented in `vc_okay` using the `exists` proof tactic to provide a witness to the claim where the witness must be the term we are trying to prove `True`. To complete the proof, the `unfold`, `exists`, and `auto` tactics are called. With both the term and the proof, the example `vc_okay` uses these to create a dependent pair with both values now accessible. In order to generate only the term, the `proj1_sig` function is called in line 15, returning the first element of the dependent pair.

```
1   (* Measure the VC *)
2   Definition vc := TMeas (EAt target (EBlob VC)).
3
4   Compute selectDep _ vc.
5   (*= {_ : term (EAt target (EBlob VC)) | True}
6   : Set *)
7
8   Example vc_okay : selectDep _ vc.
9   Proof.
10      unfold selectDep.
11      exists (TMeas (EAt target (EBlob VC))).
12      unfold privPolicy. auto.
13  Qed.
14
15  Check proj1_sig (vc_okay).
16  (* : term (EAt target (EBlob VC)) *)
```

Figure 5.4: An example of measuring the virus checker with the indexed type.

Presented in Figure 5.5, the pattern above is continued for the subsequent terms in the request. These terms, terms 2, 3, and 4 in Figure 5.1 are encoded with `vc_sign`, `vc_ss`, and `vc_op` respectively. It is evident that all proofs of the `True` cases follow the same structure. That is, call unfold, find a witness, and use auto to prove True. While it is possible to write these three examples, it is not possible to prove them all. The first two, vc_sign and vc_ss, are written and proved with vc_sign_okay and vc_ss_okay, respectively. However, vc_op violates the privacy policy as it evaluates to False, yet it is still possible to write vc_op as it is properly typed. In this way, it is possible to reason about terms without the added privacy policy constraint. With only the term, an attempt to do the proof is presented with vc_op_okay which must be Aborted because there is no possible way to prove the False case. The proposal can then be formed to include the first value in the dependent pair, or the term, taken from vc_okay, vc_sign_okay, and vc_ss_okay.

In the subset typed language, the function selectDep is the realization of the selection policy. The only way to generate a term is to call selectDep and then prove the term satisfies the privacy policy. Yet, in the indexed typed language, a term cannot be written unless it satisfies the privacy policy. There is no separation between terms and the privacy policy so there need not be a selection function as the selection policy is captured in the term definition. This will impact the term's verbiage but does not affect the success of privacy policy's installment to meet the target's need for constrained disclosure. As seen below with Figure 5.6 the privacy policy for the indexed type example looks almost the same as the subset type's implementation barring the inclusion of place in the input to the indexed type. The constructor EBlob p OP still evaluates to False realizing that a measurement of the operational environment exposes sensitive information.

The differences between the indexed type and subset type are easier to visualize with the following examples. Using the indexed language, the first task is to write a term that represents the minimal measurement of the virus checker. This is written in the example vc in Figure 5.7. Because of the nature of the term type, vc has the precondition that it satisfies the privacy policy. So, in order to generate only the term, a proof must be provided to validate the term's satisfaction of the privacy policy. The proof of satisfaction, vc_proof, is accomplished using the unfold and auto techniques. Unlike the subset type cases, this proof does not need a witness to be complete. Producing only the term is achieved in the definition vc_okay.

The other terms requested, `vc_sig` and `ss_sig`, are proven satisfactory with the lemmas `vc_sig_proof` and `ss_sig_proof`. Noting that they are all provable with the same structure, the proof is succeeds by first unfolding the definition of the privacy policy and then calling `auto`. By the indexed typed definition, when composing the terms in sequence, the structure requires that each of terms in the sequence statement have its own accompanying proof. There is no new proof strategy necessary, as is evident with the definition of `vc_ss_par`. We remember this fact when attempting to prove the sequence measurement of the virus checker and the operational environment. That is, we know the strucutre of the proof but it is impossible to complete because the term evaluates to `False`. Because we cannot write the proof that the operational environment satisfies the privacy policy, it is impossible to write the term that generates the `OP` measurement. An attempt is failed, as expected, with the lemma `op_proof` as we cannot provide a complete proof. Therefore, the goal of encoding the privacy policy and meeting the target's need for constrained disclosure has been accomplished during typechecking.

Through these examples, we have shown that the privacy policy can be realized in both the subset type and indexed type definitions by prohibiting the measurement of the operational environment to be written or executed. When attempting to generate a term for the proposal, in both types, the term must be proven `True`. While the realization of this is generated differently, the proofs of the `True` cases are similar. Perhaps the indexed type is more desireable as the proofs do not require a witness and it may be more concise to write a term. Futhermore, the absence of the `selectDep` function in the indexed type makes the overall structure tellingly breviloquent. Yet the subset type should not be dismissed as it separates terms in the language from their filtering requirements. This may be useful if a target and appraiser are preforming attestation for the first time and the appraiser does not understand the target's structure. In this case, they may be willing to accept any term and the target would only need ensure satisfaction of their privacy policy. Each implementation is beneficial in unique ways and we leave it to future work to elect a type for negotiation.

```
Definition vc_sign := TMeas (ESig (EAt target (EBlob VC)) t_priv_key).

Definition vc_ss := TMeas (ESeq (ESig (EAt target (EBlob SS)) t_priv_key)
                                (ESig (EAt target (EBlob VC)) t_priv_key)).


Definition vc_op := TMeas (ESeq (ESig (EAt target (EBlob OP)) t_priv_key)
                                (ESig (EAt target (EBlob VC)) t_priv_key)).

Example vc_sign_okay : selectDep _ vc_sign.
Proof.
    unfold selectDep.
    exists (TMeas (ESig (EAt target (EBlob VC)) t_priv_key)).
    unfold privPolicy. auto.
Qed.

Example vc_ss_okay : selectDep _ vc_ss.
Proof.
    unfold selectDep.
    exists (TMeas (ESeq (ESig (EAt target (EBlob SS)) t_priv_key)
                        (ESig (EAt target (EBlob VC)) t_priv_key))).
    unfold privPolicy. split. auto. auto.
Qed.

Example vc_op_okay : selectDep _ vc_op.
Proof.
    unfold selectDep.
    exists (TMeas (ESeq (ESig (EAt target (EBlob OP)) t_priv_key)
                        (ESig (EAt target (EBlob VC)) t_priv_key))).
    unfold privPolicy. split. auto.
Abort.
```

Figure 5.5: An example of measurements 2,3, and 4, written in the subset type, which evaluate to terms once the proof is completed.

```
Fixpoint privPolicy tp (e:evidence tp): Prop :=
match e with
| EHash _ => True
| EBlob p OP =>  False
| EBlob p _ => True
| ESig _ e' _ => privPolicy e'
| ECrypt rp e' tp => if (in_dec (eq_place_dec) tp good_encrypt)
                     then (match e' with
                                | EBlob t_priv_key red => False
                                | EBlob _ red => True
                                | _ => privPolicy e'
                           end)
                     else privPolicy e'
| ESeq _ l r => (privPolicy l) /\ (privPolicy r)
| EPar _ l r => (privPolicy l) /\ (privPolicy r)
end.
```

Figure 5.6: Privacy policy example written for the indexed type.

```
Lemma vc_proof : privPolicy (EBlob target VC).
Proof. unfold privPolicy. auto. Qed.

(*measure the virus checker*)
Definition vc := TMeas target VC.
(*: privPolicy (EBlob VC green) -> term (EBlob VC green) *)
Definition vc_okay := TMeas target VC vc_proof.

Lemma vc_sig_proof : privPolicy (ESig target (EBlob target VC) t_priv_key).
Proof. unfold privPolicy. auto. Qed.

(*sign the virus checker measurement*)
Definition vc_sig := ESig target (EBlob target VC) t_priv_key.
Definition vc_sig_okay := TSig t_priv_key (TMeas target VC vc_proof) vc_sig_proof.

Lemma ss_sig_proof : privPolicy (ESig target (EBlob target SS) t_priv_key).
Proof. unfold privPolicy. auto. Qed.

Definition ss_sig := ESig target (EBlob target SS) t_priv_key.
Definition ss_sig_okay := TSig t_priv_key (TMeas target SS vc_proof) ss_sig_proof.

(*measure and sign the virus checker and the signature server in parallel *)
Definition vc_ss := TSeq (ss_sig_okay) (ss_sig_proof)
                         (vc_sig_okay) (vc_sig_proof).

Lemma op_proof : privPolicy (EBlob target OP).
Proof. unfold privPolicy. Abort.
```

Figure 5.7: An example of measurements 2,3, and 4, written in the indexed type, which evaluate to terms once the proof is completed.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Remote attestation is a framework used to instill trust in communicating parties across a network. To make a trust decision, the appraiser first sends a protocol to the target where the target evaluates the protocol to evidence and returns the generated evidence to the appraiser. Then, once the appraiser receives the evidence, they can evaluate it to make an informed trust decision. Previously, the measurement sent to the target was chosen statically, giving the target no choice to suggest a protocol and possibly forcing them to expose sensitive information. To address this problem, we developed a negotiation framework, situated before the attestation routine, that allows the target and appraiser to decide on a protocol or protocols for attestation. Through the framework, the target a is given a chance to enforce their privacy standards thus meeting the goal of constrained disclosure. After the protocol is selected and negotiation is complete, attestation runs as intended to produce a trust decision.

We introduced the *privacy policy* and *selection policy* as part of the target's selection process. The target's privacy policy is a mathematical relation that maps evidence to a binary value stating that either the evidence exposes sensitive information or it does not. The selection policy is a means of mapping abstract goals to concrete actions. In the target's case, the selection policy is used to meet the goal of not sharing sensitive information. The focus of this work is not to write sound policies, but rather to understand the shape and structure of the polices to best capture them within the negotiation framework.

In implementing these policies, we use dependent types, specifically the subset type and indexed type, to statically ensure that terms shared with the appraiser satisfy the privacy requirements and do not expose sensitive information. The expressive nature of dependent types is essential to integrating the negotiation procedure with the specification that the target does not share sensitive information. In using a subset type, we are able to apply a predicate to terms in the language such that the terms must satisfy the privacy policy to be an element of the subset type. For the indexed type implementation, the privacy policy constraint is added to the term grammar ensuring every term written in the language passes the privacy policy check. In both cases, a proof of satisfaction of the policy is necessary before the term can be used in subsequent computations. We emphasize that both forms of dependent types provide a static check so as not to consume any runtime resources.

Selecting the correct term for attestation is an important facet of the attestation problem that we aim to understand. The bigger problem, however, is determining trust in remote machines across a network to ensure everyday users are not vulnerable to attacks. While there are portions of the negotiation framework that must be developed before it can be integrated to help make the trust decision, we have made progress in setting up and understanding the problem and attempting to capture the privacy standards of the communicating parties. With that, the hope is the future of computing will include the attestation work, that utilizes negotiation, to indisputably determine trust in remote machines.

6.2 Future Work

While we have made significant progress in understanding the framework of negotiation, we must continue to work on understanding all the mathematical relations surrounding the negotiation structure. First, the shape of the request must be formalized with a proper representation. We expect that it is a list of protocols the appraiser finds acceptable for attestation. Yet, if the appraiser is unaware of the target's structure, we are unsure how they would know what terms to use in the request. Next, there is more work to be done to realize an ordering over protocols. Ideally, the shape of the protocol would suggest the arrangement of the lattice whereby the appraiser is able to select the best term for attestation from the top of the lattice. Currently, we lack an understanding of the shape of protocols and have yet to realize and instantiate an ordering. Once the mathematical relations are brought to fruition, we must represent ISAKMP within its own grammar to be useful within the negotiation structure. After this, a complete structure for negotiation would arise. It would then be possible to generate a protocol for attestation based on the request in one fluid exercise.

References

Cardelli, L. (1996). Type systems. *ACM Comput. Surv.*, 28(1), 263–264.

Chlipala (2013). *Certified Programming with Dependent Types: A Pragmatic Introduction to the Coq Proof Assistant*. The MIT Press. MIT Press.

Coker, G., Guttman, J., Loscocco, P., Herzog, A., Millen, J., O'Hanlon, B., Ramsdell, J., Segall, A., Sheehy, J., & Sniffen, B. (2011). Principles of remote attestation. *Int. J. Inf. Secur.*, 10(2), 63–81.

Coker, G., Guttman, J., Loscocco, P., Sheehy, J., & Sniffen, B. (2008). Attestation: Evidence and trust. In L. Chen, M. D. Ryan, & G. Wang (Eds.), *Information and Communications Security* (pp. 1–18). Berlin, Heidelberg: Springer Berlin Heidelberg.

Coq development team (2016). *Coq Reference Manual*. INRIA. Version 8.5pl1.

Fogarty, S., Pasalic, E., Siek, J., & Taha, W. (2007). Concoqtion: Indexed types now!

Hajjeh, I., Serhrouchni, A., & Tastet, F. (2003). Isakmp handshake for ssl/tls. In *GLOBECOM '03. IEEE Global Telecommunications Conference (IEEE Cat. No.03CH37489)*, volume 3 (pp. 1481–1485 vol.3).

Hecker, A. (2002). *Internet Security Association and Key Management Protocol*. Technical report, ENST Paris.

Loscocco, P., Smalley, S. D., Muckelbauer, P. A., Taylor, R., Turner, S., & Farrell, J. (2000). The inevitability of failure: The flawed assumption of security in modern computing environments.

Maughan, D., Schertler, M., & Turner, J. (1998). *Internet Security Association and Key Management Protocol (ISAKMP)*. RFC 2408, RFC Editor.

McBride, C. (2002). Faking it: Simulating dependent types in haskell. *J. Funct. Program.*, 12, 375–392.

Pendergrass, J., Helble, S., Clemens, J., & Loscocco, P. (2017). Maat: A platform service for measurement and attestation.

Petz, A. (2020). An infrastructure for faithful execution of remote attestation protocols. In *Proceedings of the 7th Symposium on Hot Topics in the Science of Security*, HotSoS '20 New York, NY, USA: Association for Computing Machinery.

Qing, X. & Adams, C. (2006). Keaml - key exchange and authentication markup language. In *2006 Canadian Conference on Electrical and Computer Engineering* (pp. 634–638).

Ramalingam, R. (2017). Internet security association and key management protocol (isakmp).

Ramsdell, J., Rowe, P., Alexander, P., Helble, S., Loscocco, P., Pendergrass, J., & Petz, A. (2019). Orchestrating layered attestations. (pp. 197–221).

Sozeau, M. (2007). Subset coercions in coq. In T. Altenkirch & C. McBride (Eds.), *Types for Proofs and Programs* (pp. 237–252). Berlin, Heidelberg: Springer Berlin Heidelberg.

Xi, H. & Pfenning, F. (1999). Dependent types in practical programming. In *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '99 (pp. 214–227). New York, NY, USA: Association for Computing Machinery.

Xiu, D. & Liu, Z. (2005). A formal definition for trust in distributed systems. In J. Zhou, J. Lopez, R. H. Deng, & F. Bao (Eds.), *Information Security* (pp. 482–489). Berlin, Heidelberg: Springer Berlin Heidelberg.

Zenger, C. (1997). Indexed types. *Theor. Comput. Sci.*, 187(1–2), 147–165.