Generalized Certificate Revocation

Carl A. Gunter
University of Pennsylvania

Trevor Jim
AT&T Research

Abstract

We introduce a language for creating and manipulating *certificates*, that is, digitally signed data based on public key cryptography, and a system for *revoking* certificates. Our approach provides a uniform mechanism for secure distribution of pubic key bindings, authorizations, and revocation information. An external language for the description of these and other forms of data is compiled into an intermediate language with a well-defined denotational and operational semantics. The internal language is used to carry out consistency checks for security, and optimizations for efficiency. Our primary contribution is a technique for treating revocation data *dually* to other sorts of information using a polarity discipline in the intermediate language.

1 Introduction

Public Key Infrastructures (PKI's) have received considerable attention in the last decade in the hope that they can form a foundation for secure electronic commerce. We have developed a programming language, QCM, that can be used as the basis of a general and semantically sound PKI [10]. In this paper we extend QCM to support *certificate revocation.*

A certificate is a digitally signed document. PKI's use certificates to securely distribute data including key bindings ('Alice's key is p') and authorizations ('Alice may use the printer'). Because the data in a certificate can become invalid, most PKI's support some mechanism for revoking certificates; for example, in X.509 [14], the serial numbers of no-longer-valid certificates are periodically published on Certificate Revocation Lists (CRL's). However, certificate revocation is controversial: some researchers have pointed out that its semantics is not well understood [9, 27, 28], and others have proposed eliminating it entirely [25].

We believe that part of the confusion regarding revocation and PKI's stems from treating revocation data specially. That is, rather than treating revocation data as just another kind of data to be distributed, PKI's historically have had ad hoc mechanisms for managing CRL's, certificates containing key bindings, and certificates containing authorizations. Consequently, the semantics and distribution of the different kinds of data must be understood separately, rather than as instances of a general concept.

In this paper, we propose an alternative, general framework that can be used to uniformly and securely distribute data of all sorts, including public key bindings, authorizations, and revocation information. The work builds on our Query Certificate Manager (QCM) system [10], a programming language that already treats public key bindings and authorizations uniformly. The key observation we make is that data used for revocation should be treated *dually* to other sorts of information: for example, to determine that Alice's key is K, it is necessary to determine that (Alice,K) *is* one of the bindings issued by a certificate authority, and *is not* a binding revoked by the authority. By viewing the two kinds of data dually, and developing a *polarity discipline* for QCM programs that distinguishes them, we are able to use the same mechanisms to distribute both kinds of data securely.

The main accomplishment of the paper is to define a general and semantically sound PKI that supports revocation and that can be feasibly implementated. This is demonstrated using a denotational model to define correctness, an operational semantics that is a simplified version of our distributed implementation, and a polarity discipline that ensures that the operational semantics is sound with respect to the model.

From a programming language perspective, some of the novelties of the system are that QCM is designed to work with partial information—QCM does not calculate the exact answer for a program, only a best approximation—and that in order to handle revocation, the QCM implementation manipulates representations of infinite sets.

From a security perspective, our contributions are to provide a semantics of revocation, and to show that revocation information can be distributed by the same mechanisms used to deliver key binding and authorization data.

The paper is organized as follows. In the next section we explain why certificate revocation is both logically and pragmatically problematic, in the context of related work. In Section 3, we introduce an extension of QCM with nonmembership tests. Non-membership tests are a restricted form of negation that enable revocation to be programmed directly in QCM. In Section 4, we present our polarity discipline and show that polarities can be used to make QCM with revocation monotonic. In Section 5, we give an operational semantics for QCM with revocation and show that it is sound, using monotonicity. Section 6 describes our external language, and Section 7 concludes. Proofs are sketched in an Appendix.

2 Certificates and Revocation

We will refer to any data with a digital signature as a *certificate.* Certificates are tamper-evident (modifying the data makes the signature invalid) and unforgeable (only the holder of the secret, signing key can produce the signature). These properties make certificates useful in conducting secure electronic transactions.

Many different kinds of data can be signed. For example, a certificate may represent a binding (e.g., 'p is the public key of Alice'), or it may indicate a permission (e.g., 'Alice has permission to use the swimming pool'). In any case, the signer of the certificate, who is known as the *issuing party*, may wish to indicate a term of validity for the certificate. For instance, the certificate giving Alice permission to use the pool could be marked valid until the end of the academic year. The *relying party* who examines a certificate must take this validity period into account: when deciding whether to admit Alice to the swimming pool, the relying party should consider the certificate invalid if the academic year is over, and deny access.

Revocation is used to invalidate a certificate prematurely, before the end of its validity period. Revocation might be needed due to key compromise (e.g., the signing key is stolen), change of affiliation (e.g., Alice drops out of school), or many other reasons.

Certificates and revocation are core mechanisms in Public Key Infrastructures (PKI's). For example, the best known PKI is the ISO Directory, based on the X.500 series of standards [13]. One of these standards, X.509 [14], proposes a hierarchy of *Certificate Authorities (CA's)* which produce *public key certificates* binding principals (like people, devices, or entities) to public keys. In recognition of the practicalities of large systems, X.509 provides a mechanism for dealing with certificates that become too old and must therefore be considered expired, as well as certificates that, for some reason, need to be revoked ahead of their expiration date. The latter are announced on *Certificate Revocation Lists (CRL's)* which are signed by certificate authorities (CRL's are certificates too).

A typical scenario under the X.509 standard would proceed as follows.

- The issuing party provides a certificate with a period during which it is to be considered valid. A serial number included in the certificate uniquely identifies it.
- The issuing party provides a CRL, which is a certificate that contains serial numbers of revoked certificates. The CRL also has a time issued and a time for the next CRL to be issued, so it is possible to determine whether the CRL is current.
- To validate a certificate, the relying party checks the correctness of its signature, checks that the certificate has not expired, and checks to see if the serial number of the certificate is listed on the current CRL of the issuing party. The signature of the CRL itself must also be checked.

The most recent X.509 standard provides for some variations on this protocol, such as *indirect CRL's* that are signed not by the issuing party, but by a third party designated by the issuing party.

Attribute certificates are yet another kind of certificate, used not to bind keys to identities, but rather to use keys for a range of objectives requiring authentication and/or secrecy. For instance, the IETF Working Group, PKIX [29, 12], focuses on developing a system based on CA's that can support security for network directory services (like DNS), electronic mail (*viz.* extensions of PEM [16]), and web pages. Indeed, the orginal aim of the certificate authorities was to support *authorization* for the Directory [13, 14]. A directory system like the Lightweight Directory Access Protocol (LDAP) [30] can use the certificate authorities to support control over which principals can access or alter the Directory. Attribute certificates can be used to bind a subject to a set of permissions, thus supporting authorization. How a system should manage attribute certificates is not well understood. For instance, Warwick Ford, the co-chairman of PKIX, notes in [8]:

> Attribute certificates represent an important area of electronic commerce technology which is yet to be fully explored or developed. (page 253)
>
> Certified distribution of authorization information is an active development area and. . . the 'best' approach is far from clear. (page 256)

Support for this critical function has been the subject of recent research on *policy verification engines* that support the use of certificates representing authorization and identity information to state and verify security policies [2, 6, 7, 10]. Although the X.509 system supports revocation for attribute certificates just as it does for public key certificates, none of these new verification engines directly provide such support.

In the remainder of this section, we discuss four aspects of revocation in more detail: (1) the semantics of revocation; (2) the means for distributing revocation state; (3) whether another mechanism for achieving similar goals would work better; and (4) tradeoffs between the risks, costs, and liabilities of revocation.

2.1 What Does Revocation Mean?

Consider a certificate,

'p is the public key of Alice' (signed q).

Suppose this certificate has serial number n, and n appears on a CRL. What does this mean? Fox and LaMacchia point out at least three possible interpretations [9]:

1. p is not the public key of Alice;
2. q can no longer vouch for whether p is the public key of Alice; or
3. the signing key of q has been compromised.

The relying party should act quite differently depending on what interpretation it takes. For example, suppose we have two additional certificates:

A. 'p is the public key of Alice' (signed r).

B. 'r is the public key of Bob' (signed q).

If we take interpretation 1, then we should discard A, but we may still continue to trust B. On the other hand, if 2 holds then we may choose to trust both A and B. Finally, if 3 holds, then we may trust A but not B.

Clearly, a PKI must unambiguously specify how revocation should be interpreted: otherwise, we might wrongly believe that Alice's key is p (a security breach), or we might wrongly disregard Alice's correct key p (denial of service). That is, a PKI should have a semantic model for revocation. A model is crucial for preventing other kinds of security breaches, as well. For example, the following scenario was possible in SDSI 1.1 [26]:

$$\begin{aligned}&\mathit{school} = \mathit{teachers} \cup \mathit{admin} \cup \mathit{students},\\ &\mathit{employees} = \mathit{school} - \mathit{students}.\end{aligned}$$

This defines two groups, *school* and *employees*, in terms of some other groups, including *students*. Alice could be issued a student id, that is, a certificate 'Alice $\in$ *students*', good for the entire school year. If Alice drops out during the year, the administration could revoke her privileges by issuing a new certificate, 'Alice $\notin$ *students*.' Now, if Alice gets her hands on both certificates, she can prove that she is an employee: since Alice $\in$ *students*, by the first definition we have Alice $\in$ *school*. And then since Alice $\notin$ *students*, we have Alice $\in$ *employees*, even though Alice was never a teacher or administrator. The problem, of course, is that the two certificates are contradictory: they do not have a model.

This kind of security breach cannot occur in QCM, because it was designed to include restrictions that guarantee that such contradictions can never occur, while still permitting revocation to be expressed. We prove this formally using a semantics and a polarity discipline.

The semantics of revocation has also been studied by Stubblebine [27] and Stubblebine and Wright [28], who extended BAN logic [5] with time intervals and a form of revocation. Their logics are not implemented, however.

2.2 Distributing Revocation State

There is a great deal of research on ways of distributing the revocation state of certificates; see Myers [22] for a classification. Most of the work involves variations on CRL's. In the first version of X.509, CRL's were provided monolithically via a list of serial numbers for a given CA.[1] The monolithic approach has a couple of disadvantages. First, as the number of certificates grows, so does the number of revoked certificates, which can greatly increase the size of the CRL's. Second, in applications where a high percentage of certificates will be revoked, the CRL's must be distributed more often to cut down the "window of vulnerability" between the time that a certificate is revoked and the time that it appears on the CRL. This places significant burdens on the CRL server and the network. Much of the work on CRL's has been aimed at addressing these issues, and we sketch only enough of the story to try to convince the reader that the distribution and retrieval of CRL's is a non-trivial issue and remains an open problem.

A first observation is that CRL's can be divided according to who might be interested in them. For instance, a single CRL *distribution point* could be partitioned into a family of CRL's representing different reasons for revocation, possibly issued at different rates. This might allow a short list of compromised keys to be issued with a short validity period while a longer list of some lower-priority revocations is updated less frequently. It also facilitates 'on demand' retrieval of the CRL. For instance, only the CRL for people with names in a certain range may need to be retrieved to check a given certificate.

A second observation is that most of the entries in a CRL remain the same as CRL's are updated, so it is possible to issue a *delta CRL* indicating only the changes. Of course, this means that checking a certificate entails having all of the deltas from the last full release. This also opens the possibility of having relying parties choose different strategies for validation. For instance, a low-risk decision might not require checking the delta CRL's.

A third observation is that multiple CA's could use a single CRL distribution point, maintained by a third party. These are the indirect CRL's we mentioned earlier.

A fourth observation is that there are a number of ways to obtain CRL's. While a relying party may choose to query a distribution point, it is equally possible for a distribution point to 'push' a CRL to a collection of potential relying parties. If properly done, this could enable efficient distribution of CRL's (or other forms of revocation state) via unreliable network transport using UDP unicast or multicast [15, 19, 21].

Alternatives to CRL's include the work of Micali [20] and Aiello et al. [1], the tree-based schemes of Naor and Nissim [24] and Kocher [17], and online schemes like OCSP [23].

QCM does not advocate one distribution mechanism over another; we have implemented several (e.g., online and offline schemes, push and pull styles of mirroring, and Naor and Nissim's tree-based scheme [24]) and expect users to choose between them based on their costs and benefits. To make it easy for the user to change distribution mechanisms, QCM separates the specification of revocation (what party determines which certificates are revoked?) from its implementation (distribution mechanism).

Finally, we note that all of the research on distributing revocation state applies equally to the distribution of authorizations (ACL's) and key bindings (public key directories). However, in the PKI's that we are aware of, revocation is always treated specially, with its own distribution mechanism. In QCM, the mechanisms for distributing revocation data are the same as those for distributing other kinds of data. In this way, when we implement an improved distribution mechanism, it applies to all kinds of data immediately.

2.3 Can We Do Without Revocation?

Revocation is less likely to be needed if certificates have short validity periods. For example, if passes to the swimming pool were issued once each semester rather than once each academic year, then on average only half as many entries would need to appear on a CRL. If Alice's swimming privileges were revoked during the first semester, then the serial number for her certificate would only need to appear in a CRL until the end of the first semester, and not until the end of the year. After the first semester, the certificate would be rejected because of expiration anyway, so it does not need to be in a CRL.

Carrying this to an extreme, if Alice was required to obtain a fresh certificate with a very short expiration period each time she went to the swimming pool, then the issuer could revoke her privileges very quickly simply by refusing to issue new certificates. Rivest [25] advocates this approach, and it clearly has advantages, e.g., the issuing party does not have to distribute CRL's and the relying party does not need to worry about CRL's going stale.

However, there are also drawbacks. Most seriously, it requires Alice to do more work, and it may require the issuer to

[1] To be more precise, there were two such lists, one for certificates for other CA's and one for the certificates of end users.

sign many more certificates than the X.509 approach, potentially placing an unacceptable burden on certificate servers. Another problem is that it does not address the problem of key compromise, which is closely related to revocation; Rivest proposes a separate mechanism, "suicide bureaus," to handle key compromise.

Our view is that in some circumstances, short certificate lifetimes are appropriate, and in others, revocation distribution mechanisms like CRL's should be used. We provide support for both in QCM and we let the user decide between them. McDaniel and Rubin [18] study the advantages and disadvantages of the two approaches in more detail.

2.4 Risk, Cost, and Liability

Any system for certificate revocation must support tradeoffs in the risks, costs, and liabilities that arise in electronic transactions. For example, consider a transaction where the issuing party is a bank, the relying party is a merchant, and the certificate is the electronic equivalent of a bank credit card, presented to the merchant by a customer trying to make a purchase. It is easy for the merchant to check the signature and expiration date on the certificate, but revocation information is harder to come by. Conventionally, the issuer periodically distributes revocation state via CRL's. If CRL's are issued often, then revoked certificates are more likely to be caught, lowering risk. At the same time, the cost of distribution increases: more compute cycles and bandwidth are needed for more frequent CRL updates, and more work is required from the issuing and/or relying parties.

Much of the research into revocation concerns how to reduce the distribution cost as a whole, but an equally important issue is how the costs of revocation are divided among the parties. This includes not only distribution costs, but also *liability*, which determines what party pays how much when a transaction goes bad. In other words, each party of the transaction tries to balance its portion of the cost of distributing revocation state against the risk (probability) of a bad transaction and the liability that might be incurred.

A revocation system should allow tradeoffs in risk, cost, and liability to be expressed and (as much as possible) enforced. For example, the bank may want to assume all liability for bad transactions, to make its credit card more appealing to merchants and thus more widely used by consumers. To mitigate its liability, it can spend some money and effort detecting bad accounts and distributing revocations, and it may want to *require* merchants to check for revocation. On the other hand, it may not be possible for the bank to take on all liability, so the merchant should be free to make more effort to prevent bad transactions than required by the bank. For example, the merchant could consult a third-party credit rating agency before approving large purchases. In QCM, it is possible for the issuing party to force the relying party to check for revocation, and it is also possible for the relying party to check for revocation independently.

Rivest [25] and McDaniel and Rubin [18] have more discussion of these issues.

3 QCM with Revocation

QCM is a programming language for securely specifying and evaluating distributed tables; its syntax is based on a calculus of set comprehensions [3, 4]. QCM programs can express *security policies* much like those of PolicyMaker [2], as well as the *groups* of SDSI [26]. QCM improves on SDSI and PolicyMaker by supporting automatic retrieval of remotely-stored certificates as part of group or policy enquiries; SDSI and PolicyMaker leave certificate retrieval to a separate, as-yet-undefined mechanism. Conversely, while systems like X.509's Directory [13] and LDAP [30] provide certificate distribution, they are not integrated with verification (certificate use) as in QCM.

Users of QCM write programs (security policies) in a high level language that we call the *external* language. Our implementation translates programs in the external language into an *internal* language that facilitates query decomposition and optimization. In this section, we will introduce the internal language and define its denotational semantics. We focus on the features we need for key compromise and revocation, omitting some of the other mechanisms of the language to simplify the presentation. These omitted features are described in other papers [10, 15].

3.1 Introduction to QCM

We will introduce QCM by example. Suppose a research group L is collaborating with a group R at a remote site and wishes to maintain an appropriate Access Control List for the resources at L to be used in the project. L can define a set, named ACL, of the permitted users with a QCM definition:

$$\mathrm{ACL} = \mathrm{LocalUsers} \cup K_R\$\mathrm{ACL}. \qquad (\dagger)$$

Here K_R is the public key for the group R. The notation $K_R\$\mathrm{ACL}$ is pronounced, "K_R's ACL," and it is the global name of a set ACL defined by R. K_R's ACL is distinct from the ACL defined by (†), which is known globally as K_L's ACL. By qualifying names with keys, QCM ensures that the sets defined by different principals will not be confused. Moreover, the QCM implementation will discard any information regarding $K_R\$\mathrm{ACL}$ unless it comes with a signature verified by K_R; this means that only R, the holder of the secret, signing key, can convince QCM that a user is in $K_R\$\mathrm{ACL}$.

The definition (†) says that K_L's ACL is the union of a set, LocalUsers, and the set $K_R\$\mathrm{ACL}$. LocalUsers is defined separately by L, for example, its members can be listed explicitly:

$$\mathrm{LocalUsers} = \{K_L\}.$$

After L has made these definitions, a QCM evaluator on the local machine can be queried with set expressions involving ACL and LocalUsers. For example, if QCM were asked the query 'ACL?' it would eventually return a set $\{K_L, \ldots\}$. Exactly how it does this is largely hidden from the user. LocalUsers is easy to obtain, of course, but to obtain $K_R\$\mathrm{ACL}$ QCM might use a variety of different strategies. The most straightforward would be to send a message to a QCM evaluator at R's site. An optimization would be to cache the response. An extension of this optimization is to mirror the set $K_R\$\mathrm{ACL}$ locally at L. This option breaks into two possibilities: one in which R 'pushes' the ACL (or a delta of it) whenever it changes, the second in which L 'pulls' the ACL (if it has changed) whenever it needs to use it. In each case, it is essential to secure the integrity of the communications, so QCM signs messages and verifies signatures when appropriate. QCM automatically and seamlessly supports all of these mechanisms, as well as other commonly-used mechanisms like online versus offline signing [10, 15].

Usually, the entire ACL is not needed; a more typical query would ask whether K_{Alice} was a member of the ACL. To keep the language simple, QCM does not have such Boolean queries, but they can be encoded as set queries. For example, to see if K_{Alice} is a member of the ACL, it is sufficient to ask the query

$$\{\text{“yes”} \mid x \in \text{ACL}, x = K_{\text{Alice}}\}. \qquad (\ddagger)$$

This will evaluate to the set $\{\text{“yes”}\}$ if $K_{\text{Alice}} \in \text{ACL}$, and if $K_{\text{Alice}} \notin \text{ACL}$, it will evaluate to the empty set, $\{\}$.

We have not yet discussed how QCM supports certificates. In QCM a certificate is a signed statement about names, for example:

$$\text{‘}K_{\text{Alice}} \in \text{ACL'} \ (\text{signed } K_R) \qquad (\S)$$

This certificate is a statement by K_R attesting that K_{Alice} is in its ACL. When QCM receives (§), it will believe that $K_{\text{Alice}} \in K_R\$\text{ACL}$ (after verifying the signature, of course). So, if QCM is asked the query (‡) and is given the certificate (§) at the same time, it can evaluate the query to $\{\text{“yes”}\}$ without sending any messages to R.

If QCM is asked the query ‘ACL?’ and is given (§) at the same time, it behaves in exactly the same way: it believes that $K_{\text{Alice}} \in K_R\$\text{ACL}$, and it evaluates the query *without sending any messages to R*. The answer returned is $\{K_L, K_{\text{Alice}}\}$, regardless of whether there are any additional keys in $K_R\$\text{ACL}$. This illustrates an important point: the answer returned by QCM is not guaranteed to be an exact answer. However, the answer returned is always a lower bound (subset) of the ‘real’ answer. This is guaranteed by careful restrictions built into the language.

This is a sensible choice, for two reasons. First, it is conservative: some people who should be approved may be denied access, but no one who should be denied access will be approved. Second, there are situations in which it is appropriate for QCM to refuse to send any messages. For example, we might need to guard against an adversary who submits queries so as to make QCM send many messages. QCM can defend against such an attack while still functioning by operating in a mode where messages are not sent but certificates are accepted.

However, using lower bounds presents difficulties for revocation. The most obvious way to add revocation to QCM would be to add a non-membership test to the language. For example, suppose we define an access control list that assigns a unique serial number to each user on the list:

$$\text{ACL} = \{(K_L, 1), (K_R, 2), \ldots\}$$

If we separately define CRL to be the set of serial numbers of users who should be revoked, then the query

$$\{x \mid (x, n) \in \text{ACL}, n \notin \text{CRL}\}$$

is the set of users who should have access. If we wish to calculate a lower bound for the query, we need a lower bound for ACL, but we need an *upper* bound for CRL. Fortunately, these are dual notions, so most of the existing machinery of QCM can be adapted to calculate upper bounds. The main additions required are certificates that give upper bounds (‘$K_{\text{Alice}} \notin$ CRL’) and the polarity discipline of Section 4 that will guarantee that we do not run into the semantic inconsistencies illustrated in Section 2.1.

Table 1: QCM's internal language

Keys	K	$\in$	Key
Constants	c	$\in$	$\mathsf{Key} \cup \mathsf{Num}$ $\cup\, \mathsf{Str} \cup \mathsf{Bool}$
Positive variables	x^+	$\in$	Var^+
Negative variables	x^-	$\in$	Var^-
Positive names	u^+	$\in$	Name^+
Negative names	u^-	$\in$	Name^-
Variables	x	$::=$	$x^+ \mid x^-$
Names	u	$::=$	$u^+ \mid u^-$
Polarities	γ	$::=$	$+ \mid -$
Expressions	e	$::=$	
			$x \mid c \mid$
Qualified names			$e\$u \mid$
Enumerated sets			$\{e, \ldots, e\} \mid$
Tuples			$(e, \ldots, e) \mid$
Set union			$\bigcup e \mid$
Comprehensions			$\{e \mid g, \ldots, g\} \mid$
Evaluate at			$(e@e) \mid$
Co-finite sets			$\mathsf{cmpl}\{w, \ldots, w\}$
Values	v	$::=$	$c \mid (v, \ldots, v) \mid$ $\{v, \ldots, v\} \mid$ $\mathsf{cmpl}\{w, \ldots, w\}$
Comparable values	w	$::=$	$c \mid (w, \ldots, w)$
	g	$::=$	
Generators			$p \in e \mid$
Guards			$e = e \mid$
			$e \neq e \mid$
			$e \notin e$
Patterns	p	$::=$	$x \mid (x, \ldots, x)$
Definitions	d	$::=$	$u = e$
Programs	P	$::=$	$d_1, \ldots, d_n$

3.2 The Internal Language

A grammar for the internal language is given in Table 1. It includes the usual constants (numbers, booleans, and strings) as well as keys. Names u and local variables x are tagged with positive or negative *polarities*, γ. These annotations are used in the polarity discipline given in the next section; they can be inferred for variables, but they are required for names.

The expressions e of the language include constants, variables, names, tuples $(e_1, \ldots, e_n)$, sets $\{e_1, \ldots, e_n\}$, and a set union operator. They also include set comprehensions such as $\{x \mid (x, y) \in u, x = y\}$. Informally, this comprehension is an iteration construct that builds a set with an element x for each element of u that matches the pattern (x, y) and satisfies the condition $x = y$. Patterns are restricted so that a variable may not occur twice. Thus, (x, y) is a well-formed pattern, but (x, x) is not. Notice that every pattern is an expression. Tuples and tuple patterns must contain at least two components, so an expression like (x) is not well-formed. On the other hand, an enumerated set may have one or no elements. An expression of the form $\{e\}$ is a singleton, and $\{\}$ is the empty set.

We use boldface metavariables for sequences, e.g., $\mathbf{g}$ denotes a sequence $g_1, \ldots, g_n$ of guards and generators. In the expression $\{e_1 \mid x \in e_2, \mathbf{g}\}$, the visible occurrence of x is a binding of that variable and binds free occurrences of x in e_1 and $\mathbf{g}$, but not in e_2. Guards do not introduce bindings. For example, if e_2 and e_3 have no free variables, then the meaning of the expression $\{x \mid x \in e_2, x \in e_3\}$ is the same as that of e_3. Expressions are considered syntactically identical modulo renaming of bound variables, reordering of enumerated sets, and elimination of duplicates in enumerated sets.

Set difference is worth introducing as syntactic sugar:

$$e_1 - e_2 = \{x \mid x \in e_1, x \notin e_2\}$$

Note that x in the generator binds the occurrences of x in the guard and the range. Similar syntactic sugar, like a binary union operator, will be used without comment.

A program in the internal language is a set of definitions,

$$u_1 = e_1, \ldots, u_n = e_n.$$

We require the definitions to be acyclic, and $u_i \neq u_j$ if $i \neq j$.

QCM evaluation is based on a distributed family of QCM processes, each created from a QCM program. These processes act as servers to applications and to other QCM processes, both of which query QCM about the values of QCM expressions. QCM responds with a signed table created from the QCM processes acting as a distributed database over an ad hoc virtual private network. For brevity, these *response certificates* are omitted from this treatment. Dually, an application that has obtained one or more certificates previously may 'push' them to a QCM process in order to spare the process the trouble of obtaining them from other sources. These are called *push certificates*, and they take two forms:

$$\text{`}u^+ \sqsupseteq e\text{' (signed } K\text{)},$$
$$\text{`}u^- \sqsubseteq e\text{' (signed } K\text{)}.$$

The first form expresses a lower bound for a positive name. The second form expresses an upper bound for a negative name, which might be used for revocation and key compromise. These certificates generalize the membership and non-membership certificates we have been using in our exposition.

We require the bounds to be sets, and we do not permit certificates with upper bounds for positive names, or lower bounds for negative names. Because of these restrictions, there is *always* a model for any set of push certificates; in fact, there is a best, or minimal model. For example, in the minimal model of the certificates

$$\text{`ACL}^+ \sqsupseteq \{(K_L, 1), (K_R, 2)\}\text{' (signed } K\text{)},$$
$$\text{`CRL}^- \sqsubseteq \mathsf{cmpl}\{2\}\text{' (signed } K\text{)},$$

$K\$\text{ACL}^+$ has the value $\{(K_L, 1), (K_R, 2)\}$, and $K\$\text{CRL}^-$ has the value $\mathsf{cmpl}\{2\}$, the compliment of the set $\{2\}$. (The condition $\text{CRL}^- \sqsubseteq \mathsf{cmpl}\{2\}$ is equivalent to $2 \notin \text{CRL}^-$.)

Existence of the minimal model ensures that we avoid the semantic inconsistencies of Section 2.1. It also provides a way to evaluate queries that takes advantage of any pushed certificates: we construct the minimal model and evaluate the query in the model. The crucial *monotonicity* result of the next section shows that this results in lower bounds for positive sets, and upper bounds for negative sets.

We have already seen several examples of how QCM can express CRL's. Now we will give an example that shows how QCM can guard against key compromise. Key compromise occurs when a private, signing key is revealed to the adversary. Once the adversary has a signing key, they masquerade as the keyholder. To protect against this, we can maintain a set of compromised keys in QCM:

$$\text{Compromised}^- = \{K_1, K_2, \ldots\}$$

Now, anyone who wants to protect themselves against the compromised keys can refer to Compromised^- in writing their policies. For example,

$$\text{ACL}^+ = \{x^+ \mid K_R \notin \text{Compromised}^-, x^+ \in K_R\$\text{ACL}^+\}.$$

Here we check that the key K_R is not in the compromised set before adding any elements of $K_R\$\text{ACL}^+$ to the local ACL. In general, use of a compromised key can be prevented by transforming expressions of the form $e\$u$ into $\{x \mid e \notin \text{Compromised}, x \in e\$u\}$. This is a simple transformation that can be performed during the translation from the external language to the internal language.

The reader may be curious whether the program needs to be changed each time one of the sets in the definitions is changed, say by adding a new user to an ACL. For the treatment in this paper the answer is yes, but in practice QCM is implemented as a middleware system on top of one or more data sources like flat files, XML expressions, or LDAP databases. When the data in these underlying data sources changes, the runtime system takes this into account without the need for recompilation.

3.3 Denotational Semantics

The denotational semantics is given using a recursive domain equation and a collection of semantic clauses that compositionally describe the meanings of QCM expressions.

Let $[\![\mathsf{Key}]\!], [\![\mathsf{Num}]\!], [\![\mathsf{Str}]\!], [\![\mathsf{Bool}]\!]$ be the semantic spaces that interpret principals, numbers, strings, and booleans respectively. The semantic universe is defined via the following domain equation, where $\uplus$ is the disjoint union:

$$\begin{aligned} U = {} & [\![\mathsf{Key}]\!] \uplus [\![\mathsf{Num}]\!] \uplus [\![\mathsf{Str}]\!] \uplus [\![\mathsf{Bool}]\!] \\ & \uplus \mathcal{P}_{\text{fin}}(U) \uplus \mathcal{P}_{\overline{\text{fin}}}(U) \\ & \uplus \Pi(U, U) \uplus \Pi(U, U, U) \uplus \cdots \end{aligned}$$

We use the notation $\mathcal{P}_{\text{fin}}(U)$ for the finite subsets of U, and $\mathcal{P}_{\overline{\text{fin}}}(U)$ for the co-finite subsets of U. The expression $\Pi(U,U)$ is for pairs of U's, while $\Pi(U,U,U)$ is for triples, and so on.

Notice that every set in the semantic universe U is either finite or co-finite with respect to U; for example, $\mathsf{Num} \notin U$. This means that every element of U has a finite representation, which is convenient for our operational semantics.

To achieve monotonicity, it is essential to restrict comparison by equality or inequality to elements that are in a distinguished subset, C, of the universe; we define C to be the least subset of U that includes the elements of base type (principals, integers, booleans, strings) and is closed under the tuple operation.

An *environment* is a function $\rho : \mathsf{Var} \uplus (\mathsf{Key} \times \mathsf{Name}) \to U$. We write $\rho(x)$ for the value on variables and $\rho(K,u)$ for the value on principal/name pairs. We restrict ourselves to environments ρ such that $\rho(K,u)$ is a set for any K and u.

The semantics of the language is a partial function on expressions e and environments ρ denoted by $[\![e]\!]\rho \in U$. In our definition, we use the convention that in a clause of the form

$$[\![e]\!]\rho = \cdots [\![e']\!]\rho' \cdots,$$

the meaning of e relative to ρ has the value on the right hand side *provided* $[\![e']\!]\rho'$ is defined.

The denotational semantics of QCM is given by the semantic clauses of Table 2. Some expressions have no meaning; that is, $[\![e]\!]\rho$ may be undefined. A subtle example of this comes up in the clauses defining the meaning of comprehensions:

$$[\![\{e_1 \mid x \in e_2, \mathbf{g}\}]\!]\rho = \bigcup_{\nu \in [\![e_2]\!]\rho} [\![\{e_1 \mid \mathbf{g}\}]\!]\rho[x \mapsto \nu]$$

Here the right hand side may not be well-defined. For example, if $[\![e_2]\!]\rho$ is a co-finite set, the right hand side is an infinite union. The infinite union may result in a set which is neither finite nor co-finite, and hence, not in the semantic universe U. One example is

$$\{\{x\} \mid x \in \mathsf{cmpl}\{\}\}.$$

Intuitively, this is the set of all singletons; it is neither finite nor co-finite, and hence, has no meaning in our semantics.

4 The Polarity Discipline

We define an ordering $\sqsubseteq$ as the least relation on U satisfying the following properties:

- $\nu \sqsubseteq \nu$ for any ν
- $(\nu_1, \ldots, \nu_n) \sqsubseteq (\nu'_1, \ldots, \nu'_n)$ if $\nu_i \sqsubseteq \nu'_i$ for each $1 \leq i \leq n$.
- $\nu \sqsubseteq \nu'$ if ν and ν' are sets, and for every $\nu_0 \in \nu$, there exists a $\nu'_0 \in \nu'$ such that $\nu_0 \sqsubseteq \nu'_0$.

This is the ordering on U that would arise from treating it as the solution of a domain equation as a partial order, with coordinate-wise ordering on tuples and the lower powerdomain ordering on sets. Note in particular that $\nu \subseteq \nu'$ implies $\nu \sqsubseteq \nu'$. Also, elements of C are $\sqsubseteq$-comparable only to themselves: if $\nu \in C$ and $\nu \sqsubseteq \nu'$ or $\nu' \sqsubseteq \nu$, then $\nu = \nu'$.

For a polarity γ we define

$$\sqsubseteq_\gamma \quad = \quad \begin{cases} \sqsubseteq & \text{if } \gamma = +, \\ \sqsupseteq & \text{if } \gamma = -, \end{cases}$$

and we extend the ordering $\sqsubseteq$ to environments: $\rho \sqsubseteq \rho'$ iff

- $\rho(x^\gamma) \sqsubseteq_\gamma \rho'(x^\gamma)$ for every x^γ, and
- $\rho(K, u^\gamma) \sqsubseteq_\gamma \rho'(K, u^\gamma)$ for every $K\$u^\gamma$.

We will adopt the convention that $[\![e]\!]\rho \sqsubseteq [\![e']\!]\rho'$ iff $[\![e]\!]\rho$ and $[\![e']\!]\rho'$ are both undefined, or they are both defined and ordered by $\sqsubseteq$.

The rules for assigning polarities to QCM expressions are given in Table 3.

The last, most interesting rule shows how non-membership guards affect polarity:

$$\frac{\{e_1 \mid \mathbf{g}\} : \gamma \qquad e_3 : -\gamma}{\{e_1 \mid e_2 \notin e_3, \mathbf{g}\} : \gamma}$$

Here, $-+ = -$ and $-- = +$, so the rule is contravariant in e_3. Every other rule treats polarities covariantly.

Another point to note is that the rules do not take the polarities of operands of other guards into consideration (e.g., the polarities of e_2 and e_3 in the guard $e_2 = e_3$ are ignored). This is because the denotational semantics forces their meanings to be in C, and every element of C is $\sqsubseteq$-comparable only to itself. Similarly, the polarity of the qualifier e in the expression $e\$u^\gamma$ is also irrelevant.

Some expressions (including all values) have both positive and negative polarity; some have only one or the other; and some have neither polarity. For example, set difference has the following derived polarity rule:

$$\frac{e_1 : \gamma \qquad e_2 : -\gamma}{e_1 - e_2 : \gamma}$$

Here the subtrahend is treated contravariantly. Consequently, the expression $x^+ - x^+$ has no polarity. Nevertheless, it is possible to use positive variables in 'negative' positions, as in the following example.

$$\begin{aligned} &\{y^+ \mid x^+ \in \{K, K'\}, \\ &\qquad y^+ \in x^+\$\text{ACL}^+ - x^+\$\text{CRL}^-\} : + \end{aligned}$$

The expression denotes elements of U that appear on the ACL's of K and K', but not on their respective CRL's. The positive variable x^+ is used in the negative expression $x^+\$\text{CRL}^-$, but in a position where its polarity does not matter.

It is possible to translate QCM expressions without polarity annotations on variables into internal QCM expressions in time proportional to the size of the term. Thus QCM as a programming language does not need to expose the complexity of tagged variables to the programmer: the existence of a polarity is inferred automatically. However, even for external QCM we insist on annotating the polarities of names: a programmer must indicate for each name whether it is to be viewed as 'ACL-like' or 'CRL-like.'

Theorem 1 (Monotonicity) *Let e be an expression and let ρ, ρ' be environments such that $\rho \sqsubseteq \rho'$. Suppose $\nu = [\![e]\!]\rho$ and $\nu' = [\![e]\!]\rho'$ are defined.*

1. *If $e : +$, then $\nu \sqsubseteq \nu'$.*
2. *If $e : -$, then $\nu \sqsupseteq \nu'$.*
3. *If $\nu \in C$ or $\nu' \in C$, then $\nu = \nu'$.*

The Monotonicity Theorem justifies our strategy of evaluating push certificates using the minimal model described in the previous section.

Table 2: Clauses of QCM's denotational semantics

Variables: $[\![x]\!]\rho = \rho(x)$

Constants: Constants are given their usual interpretation. For instance $[\![2]\!]\rho$ is the number 2 viewed as an element of the $[\![\mathsf{Num}]\!]$ part of U. Principals K take their meanings in the $[\![\mathsf{Key}]\!]$ part of the domain.

Qualified Names: $[\![e\$u]\!]\rho = \rho([\![e]\!]\rho, u)$. Note that the conventions about definition mean that the meaning of $e\$u$ with respect to environment ρ is given by the expression on the right if $[\![e]\!]\rho$ is defined and is a principal; it is undefined otherwise. Also, our restriction on environments implies that the meaning of $e\$u$ is a set if it is defined.

Evaluate At: $[\![e_1@e_2]\!]\rho = [\![e_1]\!]\rho$ provided $[\![e_2]\!]\rho \in [\![\mathsf{Key}]\!]$; otherwise $[\![e_1@e_2]\!]\rho$ is undefined.

Enumerated Sets: $[\![\{e_1, \ldots, e_n\}]\!]\rho = \{[\![e_1]\!]\rho, \ldots, [\![e_n]\!]\rho\}$.

Co-finite Sets: $[\![\mathsf{cmpl}\{w_1, \ldots, w_n\}]\!]\rho = U - \{[\![w_1]\!]\rho, \ldots, [\![w_n]\!]\rho\}$.

Notice here that the complement is taken with respect to U, not C.

Tuples: $[\![(e_1, \ldots, e_n)]\!]\rho = ([\![e_1]\!]\rho, \ldots, [\![e_n]\!]\rho)$.

Set Unions: $[\![\bigcup e]\!]\rho = \{\nu \mid \nu \in \nu' \text{ for some } \nu' \in [\![e]\!]\rho\}$.

Is an Element of: $[\![\{e_1 \mid x \in e_2, \mathbf{g}\}]\!]\rho = \bigcup_{\nu \in [\![e_2]\!]\rho} [\![\{e_1 \mid \mathbf{g}\}]\!]\rho[x \mapsto \nu]$

Is a Pattern in: $[\![\{e_1 \mid (\mathbf{x}) \in e_2, \mathbf{g}\}]\!]\rho = \bigcup_{(\boldsymbol{\nu}) \in [\![e_2]\!]\rho} [\![\{e_1 \mid \mathbf{g}\}]\!]\rho'$

where $\rho' = \rho[\mathbf{x} \mapsto \boldsymbol{\nu}]$.

Is Equal to: $[\![\{e_1 \mid e_2 = e_3, \mathbf{g}\}]\!]\rho = [\![\{e_1 \mid \mathbf{g}\}]\!]\rho$ provided $[\![e_2]\!]\rho = \nu_2$ and $[\![e_3]\!]\rho = \nu_3$ are equal and both are elements of C. If they are both in C but they are not equal then the meaning of the expression is the empty set $\{\}$. If either of them is undefined or not in C then the meaning is undefined.

Is not Equal to: $[\![\{e_1 \mid e_2 \neq e_3, \mathbf{g}\}]\!]\rho = [\![\{e_1 \mid \mathbf{g}\}]\!]\rho$ provided $[\![e_2]\!]\rho = \nu_2$ and $[\![e_3]\!]\rho = \nu_3$ are unequal and both are elements of C. If both are elements of C but they are not equal then the meaning of the expression is the empty set $\{\}$. If either of them is undefined or not in C then the meaning is undefined.

Is not an Element of: $[\![\{e_1 \mid e_2 \notin e_3, \mathbf{g}\}]\!]\rho = [\![\{e_1 \mid \mathbf{g}\}]\!]\rho$ provided $[\![e_2]\!]\rho = \nu_2$ is in C, and $[\![e_3]\!]\rho$ is a set ν_3 and $\nu_2 \notin \nu_3$. If ν_2 is in C and ν_3 is a set with $\nu_2 \in \nu_3$, then the meaning is the empty set $\{\}$. If ν_2 is not in C, or ν_3 is not a set, then the meaning is undefined. Note that ν_3 may contain elements not in C.

Base Case: $[\![\{e \mid \}]\!]\rho = \{[\![e]\!]\rho\}$.

Table 3: QCM's polarity rules

$$c : \gamma \qquad \frac{e_1 : \gamma \quad \ldots \quad e_n : \gamma}{(e_1, \ldots, e_n) : \gamma} \qquad \frac{e_1 : \gamma \quad \ldots \quad e_n : \gamma}{\{e_1, \ldots, e_n\} : \gamma} \qquad x^{\gamma} : \gamma \qquad e\$u^{\gamma} : \gamma \qquad \frac{e : \gamma}{\bigcup e : \gamma}$$

$$\frac{e_1 : \gamma}{(e_1@e_2) : \gamma} \qquad \mathsf{cmpl}\{\mathbf{w}\} : \gamma \qquad \frac{e : \gamma}{\{e \mid \} : \gamma} \qquad \frac{\{e_1 \mid \mathbf{g}\} : \gamma \quad x : \gamma \quad e_2 : \gamma}{\{e_1 \mid x \in e_2, \mathbf{g}\} : \gamma} \qquad \frac{\{e_1 \mid \mathbf{g}\} : \gamma \quad p : \gamma \quad e_2 : \gamma}{\{e_1 \mid p \in e_2, \mathbf{g}\} : \gamma}$$

$$\frac{\{e_1 \mid \mathbf{g}\} : \gamma}{\{e_1 \mid e_2 = e_3, \mathbf{g}\} : \gamma} \qquad \frac{\{e_1 \mid \mathbf{g}\} : \gamma}{\{e_1 \mid e_2 \neq e_3, \mathbf{g}\} : \gamma} \qquad \frac{\{e_1 \mid \mathbf{g}\} : \gamma \quad e_3 : -\gamma}{\{e_1 \mid e_2 \notin e_3, \mathbf{g}\} : \gamma}$$

5 Operational Semantics

We now present a formal operational semantics for the internal language of QCM. For simplicity, the semantics omits details of the QCM implementation that are not relevant to our main Soundness Theorem. Some of the important omissions are:

- The semantics assumes nodes can communicate instantaneously and securely. In our implementation, nodes must exchange explicit, signed messages to communicate.
- The semantics does not mention expiration times. Our implementation keeps track of a valid time interval for each job executed, and this interval is adjusted each time a certificate is used or a message is exchanged.
- The semantics does not describe how certificates are 'pushed' at a node, or how certificates are verified, or what happens if a signature is found to be invalid.
- The semantics ignores efficiency; in practice we perform meaning-preserving source-to-source transformations to optimize evaluation. For example, we reorder computations and insert @-annotations to reduce the size of messages and intermediate results.

Another paper describes how our implementation handles these details [10].

The rules of Table 4 define a rewriting relation, →, for computations that can be performed locally, on a single node in the network. The rules use an auxiliary function, eq, for testing the equality of values. We define eq to be the least partial function from values to booleans satisfying the following conditions.

- $\mathsf{eq}(c, c) = \mathsf{true}$
- $\mathsf{eq}(c, c') = \mathsf{false}$ if $c \neq c'$
- $\mathsf{eq}(\ (v_1, \ldots, v_n), (v'_1, \ldots, v'_n)\ ) = \mathsf{true}$ if $\mathsf{eq}(v_i, v'_i) = \mathsf{true}$ for all $i \in \{1, \ldots, n\}$
- $\mathsf{eq}(\ (v_1, \ldots, v_n), (v'_1, \ldots, v'_n)\ ) = \mathsf{false}$ if $\mathsf{eq}(v_i, v'_i)$ is defined for all $i \in \{1, \ldots, n\}$, and $\mathsf{eq}(v_j, v'_j) = \mathsf{false}$ for some $j \in \{1, \ldots, n\}$

In other words, $\mathsf{eq}(w, w')$ is true iff $w = w'$, and $\mathsf{eq}(v, v')$ is undefined iff v or v' is not a comparable value. In particular, eq is not defined on sets, e.g., $\mathsf{eq}(\{\}, \{\})$ is undefined. We do not permit equality on sets because this would make the query language nonmonotonic.

Notice that the rule for non-membership may require applying the function eq to some values on which it is undefined. However, it is decidable whether eq is defined. When eq is undefined, an evaluation can become *stuck*.

Other examples of stuck expressions arise from co-finite sets. For example,

- $\{e \mid p \in \mathsf{cmpl}\{\mathbf{w}\}\}$
- $\bigcup\{\mathsf{cmpl}\{\mathbf{w}\}, \{\mathbf{v}\}\}$

It seems hard to imagine a rule for effectively evaluating the first expression here; in fact, in our semantics, such expressions may not have a meaning, because they would denote sets which are neither finite nor co-finite. On the other hand, it is possible to extend our rules for $\bigcup$ to handle co-finite sets. One design issue is that we have used comprehensions to define sugared versions of set intersection and difference; if we cannot use comprehensions to range over co-finite sets, we cannot perform intersection between and difference from co-finite sets. This could be addressed by adding intersection and difference as primitives to the language and adding operational rules for them.

Table 5: Jobs and evaluation contexts

Job numbers	n	$\in$	Num
Job identifiers	i, j	$::=$	(K, n)
Jobs	J	$::=$	$\mathsf{w}^i_j(e) \mid \mathsf{a}^i_j(E[e@K])$
Evaluation contexts	E	$::=$	$[\cdot] \mid E\$u \mid e@E \mid$ $(\mathbf{v}, E, \mathbf{e}) \mid \{\mathbf{v}, E, \mathbf{e}\} \mid$ $\bigcup E \mid$ $\{e \mid p \in E, \mathbf{g}\} \mid$ $\{e \mid E = e, \mathbf{g}\} \mid$ $\{e \mid v = E, \mathbf{g}\} \mid$ $\{e \mid E \neq e, \mathbf{g}\} \mid$ $\{e \mid v \neq E, \mathbf{g}\} \mid$ $\{e \mid E \notin e, \mathbf{g}\} \mid$ $\{e \mid v \notin E, \mathbf{g}\}$

Lemma 2 (Local Soundness) *If $e \to e'$, then $[\![e]\!] = [\![e']\!]$.*

Next we define a left-to-right order of evaluation, using the machinery of redexes and evaluation contexts.

Definition: We say an expression e is a *local redex* if it matches the left hand side of a rule of Table 4. We say e is a *global redex* if it has the form $e@K$ or $K\$u$.

Evaluation contexts are defined in Table 5. If E is an evaluation context and e is an expression, then $E[e]$ is the expression obtained by replacing the hole, $[\cdot]$, of E with e. Note that the grammar for E ensures that the hole does not appear in the scope of a variable binding, so capture of e's free variables is not an issue. We say e *has parse* $E[e']$ if $e = E[e']$ and e' is a redex.

Redexes and evaluation contexts establish a notion of "next step," formalized as follows.

Lemma 3 (Parsing) *For any expression e, exactly one of the following holds:*

1. *e is a value;*
2. *there is a unique parse $E[e']$ of e; or*
3. *e is not a value and does not have a parse.*

In other words, either evaluation is completed, or there is a unique redex to work on, or the expression is stuck.

Global computation in QCM involves multiple *jobs* running at locations distributed throughout a network. To simplify our formal treatment, we identify locations with principals; that is, each principal is a distinct location. To distinguish the jobs running at a location we use *job numbers*, ranged over by n. Each location will ensure that it assigns a unique job number to each of its jobs, but because a location has no control over the job numbers assigned by other locations, unique global *job identifiers* consist of the job number and location together. We use i and j to range over job identifiers, and we write $\mathsf{loc}(i)$ for the location given by identifier i: if $i = (K, n)$, then $\mathsf{loc}(i) = K$.

Table 4: Rules defining the local evaluation relation, $e \to e'$

$$
\begin{array}{rcl}
\bigcup(\{\mathbf{v}_1\},\ldots,\{\mathbf{v}_n\}) & \to & \{\mathbf{v}_1,\ldots,\mathbf{v}_n\} \\
\{e \mid \} & \to & \{e\} \\
\{e \mid v_1 = v_2, \mathbf{g}\} & \to & \begin{cases} \{\} & \text{if } \mathsf{eq}(v_1,v_2) = \mathsf{false} \\ \{e \mid \mathbf{g}\} & \text{if } \mathsf{eq}(v_1,v_2) = \mathsf{true} \end{cases} \\
\{e \mid v_1 \neq v_2, \mathbf{g}\} & \to & \begin{cases} \{\} & \text{if } \mathsf{eq}(v_1,v_2) = \mathsf{true} \\ \{e \mid \mathbf{g}\} & \text{if } \mathsf{eq}(v_1,v_2) = \mathsf{false} \end{cases} \\
\{e \mid v_1 \notin \{\mathbf{v}\}, \mathbf{g}\} & \to & \begin{cases} \{\} & \text{if } \mathsf{eq}(v_1,v_i) = \mathsf{true} \text{ for some } v_i \in \mathbf{v} \\ \{e \mid \mathbf{g}\} & \text{if } \mathsf{eq}(v_1,v_i) = \mathsf{false} \text{ for every } v_i \in \mathbf{v} \end{cases} \\
\{e \mid v \notin \mathsf{cmpl}\{\mathbf{w}\}, \mathbf{g}\} & \to & \begin{cases} \{e \mid \mathbf{g}\} & \text{if } \mathsf{eq}(v,w_i) = \mathsf{true} \text{ for some } w_i \in \mathbf{w} \\ \{\} & \text{if } \mathsf{eq}(v,w_i) = \mathsf{false} \text{ for every } w_i \in \mathbf{w} \end{cases} \\
\{e \mid p \in \{\}, \mathbf{g}\} & \to & \{\} \\
\{e \mid x \in \{v, \mathbf{v}'\}, \mathbf{g}\} & \to & \bigcup\{\ \{e \mid \mathbf{g}\}[x \mapsto v],\ \{e \mid p \in \{\mathbf{v}'\}, \mathbf{g}\}\ \} \\
\{e \mid (x_1,\ldots,x_n) \in \{(v_1,\ldots,v_n), \mathbf{v}'\}, \mathbf{g}\} & \to & \bigcup\{\ \{e \mid \mathbf{g}\}[x_1,\ldots,x_n \mapsto v_1,\ldots,v_n],\ \{e \mid p \in \{\mathbf{v}'\}, \mathbf{g}\}\ \}
\end{array}
$$

Each job J runs at the request of a client, so jobs are associated with two job identifiers, the identifier of the job itself, and the identifier of the client job. Jobs can be in one of two states, given by the following constructors:

- $\mathsf{w}^i_j(e)$: A job running at i for client j, working on the expression e.
- $\mathsf{a}^i_j(E[e@K])$: A job running at i for client j, awaiting a value for e from location K.

A *global state* consists of a set of jobs and a set of job identifiers, and is written $\mathbf{J};\mathbf{i}$. We write $\vdash \mathbf{J};\mathbf{i}$ and say $\mathbf{J};\mathbf{i}$ *is well-formed* if every identifier in $\mathbf{J}$ appears in $\mathbf{i}$, and every job in $\mathbf{J}$ has a distinct identifier and client identifier (for simplicity, a client can have only one outstanding request). The set $\mathbf{i}$ will be used to ensure that job numbers are not re-used.

A network of QCM programs is described by a partial function $D : \mathsf{Key} \times \mathsf{Key} \times \mathsf{Name} \to \mathsf{Exp}$. The expression $D(K_1, K_2, u)$ is what K_1 believes $K_2\$u$ to be, and may be completely unrelated to $D(K_2, K_2, u)$. $D(K_1, K_2, u)$ is required to be closed. We write $\rho \models D$ if for all K, we have $\rho(K\$u) = \llbracket D(K, K, u)\rrbracket$, and for all K_1 and K_2, we have $\llbracket D(K_1, K_2, u^\gamma)\rrbracket \sqsubseteq_\gamma \rho(K_2\$u^\gamma)$. We allow $\llbracket D(K_1, K_2, u)\rrbracket$ differ from $\rho(K_2\$u)$ to model the effect of push certificates in the system.

We define a rewriting relation, $\Rightarrow_D$, on global states by the rules of Table 6. We write $\vdash \mathbf{J};\mathbf{i} \Rightarrow^*_D \mathbf{J}';\mathbf{i}'$ if $\mathbf{J}';\mathbf{i}'$ is obtained by zero or more rewrites from a well-formed global state $\mathbf{J};\mathbf{i}$.

Theorem 4 (Soundness) *If* $\vdash \mathsf{w}^i_j(e), \mathbf{J};\mathbf{i} \Rightarrow^*_D \mathsf{w}^i_j(e'), \mathbf{J}';\mathbf{i}'$, $\rho \models D$, *and* $e : \gamma$, *then* $\llbracket e'\rrbracket\rho \sqsubseteq_\gamma \llbracket e\rrbracket\rho$.

6 The External Language

The external language of QCM is a simplified version of the internal language that hides the details of revocation and the polarity discipline from users. We describe the external language informally here; it is likely to evolve as we gain more experience from our implementation.

An external language program is a sequence of definitions for names,

$$u_1 = e_1,\ \ldots,\ u_n = e_n.$$

The program must be an internal language program, with a few additional restrictions. Except for a distinguished name, Compromised$^-$, every name u_i must be a positive name, and the corresponding definition e_i can only refer to positive names and must not use non-membership (we permit non-membership in the internal language only). This simple restriction ensures that the user can only define positive sets.

The optional distinguished name, Compromised$^-$, is used to specify what keys are to be considered compromised. For example, the definition

$$\text{Compromised}^- = K_{\text{Alice}}\$\text{Compromised}^-$$

would be used to delegate responsibility for key compromise to Alice. The definition of Compromised$^-$ must refer only to negative names and must not use non-membership; this ensures that the compromised set is negative.

We use the method suggested in Section 3 to ensure that compromised keys are not relied on in determining the user's policies. Namely, we transform the program into an internal language program by replacing each subexpression $e\$u$ of a positive definition by $\{x \mid e \notin \text{Compromised}^-, x \in e\$u\}$. The resulting program will have a negative definition for Compromised$^-$, and positive definitions for positive names, so it satisfies our polarity discipline and our soundness result will apply.

Table 6: Rules defining the global evaluation relation, $\mathbf{J};\mathbf{i} \Rightarrow_D \mathbf{J}';\mathbf{i}'$

Local Evaluation The local evaluation rules are applied at a given node.

$$\mathsf{w}_j^i(E[e]), \mathbf{J}; \mathbf{i} \Rightarrow_D \mathsf{w}_j^i(E[e']), \mathbf{J}; \mathbf{i} \quad \text{if } e \to e'$$

Evaluation of names When a qualified name $K\$u$ is encountered at a node with a definition for $K\$u$, it is replaced by the definition. If no definition is available locally, then a query will be sent to K.

$$\mathsf{w}_j^i(E[K\$u]), \mathbf{J}; \mathbf{i} \Rightarrow_D \mathsf{w}_j^i(E[e]), \mathbf{J}; \mathbf{i} \qquad \text{if } D(\mathsf{loc}(i), K, u) = e$$

$$\mathsf{w}_j^i(E[K\$u]), \mathbf{J}; \mathbf{i} \Rightarrow_D \mathsf{w}_j^i(E[(K\$u)@K]), \mathbf{J}; \mathbf{i} \qquad \text{if } D(\mathsf{loc}(i), K, u) \text{ is undefined and } K \neq \mathsf{loc}(i).$$

Queries To evaluate an expression at a different location K, we start a job at K marked with the requesting location and a fresh job number.

$$\mathsf{w}_j^i(E[e@K]), \mathbf{J}; \mathbf{i} \Rightarrow_D \mathsf{w}_j^i(E[e]), \mathbf{J}; \mathbf{i} \qquad \text{if } K = \mathsf{loc}(i)$$

$$\mathsf{w}_j^i(E[e@K]), \mathbf{J}; \mathbf{i} \Rightarrow_D \mathsf{a}_j^i(E[e@K]), \mathsf{w}_i^{(K,n)}(e), \mathbf{J}; (K,n), \mathbf{i} \qquad \text{if } K \neq \mathsf{loc}(i) \text{ and } (K,n) \notin \mathbf{i}$$

Responses When K_1 has completed the evaluation of an expression on behalf of K, then it is provided to K and substituted into the awaiting context.

$$\mathsf{a}_j^i(E[e@K]),\ \mathsf{w}_i^{i'}(v), \mathbf{J}; \mathbf{i} \Rightarrow_D \mathsf{w}_j^i(E[v]), \mathbf{J}; \mathbf{i}$$

The internal language program can be maintained by an online QCM server that accepts queries about the user's policies, and calculates and signs answers on the spot. These signed answers are, of course, certificates. However, we do not support their revocation because they are produced on demand and are given short lifetimes.

The program can also be used to produce certificates that are signed offline and distributed by a QCM server that does not have access to the signing key. These certificates will typically be given longer lifetimes, and so we support their revocation, as follows.

Such "offline programs" are evaluated in two phases. Given a program $u_1 = e_1, \ldots u_n = e_n$, we first evaluate each definition e_i to some value $\{v_1, \ldots, v_m\}$; this determines the user's policies. The first phase is completed by issuing the certificates. The certificates contain statements of the form

$$u_i^+ \sqsupseteq \{v_j \mid n_{i,j} \notin K\$\text{Revoked}_{u_i}^-\},$$

where $n_{i,j}$ is a unique serial number for the certificate, and K is the key of the party who will determine the revocation state of each certificate. The semantics of QCM guarantee that a relying party will only conclude that $v_j \in u_i$ provided $n_{i,j} \notin K\$\text{Revoked}_{u_i}^-$; so, relying parties are forced to consult the revocation set, $K\$\text{Revoked}_{u_i}^-$.

In the second phase, the certificates and assignment of serial numbers are given to a QCM server that fields queries about the user's policies from the network and responds with an appropriate set of the pre-signed certificates. We describe how the server does this in a separate paper [10]. The server may also handle the revocation sets $K\$\text{Revoked}_{u_i}^-$, or this may be handled by a separate server.

The external language and compilation we have described here are only one way of using the internal language to support revocation. For example, to simplify things for the user, we have hard-coded some decisions into our compilation. We have placed the burden (or privilege) of key compromise on the relying parties: each relying party must specify what keys are compromised. On the other hand, we have given control over revocation to the issuing party: each certificate issued is useless to relying parties unless they check the revocation state. We expect the design of our external language and compilation to change in response to requests from our user community.

7 Conclusion

Certificate revocation presents a variety of fundamental problems in the semantics of certificates, the distribution of revocation state, and the apportionment of risk, cost, and liability. We have addressed these problems by extending a set-based programming language, QCM, with non-membership tests. The resulting language can express certificate revocation and key compromise, and has a semantic model and a sound implementation that supports varied distribution strategies and tradeoffs in risk and liability.

References

[1] William Aiello, Sachin Lodha, and Rafail Ostrovsky. Fast digital identity revocation. In *Advances in Cryptology: CRYPTO '98*, number 1462 in Lecture Notes in Computer Science, pages 137–152. Springer-Verlag, 1998.

[2] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proceedings of the 17th Symposium on Security and Privacy*, pages 164–173. IEEE Computer Society Press, 1996.

[3] Peter Buneman, Leonid Libkin, Dan Suciu, Val Tannen, and Limsoon Wong. Comprehension syntax. *SIGMOD Record*, 23(1):87–96, March 1994.

[4] Peter Buneman, Shamim Naqvi, Val Tannen, and Limsoon Wong. Principles of programming with complex objects and collection types. *Theoretical Computer Science*, 149(1):3–48, September 1995.

[5] Michael Burrows, Martín Abadi, and Roger M. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.

[6] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian M. Thomas, and Tatu Ylonen. SPKI certificate theory. Internet Draft, March 1998.

[7] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian M. Thomas, and Tatu Ylonen. SPKI examples. Internet Draft, March 1998.

[8] Warwick Ford and Michael S. Baum. *Secure Electronic Commerce.* Prentice Hall, 1999.

[9] Barbara Fox and Brian LaMacchia. Certificate revocation: Mechanics and meaning. In Hirschfeld [11], pages 158–164.

[10] Carl A. Gunter and Trevor Jim. Policy directed certificate retrieval, July 1998. `www.cis.upenn.edu/~qcm/papers/qcm-abstract.html`.

[11] Rafael Hirschfeld, editor. *Financial Cryptography: Second International Conference, FC'98*, number 1465 in Lecture Notes in Computer Science, Anguilla, British West Indies, 23–25 February 1998. Springer-Verlag.

[12] R. Housley, W. Ford, W. Polk, and D. Solo. *Internet X.509 Public Key Infrastructure: Certificate and CRL Profile.* IETF RFC 2459, January 1999.

[13] ISO/IEC 9594–1. *Information technology—Open Systems Interconnection—The Directory: Overview of concepts, models and services*, 1997. Equivalent to ITU-T Rec. X.500, 1997.

[14] ISO/IEC 9794–8. *Information technology—Open Systems Interconnection—The Directory: Authentication framework*, 1997. Equivalent to ITU-T Rec. X.509, 1997.

[15] Pankaj Kakkar, Michael McDougall, Carl A. Gunter, and Trevor Jim. Credential distribution with local autonomy. `www.cis.upenn.edu/~qcm/papers/autonomy-abstract.html`, March 1999.

[16] S. Kent. *Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management.* IETF RFC 1422, February 1993.

[17] Paul Kocher. On certificate revocation and validation. In Hirschfeld [11], pages 172–177.

[18] Patrick McDaniel and Aviel D. Rubin. A response to "Can we eliminate certificate revocation lists?". Technical Report 99.8.1, AT&T Labs, August 1999.

[19] Patrick D. McDaniel and Sugih Jamin. Windowed key revocation in public key infrastructures, 1999.

[20] Silvio Micali. Efficient certificate revocation. Technical Report TM–542, Massachusetts Institute of Technology, March 1996.

[21] Jonathan K. Millen and Rebecca N. Wright. Certificate revocation the responsible way. In *Proceedings of Computer Security, Dependability, and Assurance: From Needs to Solutions (CSDA'98)*, pages 196–203. IEEE Computer Society, 1999.

[22] Michael Myers. Revocation: Options and challenges. In Hirschfeld [11], pages 165–171.

[23] Michael Myers, Rich Ankney, Ambarish Malpani, Slava Galperin, and Carlisle Adams. *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol—OCSP*, March 1999.

[24] Moni Naor and Kobbi Nissim. Certificate revocation and certificate update. In *7th USENIX Security Symposium*, 1998.

[25] Ronald L. Rivest. Can we eliminate certificate revocation lists? In Hirschfeld [11], pages 178–183.

[26] Ronald L. Rivest and Butler Lampson. SDSI—a simple distributed security infrastructure. `theory.lcs.mit.edu/~cis/sdsi.html`, 1996.

[27] Stuart Stubblebine. Recent-secure authentication: Enforcing revocation in distributed systems. In *Proceedings of the 1995 IEEE Symposium on Research in Security and Privacy*, pages 224–235, 1995.

[28] Stuart Stubblebine and Rebecca N. Wright. An authentication logic supporting synchronization, revocation, and recency. In *Proceedings of the Third ACM Conference on Computer and Communications Security*, March 1996.

[29] Sean Turner and Alfred Arsenault. *Internet X.509 Public Key Infrastructure: PKIX Roadmap.* IETF, 1999. `www.ietf.org/internet-drafts/draft-ietf-pkix-roadmap-01.txt`.

[30] W. Yeong, T. Howes, and S. Kille. *Lightweight Directory Access Protocol.* IETF RFC 1777, 1995.

A Proof Sketches

A.1 Proof of the Monotonicity Theorem

Theorem 1 is proved by induction on the structure of e. The interesting case is given below.

- $e = \{e_1 \mid e_2 \notin e_3, \mathbf{g}\}$. Let $\nu_2 = [\![e_2]\!]\rho$, $\nu_2' = [\![e_2]\!]\rho'$, $\nu_3 = [\![e_3]\!]\rho$, and $\nu_3' = [\![e_3]\!]\rho'$.

  If $e : +$ and $\nu_2 \in \nu_3$, then $\nu = \{\}$. Then $\nu \sqsubseteq \nu'$ because ν' is a set by the definition of meaning, and $\{\}$ is the least set in the ordering $\sqsubseteq$.

  If $e : +$ and $\nu_2 \notin \nu_3$, then $\nu = [\![\{e_1 \mid \mathbf{g}\}]\!]\rho$. By induction and the polarity rules, $\nu_2 \sqsubseteq \nu_2'$ and $\nu_3 \sqsupseteq \nu_3'$. By the definition of meaning, $\nu_2, \nu_2' \in C$, so by Lemma 5, $\nu_2 = \nu_2'$. Also by the definition of meaning, ν_3, ν_3' are sets, so by Lemma 6, $\nu_3 \cap C \supseteq \nu_3' \cap C$. In particular, $\nu_2 = \nu_2' \notin \nu_3'$. By the definition of meaning, $\nu' = [\![\{e_1 \mid \mathbf{g}\}]\!]\rho'$. Then by induction, $\nu \sqsubseteq \nu'$, proving (1).

  If $e : -$ and $\nu_2' \in \nu_3'$, then $\nu' = \{\}$. Then $\nu \sqsupseteq \nu'$ because ν is a set by the definition of meaning, and $\{\}$ is the least set in the ordering $\sqsubseteq$.

If $e : -$ and $\nu_2' \not\in \nu_3'$, then $\nu' = [\![\{e_1 \mid \mathbf{g}\}]\!]\rho'$. By induction and the polarity rules, $\nu_2 \sqsupseteq \nu_2'$ and $\nu_3 \sqsubseteq \nu_3'$. By the definition of meaning, $\nu_2, \nu_2' \in C$, so by Lemma 5, $\nu_2 = \nu_2'$. Also by the definition of meaning, ν_3, ν_3' are sets, so by Lemma 6, $\nu_3 \cap C \subseteq \nu_3' \cap C$. In particular, $\nu_2' = \nu_2 \not\in \nu_3$. By the definition of meaning, $\nu = [\![\{e_1 \mid \mathbf{g}\}]\!]\rho$. Then by induction, $\nu \sqsupseteq \nu'$, proving (2).

Both ν and ν' are sets, so $\nu, \nu' \not\in C$, proving (3). □

Lemma 5 *If $\nu \sqsubseteq \nu'$, and $\nu \in C$ or $\nu' \in C$, then $\nu = \nu'$.* □

Lemma 6 *For any sets $\nu, \nu' \in U$, if $\nu \sqsubseteq \nu'$ then $\nu \cap C \subseteq \nu' \cap C$.* □

A.2 Proof of the Local Soundness Lemma

The proof of Lemma 2 is trivial for every rule but the last two rules, which require the following lemma.

Lemma 7 (Substitutivity) $[\![e]\!](\rho[x \mapsto [\![e']\!]\rho]) = [\![e[x \mapsto e']]\!]\rho$

Proof: By induction on the structure of e.

- $e = x$. Then $[\![x]\!](\rho[x \mapsto [\![e']\!]\rho]) = [\![e']\!]\rho = [\![x[x \mapsto e']]\!]\rho$.
- $e = y \neq x$. Then $[\![y]\!](\rho[x \mapsto [\![e']\!]\rho]) = [\![y]\!]\rho = [\![y[x \mapsto e']]\!]\rho$.
- $e = \{e_1 \mid y \in e_2, \mathbf{g}\}$. Then

  $$[\![e]\!](\rho[x \mapsto [\![e']\!]\rho]) \\ = \bigcup_{\nu \in [\![e_2]\!](\rho[x \mapsto [\![e']\!]\rho])} [\![e_1]\!](\rho[x \mapsto [\![e']\!]\rho][y \mapsto \nu]).$$

  By induction, this is equal to

  $$\bigcup_{\nu \in [\![e_2[x \mapsto v]]\!]\rho} [\![e_1]\!](\rho[x \mapsto [\![e']\!]\rho][y \mapsto \nu]). \qquad (*)$$

  We can assume that y is not x and is not free in e' (otherwise, rename y in e). Then

  $$\rho[x \mapsto [\![e']\!]\rho][y \mapsto \nu] = (\rho[y \mapsto \nu])[x \mapsto [\![e']\!](\rho[y \mapsto \nu])]$$

  Then by induction,

  $$[\![e_1]\!](\rho[x \mapsto [\![e']\!]\rho][y \mapsto \nu]) = [\![e_1[x \mapsto e']]\!](\rho[y \mapsto \nu])$$

  This shows that $(*)$ is equal to

  $$\bigcup_{\nu \in [\![e_2[x \mapsto v]]\!]\rho} [\![e_1[x \mapsto e']]\!](\rho[y \mapsto \nu]) = [\![e[x \mapsto e']]\!]\rho,$$

  as desired.

The remaining cases are proved similarly. □

A.3 Proof of the Parsing Lemma

Proof: First, note that no value contains a redex, so no value has a parse. This shows that (1), (2), and (3) are mutually exclusive. For the remainder, we show that every e falls into at least one of these cases, by induction on the structure of e.

- $e = x$. Then e is not a value, and it contains no redex, so we have case (3).
- $e = c$. Then e is a value, so we have case (1).
- $e = e'\$u$. By induction, it is sufficient to consider the following cases.
  - e' is a value. If e' is a principal K, then e is a redex and we have (2). Otherwise, e is not a redex and we have (3).
  - e' has a unique parse $E[e'']$. Then there is a unique context $E' = E\$u$ and redex e'' such that $e = E[e'']$, so we have (2).
  - e' is not a value and has no parse. Then we have (3).
- The other cases are proved similarly. □

A.4 Proof of the Soundness Theorem

Theorem 4 is proved by induction on the definition of $\Rightarrow_D^*$.

- If the reduction has length 0, then $e = e'$ and the result follows immediately.
- If the first step of the reduction does not involve job i, then the result is immediate by induction. Note, job i cannot be the response to a query in this reduction because it still appears at the end of the reduction; the set $\mathbf{i}$ would prevent it from being re-introduced.
- The reduction has the form

  $$\begin{aligned} &\mathsf{w}_j^i(E[e_1]), \mathbf{J}; \mathbf{i} \\ &\quad \Rightarrow_D \mathsf{w}_j^i(E[e_1']), \mathbf{J}; \mathbf{i} \\ &\quad \Rightarrow_D^* \mathsf{w}_j^i(e'), \mathbf{J}'; \mathbf{i}' \end{aligned}$$

  where $e = E[e_1]$ and $e_1 \to e_1'$.

  Then $[\![e_1]\!]\rho = [\![e_1']\!]\rho$ by the Local Soundess Lemma. It follows immediately that $[\![e]\!]\rho = [\![E[e_1]]\!]\rho = [\![E[e_1']]\!]\rho$; and by induction, $[\![e']\!]\rho \sqsubseteq_\gamma [\![E[e_1']]\!]\rho$, so that $[\![e']\!]\rho \sqsubseteq_\gamma [\![e]\!]\rho$ as desired.
- The reduction has the form

  $$\begin{aligned} &\mathsf{w}_j^i(E[K\$u^{\gamma_1}]), \mathbf{J}; \mathbf{i} \\ &\quad \Rightarrow_D \mathsf{w}_j^i(E[e_1]), \mathbf{J}; \mathbf{i} \\ &\quad \Rightarrow_D^* \mathsf{w}_j^i(e'), \mathbf{J}'; \mathbf{i}' \end{aligned}$$

  where $e = E[K\$u^{\gamma_1}]$ and $D(\mathsf{loc}(i), K, u^{\gamma_1}) = e_1$.

  Since $\rho \models D$, we have $[\![e_1]\!]\rho \sqsubseteq_{\gamma_1} [\![K\$u^{\gamma_1}]\!]\rho$. Then by Lemma 8, we have $[\![E[e_1]]\!]\rho \sqsubseteq_\gamma [\![E[K\$u^{\gamma_1}]]\!]\rho$. By induction, we have $[\![e']\!]\rho \sqsubseteq_\gamma [\![E[e_1]]\!]\rho$, so $[\![e']\!]\rho \sqsubseteq_\gamma [\![e]\!]\rho$ as desired.
- The reduction has the form

$$\begin{aligned}
&\mathsf{w}_j^i(E[K\$u]), \mathbf{J}; \mathbf{i}\\
&\qquad \Rightarrow_D \mathsf{w}_j^i(E[(K\$u)@K]), \mathbf{J}; \mathbf{i}\\
&\qquad \Rightarrow_D^* \mathsf{w}_j^i(e'), \mathbf{J}'; \mathbf{i}'
\end{aligned}$$

where $e = E[K\$u]$.

This follows because $[\![K\$u]\!] = [\![(K\$u)@K]\!]$.

- The reduction has the form

$$\begin{aligned}
&\mathsf{w}_j^i(E[e_1@K]), \mathbf{J}; \mathbf{i}\\
&\qquad \Rightarrow_D \mathsf{a}_j^i(E[e_1@K]), \mathsf{w}_i^{i'}(e_1), \mathbf{J}; i', \mathbf{i}\\
&\qquad \Rightarrow_D^* \mathsf{a}_j^i(E[e_1@K]), \mathsf{w}_i^{i'}(v), \mathbf{J}_1; \mathbf{i}_1\\
&\qquad \Rightarrow_D \mathsf{w}_j^i(E[v]), \mathbf{J}_1; \mathbf{i}_1\\
&\qquad \Rightarrow_D^* \mathsf{w}_j^i(e'), \mathbf{J}'; \mathbf{i}'.
\end{aligned}$$

By induction,

$$[\![v]\!]\rho \sqsubseteq_{\gamma_1} [\![e_1]\!]\rho = [\![e_1@K]\!]\rho.$$

Since $E[e_1@K] : \gamma$, there must be a polarity γ_1 such that $e_1 : \gamma_1$ and $E[x^{\gamma_1}] : \gamma$, where x^{γ_1} is a fresh variable. Then by Lemma 8,

$$[\![E[v]]\!]\rho \sqsubseteq_{\gamma} [\![E[e_1@K]]\!]\rho.$$

By induction, $[\![e']\!]\rho \sqsubseteq_{\gamma} [\![E[v]]\!]\rho$, so $[\![e']\!]\rho \sqsubseteq_{\gamma} [\![e]\!]\rho$, as desired.

- The remaining cases are proved similarly. □

Lemma 8 *If $[\![e]\!]\rho \sqsubseteq_{\gamma} [\![e']\!]\rho$ and $E[x^{\gamma}] : \gamma'$, where x^{γ} is not free in E, then $[\![E[e]]\!]\rho \sqsubseteq_{\gamma'} [\![E[e']]\!]\rho$.*

Proof: Let $\rho_1 = \rho[x^{\gamma} \mapsto [\![e]\!]\rho]$ and $\rho_2 = \rho[x^{\gamma} \mapsto [\![e']\!]\rho]$. Then $\rho_1 \sqsubseteq \rho_2$. By the Monotonicity Theorem,

$$[\![E[x^{\gamma}]]\!]\rho_1 \sqsubseteq_{\gamma'} [\![E[x^{\gamma}]]\!]\rho_2.$$

And by Substitutivity,

$$[\![E[x^{\gamma}]]\!]\rho_1 = [\![(E[x^{\gamma}])[x^{\gamma} \mapsto e]]\!]\rho = [\![E[e]]\!]\rho,$$

and similarly,

$$[\![E[x^{\gamma}]]\!]\rho_2 = [\![E[e']]\!]\rho.$$

Then we have

$$[\![E[e]]\!]\rho \sqsubseteq_{\gamma'} [\![E[e']]\!]\rho,$$

as desired. □