Certificate Chain Expressions

Angela Roles
University of Texas Department of Computer Sciences

Abstract

Since the traditional sense of certificates in networks uses only the identity of the issuer, the identity of the subject, and the public key of the subject, there is no way to limit the use of an issued certificate. However, there may be cases in which we also wish a certificate to express limited "trust" of other nodes in the network. Thus, our research extends a certificate to also contain a certificate chain expression which indicates the situations in which the current certificate may be chained with other certificates. We define a calculus for these certificate chain expressions and present some properties, and then we illustrate the properties with a specific certificate graph.

1. Motivation

Consider a network in which each node a has a private key R_a and a public key B_a. In this network, if node a wishes to send a message m to another node c, node a must encrypt the message using B_c before sending the encrypted message to node c. We denote the encrypted message by $B_c<m>$.

If node a knows the public key B_c of another node c in the network, node a can issue a certificate from a to c, identifying the public key of node c. Then, if another node d knows the public key of node a, this node d can use the certificate from a to c to obtain the public key of node c.

As in [1], a certificate from node a to node c is signed by the private key of a and contains three items: the identity of certificate issuer a, the identity of certificate subject c, and the public key, B_c, of the certificate subject c. The certificate can thus be expressed in the following form:

$$R_a<a, c, B_c>$$

Any node that already knows the public key of a can use B_a to decrypt the certificate and thereby obtain the public key of c, B_c.

In a more compact form, we can represent a certificate from node a to node c as (a, c).

Using the certificates issued by each node in a given network, we can form a certificate graph. This graph is a directed graph in which each node in the graph is a node in the network. Given two nodes u and v in the graph, a directed edge from u to v represents a

certificate issued by *u* for *v* in the network.

For instance, consider the certificate graph in Figure 1.

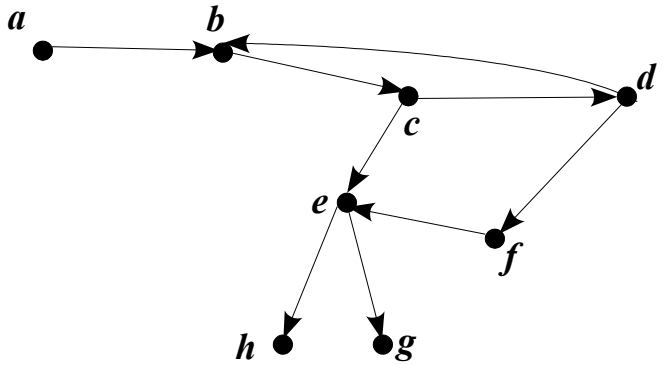

Figure 1. A sample certificate graph

From this graph, we see that the nodes in the network are *a*, *b*, *c*, *d*, *e*, *f*, *g*, and *h*. The certificates shown on the graph, denoted in the compact form, are (*a*, *b*), (*b*, *c*), (*c*, *d*), (*c*, *e*), (*d*, *b*), (*d*, *f*), (*e*, *g*), (*e*, *h*), and (*f*, *e*).

Certificates of the form described above allow us to define a network in which a node can chain any number of certificates together to obtain public keys of other nodes. There is no indication of how a particular certificate may be used, and thus, there is no way to limit the use of an issued certificate. As there may be cases in which we wish a certificate to express limited "trust" of other nodes in the network, the ability to indicate the situations in which a certificate may be used would prove useful.

For instance, we may wish to limit the length of the certificate chain with which a certificate may be chained. We may wish to indicate that a certificate can be chained only with certificates from particular other nodes. We may even wish to specify that an issued certificate cannot be chained with *any* successive certificates.

Hence, we consider extending the certificate to include one more item: a "certificate chain expression" to describe the set of chained certificates with which the current certificate can be chained. Our extended certificate now takes the following form:

$$R_a<a, c, CE, B_c>$$

In this certificate, *CE* denotes the certificate chain expression. The language and calculus for *CE* will be described in the next section.

2. A Calculus for Certificate Chain Expressions

In this section, we present a calculus for the definition of certificate chain expressions. Our calculus should be as simple as possible, but needs to remain expressive. We hope to present an approach that is both meaningful and useful.

2.1 Definitions

First, we need to use the identifiers of individual nodes, so these become primitives. We would also like to have a notation to mean "any node." In the case that we wish a certificate chain expression to specify that the current certificate may be chained with another certificate from "any additional node," it is much easier and more compact to have a single construct to specify this than it is to specify every single possible node. Finally, we wish to have an "empty node" indicator, for the case in which we do not wish to allow our certificate to be chained with certificates provided by any other node.

Thus, we define the following primitives for our calculus of certificate expressions:

a, b, c,...	individual node identifiers
–	any node
ϕ	empty node

Now, we wish to define operators to use with these primitives. First, we need a concatenation operation, as this is the only way to denote chains of nodes. Likewise, we need an operation to specify that any one of several possible chains is trusted; for this, we use a vertical bar to represent an "or." Finally, we would like a means of grouping or association.

Hence, we define the following operators:

•	concatenation
|	or
( )	association/grouping

We may also wish to use the concept of an arbitrary chain in our certificate chain expressions. So, we include a * ("wildcard") construct, which we take to mean the following:

*	any chain of nodes

The * is meaningful due to the existence of and the need for the concatenation operator. It is the most general certificate chain expression, because it admits any chain of any length. Thus, using this construct, we can still very simply represent the functionality of the original certificate definition without certificate chain expressions. The original certificate denoted by $R_a<a, c, B_c>$ would become $R_a<a, c, *, B_c>$.

2.2 Syntax

We can now summarize the syntax for certificate chain expressions as follows:

<*CE*> ::= *a* || *b* || *c* || ... ||
– ||
ϕ ||
<*CE*>•<*CE*> ||
<*CE*> | <*CE*> ||
*

Hence, a *CE* may consist of a single node, a –, a ϕ , a concatenation of two certificate expressions, an "or" (conjunction) of two certificate chain expressions, or a *.

We now give examples of several certificate chain expressions and discuss their meanings.

a•*b*•*c* where *a*, *b*, and *c* are nodes would denote a simple chain of nodes. We define this chain to also include the subexpressions *a* and *a*•*b*. The certificate could also be used alone, unchained with any other certificates.

–•–•– means that the current certificate can chain with any chain of certificates of length 3 or less. So, the current certificate would also chain with any chain of certificates of length 2 or with single certificates.

d•*f* | *e*•*h* means the current certificate can be chained with other chains of certificates that satisfy either *d*•*f* or *e*•*h*. Note that any given chain can satisfy only one of these two expressions, since the set of nodes in each is disjoint from the other.

c•*e*•(*g* | *h*) means that the current certificate can be chained with other chains of certificates that satisfy either *c*•*e*•*g* or *c*•*e*•*h*.

The final three of these expressions will be used in various certificates in Section 4.

2.3 Semantics

We now associate semantics with certificate chain expressions, whose syntax we have just described.

Associated with each certificate chain expression *E* is a set *S* [*E*] of certificate chains. This set defines the semantics of a chain expression.

We have:

$S[a]=\{\epsilon, a\}$ for individual node a
$S[-] = \{ \epsilon, a, b, c, ... \}$
$S[\phi]=\{\epsilon\}$
$S[E \bullet E'] = S[E] \circ S[E']$
$S[E \mid E'] = S[E] \cup S[E']$
$S[*]$ = set of all possible certificate chains

We say that two certificate chain expressions E and E' are *equivalent*, denoted $E = E'$, if and only if $S[E] = S[E']$.

In the above, we use the $\circ$ to indicate an additional set operation. This operator is similar to a Cartesian product, but uses concatenation rather than ordered pairs. So, for instance, if we had an $S[E] = \{ \epsilon, a, b \}$ and $S[E'] = \{ \epsilon, d, e \}$, then we would have

$$S[E] \circ S[E'] = \{ \epsilon, a, b \} \circ \{ \epsilon, d, f \}$$
$$= \{ \epsilon\bullet\epsilon, \epsilon\bullet d, \epsilon\bullet f, a\bullet\epsilon, a\bullet d, a\bullet f, b\bullet\epsilon, b\bullet d, b\bullet f \}$$

Since Cartesian product is a distributive operation, it is clear that $\circ$ is also distributive.

Our definitions below for the empty element (ϵ) would allow us to reduce the above set further.

We make the following definitions:

$a\bullet\epsilon = a$
$\epsilon\bullet a = \epsilon$
$\epsilon\bullet\epsilon = \epsilon$

We add one further definition governing the functionality of ϵ when involved in a chain. We establish that when present as a member of a chain, ϵ serves to "truncate" that chain. For instance, if we have a chain $a\bullet\ \epsilon\bullet\ c$, then

$a\bullet\epsilon\bullet c = a$

$S[E]$ for a chain of nodes should be calculated before the above identities are applied. So, if $E = a\bullet b\bullet c$, then we get

$$S[E] = S[a\bullet b\bullet c] = S[a] \circ S[b] \circ S[c] = \{\epsilon, a\} \circ \{\epsilon, b\} \circ \{\epsilon, c\}$$
$$= \{ \epsilon\bullet\epsilon\bullet\epsilon,\ \epsilon\bullet\epsilon\bullet c,\ \epsilon\bullet b\bullet\epsilon,\ \epsilon\bullet b\bullet c,\ a\bullet\epsilon\bullet\epsilon,\ a\bullet\epsilon\bullet c,\ a\bullet b\bullet\epsilon,\ a\bullet b\bullet c \}$$
$$= \{ \epsilon,\ a,\ a\bullet b,\ a\bullet b\bullet c \}$$

2.4 Rules

For our calculus of certificate chain expressions to be useful, we would like the expressions to be as simple as possible. Additionally, we need to be able to test two certificate chain expressions for equivalence, as this gives us the means of finding out if a

certain certificate chain expression includes particular chain.

Thus, we establish some basic rules to aid us in the simplification and comparison of certificate chain expressions, and we prove these rules using our semantics.

Rule 0:
(a) | is commutative, associative, and reflexive
(b) • is left distributive over | and right distributive over |

Proof:
Consider the three certificate chain expressions E, E', and E''.
Part (a): Using the semantics of Section 2.3, we have:

$S[E \mid E'] = S[E] \cup S[E']$ { definition of $S[e \mid e']$ }
$= S[E'] \cup S[E]$ { commutativity of union }
$= S[E' \mid E]$ { definition of $S[e' \mid e]$ }
So | is commutative.

$S[(E \mid E') \mid E''] = S[E \mid E'] \cup S[E'']$ { definition of S[(E | E') | E''] }
$= (S[E] \cup S[E']) \cup S[E'']$ { definition of $S[E \mid E']$ }
$= S[E] \cup (S[E'] \cup S[E''])$ { associativity of union }
$= S[E] \cup S[E' \mid E'']$ { definition of $S[E' \mid E'']$ }
$= S[E \mid (E' \mid E'')]$ { definition of S $[E \mid (E' \mid E'')]$ }
So | is associative.

$S[E \mid E] = S[E] \cup S[E]$ { definition of $S[E \mid E]$ }
$= S[E]$ { reflexivity of union }
So | is reflexive.

Part (b): Using the semantics of Section 2.3, we have:

$S[E\bullet(E' \mid E'')] = S[E] \circ S[E' \mid E'']$ { definition of $S[E\bullet(E' \mid E'')]$ }
$= S[E] \circ (S[E'] \cup S[E''])$ { definition of $S[E' \mid E'']$ }
$= (S[E] \circ S[E']) \cup (S[E] \circ S[E''])$ { concat. distributes over union }
$= S[(E\bullet E') \mid (E\bullet E'')]$ { definition of $S[(E\bullet E') \mid (E\bullet E'')]$ }

and

$S[(E \mid E')\bullet E''] = S[E \mid E'] \circ S[E'']$ { definition of $S[(E \mid E')\bullet E']$ }
$= (S[E] \cup S[E']) \circ S[E'']$ { definition of $S[E \mid E']$ }
$= (S[E] \circ S[E'']) \cup (S[E'] \circ S[E''])$ { concat. distributes over union }
$= S[(E\bullet E'') \mid (E'\bullet E'')]$ { definition of $S[(E\bullet E'') \mid (E'\bullet E'')]$ }

So, • is left distributive and right distributive over |.

□

Rule 1: Rule for _
– | a = –

Proof: Using the semantics of Section 2.3, we have the following:

S [– | a]	= S [–] ∪ S [a]	{ definition of S [– | a] }
	= { ϵ, a, b, c, ... } ∪ { ϵ, a }	{ definitions of S [–] and S [a] }
	= { ϵ, a, b, c, ... }	{ definition of union }
	= S [–]	{ definition of S [–] }

□

Logically, *Rule 1* makes sense. If any arbitrary node is to be trusted, then clearly, a specific node *a* should be trusted.

Rule 2: Rule for *
For an arbitrary certificate chain expression E,
* | E = *

Proof: Let E be an arbitrary certificate chain expression. Again using the semantics of Section 2.3, we have the following:

S [* | E]	= S [*] ∪ S [E]	{ definition of S [* | E] }

Since S [*] is the set of all possible certificate chains, any chain specified in S [E] is already contained in S [*]. Thus, because of the properties of sets and the union operation, we have

S [* | E]	= S [*]

□

Again, this rule is logical. If any arbitrary chain of any length is to be trusted, than obviously, any chain specified in a specific certificate chain expression is to be trusted.

Now, consider the uses ϕ. This represents the empty node (or a chain of zero nodes). Thus, given a certificate R_x<x, y, a• ϕ , B_y>, the certificate could chain with a certificate from node y to node a and nothing else. So, if an expression consists of a nonempty node concatenated with an empty node, this is equivalent to simply the nonempty node, and we can more concisely express the certificate as R_x<x, y, a, B_y>.

Similarly, if we have ϕ •a, there is no nonempty chain to satisfy the expression. So, this case reduces to ϕ.

Finally, given a certificate R_x<x, y, ϕ | a, B_y>, we know that in addition to paths specified in the certificate chain expression, the certificate can also be used alone. So we have an equivalent certificate simply by saying R_x<x, y, a, B_y>.

These comments lead us to our third rule.

Rule 3: Rules for ϕ
(a) $a\bullet\ \phi = a$
(b) $\phi\ \bullet a = \phi$
(c) $\phi \mid a = a$

Proof:
Part (a): From our semantics, we have:

$S[a\bullet\ \phi\ ]$	$= S[a]\bullet S[\ \phi\ ]$	{ definition of $S[a\bullet\ \phi\ ]$ }
	$= \{\epsilon, a\}\bullet\{\epsilon\}$	{ definitions of $S[a]$ and $S[\ \phi\ ]$ }
	$= \{\epsilon, a\}$	{ $a\bullet\ \epsilon = a$ and $\epsilon\bullet\ \epsilon = \epsilon$ }
	$= S[a]$	{ definition of $S[a]$ }

Part (b): Again from our semantics:

$S[\ \phi\ \bullet a]$	$= S[\ \phi\ ]\bullet S[a]$	{ definition of $S[\ \phi\ \bullet a]$ }
	$= \{\epsilon\}\bullet\{\epsilon, a\}$	{ definitions of $S[\ \phi\ ]$ and $S[a]$ }
	$= \{\epsilon\}$	{ $\epsilon\bullet\ a = \epsilon$ for any a (including ϵ) }
	$= S[\ \phi\ ]$	{ definition of $S[a]$ }

Part (c): Once again from our semantics:

$S[\ \phi \mid a]$	$= S[\ \phi\ ] \cup S[a]$	{ definition of $S[\ \phi \mid a]$ }
	$= \{\epsilon\}\ \cup\ \{\epsilon, a\}$	{ definitions of $S[\ \phi\ ]$ and $S[a]$ }
	$= \{\epsilon, a\}$	{ definition of union }
	$= S[a]$	{ definition of $S[a]$ }

□

Note that *Rule 3* is stated in terms of single nodes. We can extend *Rule 3* to indicate the behavior of ϕ with an arbitrary certificate chain expression *E*.

Rule 4:
For an arbitrary certificate chain expression *e*,
(a) $E\bullet\ \phi = E$
(b) $\phi\ \bullet E = \phi$
(c) $\phi \mid E = E$

Proof:
Part (a): We have

$$S\,[E\bullet\ \phi\ ] = S\,[E]\bullet S\,[\ \phi\ ] = S\,[E]\bullet \{\epsilon\}$$

from our semantics. Since

$$a\bullet\ \epsilon = a \text{ and } \epsilon\bullet\ \epsilon = \epsilon$$

we have that for any chain C in $S\,[E]$,

$$C\bullet\ \epsilon = C$$

Since any chain in $S\,[E]\bullet \{\epsilon\}$ will be of the above form, we simply get

$$S\,[E]\bullet \{\epsilon\} = S\,[E]$$

Part (b): We have

$$S\,[\ \phi\ \bullet E] = S\,[\ \phi\ ]\bullet S\,[E] = \{\epsilon\}\bullet S\,[E]$$

from our semantics. Since

$$\epsilon\bullet\ a = \epsilon$$

We have that for any chain C in $S\,[E]$,

$$\epsilon\bullet C = \epsilon$$

Since any chain in $\{\epsilon\}\bullet S\,[E]$ will be of the above form, we simply get

$$\{\epsilon\}\bullet S\,[E] = \{\epsilon\} = S\,[\ \phi\ ]$$

Part (c): We have

$$S\,[\ \phi \mid E] = S\,[\ \phi\ ] \cup S\,[E] = \{\epsilon\} \cup S\,[E]$$

Thus, we need to show that any $S\,[E]$ contains the element ϵ.

If E is a single node, a –, or a ϕ , then $S\,[E]$ clearly contains ϵ by definition.

If E is a single chain consisting only of single nodes, – constructs, or ϕ 's, then an element of $S\,[E]$ will be $\epsilon\bullet...\bullet\epsilon = \epsilon$. So, $S\,[E]$ will contain ϵ in this case as well.

If E is *, then $S\,[E]$ is the set of all possible chains, and this includes an "empty" chain ϵ. Thus, $S\,[E]$ also contains ϵ in this case. Note that S[*] can be expressed as a set of simple chains.

In any other case, $S\,[E]$ will be a set of simple chains. For any chain C in $S\,[E]$, $S\,[C]$ contains ϵ as established in the "single simple chain" case. Thus, $S\,[E]$ must contain ϵ, since it must be the case that E can be expressed as an "or" of simple chains.

So, for any E, $S\,[E]$ must contain e. Then

$$\{\epsilon\} \cup S[E] = S[E]$$

and thus

$$S[\phi \mid E] = S[E]$$

□

3. Properties

In addition to the rules supported by our semantics, we would like to establish some properties that can be proven using our rules. Since simple, expressive certificate chain expressions are our goal, the properties are especially useful for reducing the number of redundant subexpressions present. We give two examples.

First, if we have a certificate $R_d\langle d, b, -\bullet- \mid c\bullet e, B_b\rangle$, we recognize that $S[c\bullet e]$ is already a subset of $S[-\bullet-]$. So, we can actually express this certificate more simply as $R_d\langle d, b, -\bullet-, B_b\rangle$. Similarly, we intuitively recognize that for arbitrary node a,

$-\bullet- \mid a\bullet- = -\bullet-$

and $-\bullet- \mid -\bullet a = -\bullet-$

The above provides the reasoning for our first three properties.

Property 1

$-\bullet- \mid a\bullet- = -\bullet-$

Proof

$-\bullet- \mid a\bullet-$	$= (- \mid a)\bullet-$	{ *Rule 0(b)* }
	$= -\bullet-$	{ *Rule 1* }

□

Property 2

$-\bullet- \mid -\bullet c = -\bullet-$

Proof

$-\bullet- \mid c$	$= -\bullet(- \mid c)$	{ *Rule 0(b)* }
	$= -\bullet-$	{ *Rule 1* }

□

Intuitively, we can suggest a stronger statement: that $-\bullet-$ "contains" $a\bullet c$ for two specific but arbitrary nodes a and c. Using the first two properties, we can now state and prove *Property 3*, which we suspected previously from the definition of –.

Property 3
$-\bullet- \mid a \bullet c = -\bullet-$

Proof

$-\bullet- \mid a \bullet c = (-\bullet- \mid -\bullet c) \mid a \bullet c$ { *Property 2* }
$= -\bullet- \mid (-\bullet c \mid a \bullet c)$ { *Rule 0(a)* }
$= -\bullet- \mid ((- \mid a) \bullet c)$ { *Rule 0(b)* }
$= -\bullet- \mid (-\bullet c)$ { *Rule 1* }
$= -\bullet-$ { *Property 2* }

□

Now, for a second example, consider the certificate $R_b\langle b, c, d \bullet f \mid e \bullet h \mid e, B_c \rangle$. Specifically, consider the certificate chain expression $e \bullet h \mid e$. This is clearly redundant, so we would like to be able to reduce it to $e \bullet h$. So, we again note that $S[e]$ is a subset of $S[e.h]$. Our certificate therefore becomes $R_b\langle b, c, d \bullet f \mid e \bullet h, B_c \rangle$.

Using *Rules 4(a)* and *4(c)* from Section 2.4, we can attain the property for chains mentioned above and in Section 2.1. So, for instance, if we consider $a \bullet b \bullet c$, where a, b, and c are all nodes, then

$a \bullet b \bullet c \mid a \mid a \bullet b = a \bullet (\phi \mid b \bullet (\phi \mid c)$
$= a \bullet (\phi \mid b \bullet c)$
$= a \bullet b \bullet c$

A more general form of this is formally proven below.

Property 4
For two certificate chain expressions E and E':
$E \bullet E' \mid E = E \bullet E'$

Proof: We have

$E \bullet E' \mid E = E \bullet E' \mid E \bullet \phi$ { *Rule 4(a)* }
$= E \bullet (E' \mid \phi)$ { *Rule 0(b)* ($\bullet$ distributive) }
$= E \bullet E$ { *Rule 4(c)* }

□

4. Example

We consider the following certificate graph.

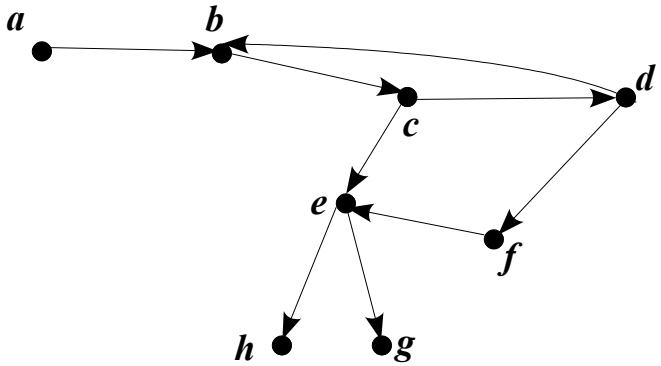

Figure 2. Certificate graph revisited

While the above graph tells us which certificates are available, it does not show the certificate chain expressions included in those certificates. So, we list the certificates below with the certificate chain expressions included.

R_a<a, b, $c \bullet e \bullet (g \mid h)$, B_b>
R_b<b, c, $d \bullet f \mid e \bullet h$, B_c>
R_c<c, d, $- \bullet - \bullet -$, B_d>
R_c<c, e, *, B_e>
R_d<d, b, $- \bullet -$, B_b>
R_d<d, f, e, B_f>
R_e<e, g, $-$, B_g>
R_e<e, h, ϕ , B_h>
R_f<f, e, g, B_e>

Using these certificates, we are ready to establish the chains of certificates with which each other certificate may be chained. For ease of presentation, when writing chains of certificates, we use the more compact certificate notation described in Section 1, where we denote a certificate from node u to node v by (u, v). We still assume that the certificate chain expressions included in the certificates above are present.

Chains beginning with certificate (a, b)

Consider the first certificate, R_a<a, b, $c \bullet e \bullet (g \mid h)$, B_b>, which we denote by (a, b). Since the first node in the certificate chain expression is c, we know that

(a, b) (b, c)

is a permitted chain in this case. Next, we must look at the certificate chain expression of (b, c), which is $d \bullet f \mid e \bullet h$. From the certificate chain expression for (a, b), we now see that only

(a, b) (b, c) (c, e) (e, h)

We note that (c, e) allows itself to be chained with any chain of any length, but (e, h) allows itself to be chained with nothing. Hence, the above is the longest possible chain beginning with (a, b) that can be used.

Chains beginning with certificate (b, c)

The chain expression associated with this certificate is $d \bullet f \mid e \bullet h$. So, from this alone and according to the certificate graph, we know that we can have as chains at most

(b, c) (c, d) (d, f)
and (b, c) (c, e) (e, h)

The certificate (c, d) can chain with (d, f), so we keep the first chain. The certificate (c, e) can chain with anything, and this includes (e, h), so the second chain can be kept as well. Note that even without the limit imposed by the certificate chain expression for (b, c), the second chain could still not be extended since (e, h) has a certificate chain expression of ϕ.

Chains beginning with certificate (c, d)

The certificate (c, d) itself can chain with any chain of length 3. From the certificate graph, we see that node d issues certificates (d, f) and (d, b).

First, consider certificate (d, f). This certificate can chain only with (f, e). So we get

(c, d) (d, f) (f, e)

Now consider certificate (d, b). Chaining with (b, c) would create a cycle, so we have only

(c, d) (d, b)

Chains beginning with certificate (c, e)

Certificate (c, e) can chain with any chain of any length. From the certificate graph, we see that node e issues certificates (e, g) and (e, h). So, we have

(c, e) (e, g)
and (c, e) (e, h)

In the graph, neither of nodes g and h issue any certificates. If node g did issue a certificate, that certificate could be added onto the first chain. However, if node h did

issue a certificate, the second chain could not be extended, because the certificate chain expression for (*e*, *h*) is ϕ .

Chains beginning with certificate (d, b)

The certificate (*d*, *b*) can chain with any chain of length 2. Node *b* issues only one certificate: (*b*, *c*). For a second certificate to chain, we note that (*b*, *c*) could chain with (*c*, *d*) or (*c*, *e*). Thus, we have

(*d*, *b*) (*b*, *c*) (*c*, *d*)

and (*d*, *b*) (*b*, *c*) (*c*, *e*)

Chains beginning with certificate (d, f)

The only certificate with which (*d*, *f*) will chain is (*f*, *e*). So we have:

(*d*, *f*) (*f*, *e*)

Chains beginning with certificate (e, g)

Certificate (*e*, *g*) will chain with a single additional certificate. Since node *g* does not issue any certificates in the graph, though, there is no certificate with which to chain.

Chains beginning with certificate (e, h)

Certificate (*e*, *h*) will not chain with any certificate at all. So, it may only be used alone or at the end of a chain.

Chains beginning with certificate (f, e)

From the graph and the certificate chain expression for (*f*, *e*), (*f*, *e*) will chain only with (*e*, *g*). So we have

(*f*, *e*) (*e*, *g*)

5. Conclusion

In this paper, we outline the need for certificate chain expressions in certificates. We present a syntax and a set of semantics for certificate chain expressions. We also establish rules and properties for use in reducing certificate chain expressions. Finally, we illustrate our work with a sample certificate graph and the certificates for this graph, with certificate chain expressions included.

Future work is possible to formalize the combination of certificate chain expressions for multiple certificates when calculating the chains of certificates with which the current certificate can chain. Also, work is needed to investigate the adaptation of certificate dispersal algorithms to take into account information from certificate chain expressions.

References

[1] M.G. Gouda and E. Jung. Certificate Dispersal in Ad-Hoc Networks. ICDS 2004.