A State-Based Model for Certificate Management Systems

Chuchang Liu, Maris A. Ozols, Marie Henderson, and Tony Cant

Information Technology Division
Defence Science and Technology Organisation
PO Box 1500, Salisbury
SA 5108, Australia
{Chuchang.Liu,Maris.Ozols,Marie.Henderson,Tony.Cant}@dsto.defence.gov.au

Abstract. A Certificate Management System (CMS) is used to generate, distribute, store and verify certificates. It supports secure electronic communication through its functions. This paper presents a state-based model for certificate management systems. The axiomatization of CMS structures and the security policy followed by CMSs is discussed. The main functions of a CMS, including certificate issuing, certificate revocation and certificate rekeying, are formally described through transitions that change states of the CMS. A major CMS client function, certificate verification, is also formally discussed. With this model, an approach to the formal specification of the structure and behavior of a CMS is provided. The approach is very general, and would be useful in guiding the developer and the evaluator of a CMS with the design, analysis and implementation of the system.

Keywords: certificate, CA (Certification Authority), certificate management systems, certificate verification, information security, formal methods.

1 Introduction

Certificates have been used with security applications to support the validation of digital signatures on documents that are exchanged through computer networks. Recently, there has been substantial interest in public key technologies [2,3,4] being used to support secure electronic communications. In order to use public key cryptography, it is necessary to make an entity's public key available to others in such a way that its authenticity (i.e., its status as the true public key of that entity) and validity are verifiable. Public key certificates can be viewed as a vehicle by which public keys may be stored, distributed or forwarded over unsecured media while minimising the danger of undetectable manipulation. Anyone who wants to use a certificate must have a valid copy of the public key of the Certification Authority (CA) who issued the certificate, and must trust the CA.

A Certificate Management System (CMS) can provide mechanisms for managing public key certificates through its administration functions. It is used to

H. Imai, Y. Zheng (Eds.): PKC 2000, LNCS 1751, pp. 75–92, 2000.

generate, distribute, store and verify certificates. Practical discussions on the structure of a CMS, its establishment, as well as functions and protocols between its components can be found in a number of research papers and references, such as in Kapidzic [6,7], Trcek [11], and PKIX Working Group Internet Draft [5].

As is well known, formal methods can provide strict proof technologies for verifying critical properties of a security system in a precise and unambiguous way, and also guide the developer towards a design of the security architecture of the system and its implementation. Formal techniques have been used for specifying authorization requirements [12] and analysing security properties of cryptographic protocols [10]. Formal methods have also been applied in the design of security devices. However, there has been a distinct lack of formal methods and techniques which can be used for the description of the structure and the behavior of CMSs. Therefore, theoretical studies of CMSs that may lead to a formal specification for the structure, functions and security properties of such systems are highly desirable.

In this paper, we present a state-based model for CMSs. One of our contributions is to propose an approach to the formalization of CMS structures and the security policy. The axiomatization of the topology structure and states of a CMS is discussed, and a general security policy followed by CMSs is also formalized. The other contribution is to provide a formal description for the main functions of a CMS, including certificate issuing, certificate revocation and certificate rekeying. All these function are described as transitions that change states of the CMS. A major CMS client function, certificate verification, is also formally discussed.

Our discussion is based on certificate management systems with top-down hierarchical structure, but the method is very general. Therefore, the methods and techniques proposed in this paper are suitable for any other kind of certificate management systems, and they would be useful in guiding the developer and the evaluator of a CMS with the design, analysis and implementation of the system.

The rest of the paper is structured as follows. Section 2 introduces the state-based model for CMSs, and gives a formal definition. Section 3 discusses the axiomatization of the CMS security policy, including certificate issuing policy, access control policy and trust policy. Section 4 formalizes the main CMS functions – certificate issuing, certificate revocation, and certificate rekeying – by defining appropriate transitions. The certificate verification function is discussed in Section 5, where a number of verification rules are presented. The last section concludes this paper with a short discussion about future work.

2 The Model

In this section, we present the state-based model for certificate management systems. An axiomatization of the CMS structure is discussed, and a formal definition of a CMS is given.

2.1 Certification Topology

We classify the entities or agents in a CMS into two classes, *Certification Authorities* (CAs) and *users*. CAs can have their own certificates, and they also issue certificates for others within the CMS. Users, also called *end entities*, are people or devices that may hold certificates issued by some CAs, but cannot issue any certificates themselves.

A user of a security service requiring knowledge of a public key generally needs to obtain and validate the certificate containing the required public key. If the user does not already hold an assured copy of the public key of the CA who signed the certificate, then he might need an additional certificate to obtain that public key. In general, a chain of multiple certificates may be needed, comprising the certificate of the public key owner signed by one CA, and zero or more additional certificates of CAs issued by other CAs. Such chains of certificates are called *certificate paths*.

There are different ways in which CMS entities might be configured so that public key users are able to find certificate paths. For instance, RFC 1422 [8] defines a rigid hierarchical structure for the Internet Privacy Enhanced Mail (PEM) [5], in which there are three types of PEM CAs: Internet Policy Registration Authority (IPRA) acts as the root of the PEM certification hierarchy at level 0, and issues certificates only for the next level of authorities, called PCAs (Policy Certification Authorities); PCAs, at level 1 of the PEM certification hierarchy, take the responsibility for establishing and publishing the certification policy with respect certifying users or subordinate certification authorities; and CAs, which are at level 2 of the hierarchy and can also be at lower levels (those at level 2 are certified by PCAs).

With our intention being to discuss CMSs in general, we assume that all the entities of a CMS, we call them agents, form a single root top-down certification hierarchy. This means that there is a special CA at the root, called the PAA (Policy Approval Authority), whose role is different from that of the other CAs. The PAA can certify others (but no one certifies it) and it holds a certificate which is trusted by everyone. All CAs can only issue certificates to their children, and no certificates are issued by users. In our model, the top-down certification hierarchy is formally described using a series of agent axioms.

Definition 1 *Let Ω be a set of agents (including a special agent, called the* PAA*) and $\downarrow$ a binary relation over Ω. Then $\langle \Omega, \downarrow \rangle$ is called a* certification topology *if it satisfies the agent axioms AA1 – AA6 given below.*

Agent Axioms

(AA1) $\forall \mathtt{X} \bullet (\mathtt{IsCA}(\mathtt{X}) \lor \mathtt{IsUsr}(\mathtt{X}))$
(AA2) $\forall \mathtt{X} \bullet (\mathtt{IsCA}(\mathtt{X}) \leftrightarrow \exists \mathtt{Y} \bullet (\mathtt{X} \downarrow \mathtt{Y}))$
(AA3) $\forall \mathtt{X} \bullet (\mathtt{IsUsr}(\mathtt{X}) \leftrightarrow \forall \mathtt{Y} \bullet \neg(\mathtt{X} \downarrow \mathtt{Y}))$
(AA4) $\forall \mathtt{X} \bullet (\mathtt{X} = \mathtt{PAA} \leftrightarrow (\mathtt{IsCA}(\mathtt{X}) \land \neg\exists \mathtt{Y} \bullet (\mathtt{Y} \downarrow \mathtt{X})))$
(AA5) $\forall \mathtt{X} \bullet \neg(\mathtt{X} \downarrow \mathtt{X})$
(AA6) $\forall \mathtt{X} \bullet \forall \mathtt{Y} \bullet (\mathtt{X} \downarrow^+ \mathtt{Y} \rightarrow \neg(\mathtt{Y} \downarrow^+ \mathtt{X}))$

Here, `IsCA(X)` means `X` is a CA, and `IsUsr(X)` means `X` is a user. The axiom AA6 includes a new relation operator $\downarrow^+$ as a transitive closure of the relation $\downarrow$, which is defined by the following formula:

$$\forall \mathtt{X} \bullet \forall \mathtt{Y} \bullet (\mathtt{X} \downarrow^+ \mathtt{Y} \leftrightarrow (\mathtt{X} \downarrow \mathtt{Y}) \vee \exists \mathtt{Z} \bullet (\mathtt{X} \downarrow^+ \mathtt{Z} \wedge (\mathtt{Z} \downarrow^+ \mathtt{Y}))$$

Note that axioms AA2 and AA4 together have ruled out the trivial case that Ω has only one agent, the PAA. Therefore, in this structure, the PAA is always a CA.

A certification topology has an intuitive direct graphical representation –all agents (i.e., CAs and users) are represented as nodes and, for any $\mathtt{X}, \mathtt{Y} \in \Omega$, there is a directed edge from `X` to `Y` if and only if $\mathtt{X} \downarrow \mathtt{Y}$ holds (in the case, we say `X` is a *parent* of `Y` or `Y` is a *child* of `X`). The PAA as a special agent at the root has no parents. An example of a certificate topology is shown in Figure 1.

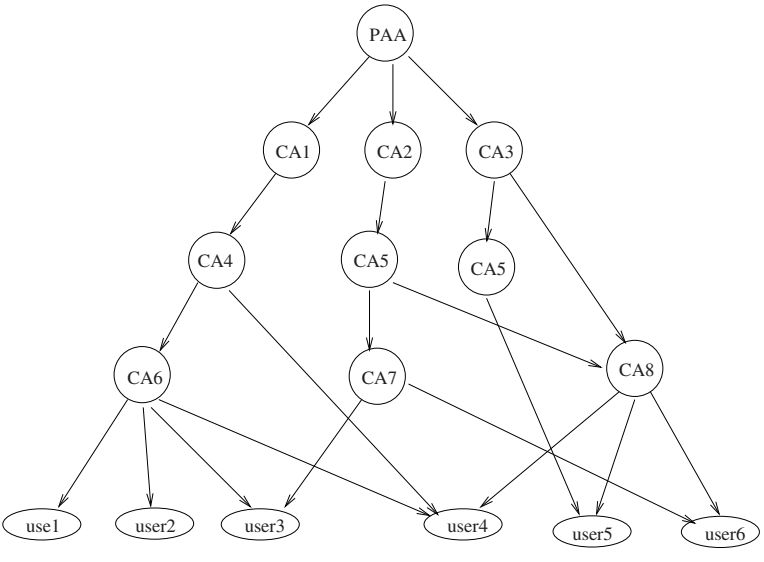

Fig. 1. A certification topology

The topology gives a trust model for certificate management. A search for finding a certificate path, as we will see, can be done based on the certification topology. In fact, certificate paths correspond to directed paths in the graphical representation of the topology. Based on the topology axioms, it is not difficult to show that, for any CA or any user, there exists at least one directed path to it from the PAA.

2.2 Certificates

A public-key certificate is a data structure consisting of a *data* part and a *signature* part. The data part contains, as a minimum, a public key and a string identifying the party (*subject*) to be associated therewith and the *issuer* of the certificate. The signature part consists of the digital signature of the issuer over the data part, thereby binding the subject's identity to the specified public key.

The certificates we consider have a standard public-key certificate format with the basic information outlined below.

Definition 2 *A certificate has the following form:*

$$\mathtt{Cert}\left(\mathtt{I}, \mathtt{D}_s, \mathtt{D}_e, \mathtt{S}, \mathtt{PK}_\mathtt{S}, \mathtt{SIG}_\mathtt{I}\langle \mathtt{I}, \mathtt{D}_s, \mathtt{D}_e, \mathtt{S}, \mathtt{PK}_\mathtt{S}\rangle\right)$$

or, simply, written as $\mathtt{Cert}\left(\mathtt{I}, \mathtt{D}_s, \mathtt{D}_e, \mathtt{S}, \mathtt{PK}_\mathtt{S}, \mathtt{SIG}_\mathtt{I}\right)$*, where* $\mathtt{I}$ *is the issuer,* $\mathtt{D}_s$ *and* $\mathtt{D}_e$ *are the start date and expiry date respectively,* $\mathtt{S}$ *is the subject of the certificate,* $\mathtt{PK}_\mathtt{S}$ *is the public key of* $\mathtt{S}$*, and* $\mathtt{SIG}_\mathtt{I}$ *is the signature of the issuer* $\mathtt{I}$*.*

There is a special certificate, named as `PAAcert`, that has the special form as follows:

$$\mathtt{Cert}\left(\_, \_, \_, \mathtt{PAA}, \mathtt{PK}_\mathtt{PAA}, \_\right)$$

`PAAcert` is intended to be the certificate of the PAA. The PAA is the top CA: its certificate is not issued by any other CA, so `PAAcert` is usually viewed as a self-signed certificate. In the case, the issuer's name, the start date, the expiry date and the signature parts are all omitted. All CAs and users would trust this certificate held by the PAA.

We denote the set of all certificates issued by CAs in the CMS as $\mathcal{C}$, called the *total certificate set* of the CMS. This means that any certificate we consider at any time must belong to $\mathcal{C}$. For any certificate $\mathtt{C} \in \mathcal{C}$, the following projection functions can be used to obtain the value of each component contained in $\mathtt{C}$. Let $\mathtt{C} = \mathtt{Cert}\left(\mathtt{I}, \mathrm{D}_s, \mathrm{D}_e, \mathtt{S}, \mathtt{PK}_\mathtt{S}, \mathtt{SIG}_\mathtt{I}\langle \mathtt{I}, \mathrm{D}_s, \mathrm{D}_e, \mathtt{S}, \mathtt{PK}_\mathtt{S}\rangle\right)$, we define:

$$\overline{\mathtt{I}}(\mathtt{C}) = \mathtt{I} \qquad \overline{\mathtt{D}_s}(\mathtt{C}) = \mathtt{D}_s$$
$$\overline{\mathtt{D}_e}(\mathtt{C}) = \mathtt{D}_e \qquad \overline{\mathtt{S}}(\mathtt{C}) = \mathtt{S}$$
$$\overline{\mathtt{PK}}(\mathtt{C}) = \mathtt{PK}_\mathtt{S} \qquad \overline{\mathtt{SIG}}(\mathtt{C}) = \mathtt{SIG}_\mathtt{I}\langle \mathtt{I}, \mathrm{D}_s, \mathrm{D}_e, \mathtt{S}, \mathtt{PK}_\mathtt{S}\rangle$$

These functions will be used frequently in our discussion.

2.3 CMS States and Transitions

At any moment in time, or at any given state, an agent in the CMS should hold zero or more certificates and, in particular, any CA should also be associated with a CRL (Certificate Revocation List) issued by itself periodically. Thus, we define states of a CMS as follows:

Definition 3 *Let $\langle \Omega, \downarrow \rangle$ be a certification topology and $\mathcal{C}$ the total certificate set. A relation s from Ω to $2^{\mathcal{C}} \times 2^{\mathcal{C}}$ is called a* CMS state *if it satisfies the state axioms SA1 – SA3 given below, where $2^{\mathcal{C}}$ is the power set of $\mathcal{C}$. Under a state s, we call $s(\mathtt{X}, \zeta, \eta)$ a triple, where $\zeta (\subset \mathcal{C})$ is a set of certificates issued to $\mathtt{X}$ and $\eta (\subset \mathcal{C})$ is a set of certificates issued by $\mathtt{X}$.*

State Axioms

(SA1) $\forall \mathtt{X} \bullet \exists \zeta \bullet \exists \eta \bullet (s(\mathtt{X}, \zeta, \eta) \wedge \forall \zeta' \bullet \forall \eta' \bullet (s(\mathtt{X}, \zeta', \eta') \rightarrow (\zeta' = \zeta) \wedge (\eta' = \eta)))$
(SA2) $\forall \mathtt{X} \bullet \forall \mathtt{C} \bullet (s(\mathtt{X}, \zeta, \eta) \wedge \mathtt{C} \in \zeta \rightarrow \overline{\mathtt{S}}(\mathtt{C}) = \mathtt{X})$
(SA3) $\forall \mathtt{X} \bullet \forall \mathtt{C} \bullet (s(\mathtt{X}, \zeta, \eta) \wedge \mathtt{C} \in \eta \rightarrow \overline{\mathtt{I}}(\mathtt{C}) = \mathtt{X})$

According to Definition 3, at any given state s (referred to as the current state), any agent $\mathtt{X}$ is associated with two certificate sets: ζ and η. All certificates in ζ should be possessed by $\mathtt{X}$ because their subject is $\mathtt{X}$; and all certificates in η must be issued by $\mathtt{X}$ itself at sometime because their issuer is $\mathtt{X}$. Thus, ζ may be viewed as the possessed certificate set of $\mathtt{X}$, while η may be viewed as the revoked certificate set of $\mathtt{X}$. Formally, we have

Definition 4 *Let s be a CMS state. If we have $s(\mathtt{X}, \zeta, \eta)$, then ζ is called the* possessed certificate set *of $\mathtt{X}$, which lists all certificates possessed by $\mathtt{X}$, and η is called the* revoked certificate set *of $\mathtt{X}$, which represents the CRL issued by $\mathtt{X}$ at the state s. In particular, if $\mathtt{C} \in \zeta$, we say that $\mathtt{X}$ holds the certificate $\mathtt{C}$; if $\mathtt{C} \in \eta$, we say that $\mathtt{X}$ has revoked the certificate $\mathtt{C}$.*

In the following, for the reason of simplifying our discussion, we will often use $\zeta_{\mathtt{X}}$ and $\eta_{\mathtt{X}}$ to denote the possessed certificate set and the revoked certificate set of an agent $\mathtt{X}$, respectively, at a given CMS state.

In the theory of state machines, transitions are usually described as actions that change states of a machine. Adopting the formal approach to defining a state machine proposed by Eastaughffe *et al.* [1], we define transitions of the CMS as follows:

Definition 5 *A CMS transition changes one CMS state into another and consists of three parts:*

- *the* Let *declaration, which introduces local variables for abbreviating expressions within the transition;*
- *the guard, denoted as* Pre*, a boolean expression which must be true for the current values of the variables; and*
- *the action list* Act*, essentially a parallel assignment statements, which involve the state changes.*

As an example, a simple transition is given as follows. Suppose $\mathtt{Y} \downarrow \mathtt{X}$, and, at the current state s, we have $s(\mathtt{X}, \zeta_{\mathtt{X}}, \eta_{\mathtt{X}})$ and now a transition involving only the issue by $\mathtt{Y}$ of a new certificate $\mathtt{C}$ to $\mathtt{X}$ is applied, then the transition might be defined as:

Let: $\mathtt{C} = \mathtt{Cert}\,(\mathtt{Y}, \mathtt{D}_s, \mathtt{D}_e, \mathtt{X}, \mathtt{PK}_\mathtt{X}, \mathtt{SIG}_\mathtt{Y})$
Pre: SucReqCert(X,Y)
Act: $\zeta_\mathtt{X} \leftarrow \zeta_\mathtt{X} \cup \{\mathtt{C}\}$

This definition means that, if the checking of the certificate request "X requests a certificate issued by Y" is successful, Y will issue a certificate to X. Therefore, when SucReqCert(X,Y) has the value "True", at the next state, denoted as s', the set of certificates held by X will be $\zeta_\mathtt{X} \cup \{\mathtt{C}\}$. Thus, we have $s'(\mathtt{X}, \zeta_\mathtt{X} \cup \{\mathtt{C}\}, \eta)$. This transition does not involve any other actions than issuing the certificate C to X by Y. Therefore, for any $\mathtt{Z} \in \Omega$, if $s(\mathtt{Z}, \zeta_\mathtt{Z}, \eta_\mathtt{Z})$ and $\mathtt{Z} \neq \mathtt{X}$, then $s'(\mathtt{Z}, \zeta_\mathtt{Z}, \eta_\mathtt{Z})$.

An expanded discussion on transitions will be given in Section 4.

2.4 A Formal Definition of a CMS

We are now able to give the formal definition of a CMS as follows.

Definition 6 *A* CMS (Certificate Management System) *is defined as a 5-tuple* $\langle \Omega, \downarrow, \mathcal{C}, \mathcal{S}, \mathcal{T} \rangle$ *where*

- Ω *is a set of agents, including the special agent* PAA*, and* Ω *with the relation* $\downarrow$ *forms a certification topology* $\langle \Omega, \downarrow \rangle$*;*
- $\mathcal{C}$ *is a non-empty set of certificates, called the* total certificate set*, that contains all possible certificates issued by CAs and a special element called the* PAAcert*, which is held by the* PAA*.*
- $\mathcal{S}$ *is a set of CMS states; and*
- $\mathcal{T}$ *is a set of transitions, a transition may change one state into another when it is applied.*

According to the definition, a CMS must satisfy all the axioms AA1 – AA6 and SA1 – SA3.

3 The CMS Policy

In this section, we present a general security policy for CMSs, which involves certificate issuing, access control, the validity of certificates, and the distribution of trust.

We assume that the operations that can be taken by agents are classified into two types: CMSservice and other. To formalize the general security policy, we introduce several predicates as follows:

CanCertify(X, Y):	X can certify Y.
ActType(a, Ty):	The type of an operation a is Ty.
CanTake(X, a):	X can perform the operation a.
Valid(X):	X is valid, where X can be a certificate, a public key, or a signature.
Holds(X,C):	X holds the certificate C.
HasRevoked(X,C):	X has revoked the certificate C.
Trusts(X,C):	X trusts the certificate C.

3.1 Certificate Issuing Policy

Let $\langle \Omega, \downarrow, \mathcal{C}, \mathcal{S}, \mathcal{T} \rangle$ be a CMS. Then, by the agent axioms, we can directly obtain the following conclusion: For any $\mathtt{X} \in \Omega$, its certification domain is defined. Formally, we define

$$\forall \mathtt{X} \bullet (\mathtt{IsCertDom}(\mathcal{D}, \mathtt{X}) \leftrightarrow \forall \mathtt{Y} \bullet (\mathtt{Y} \in \mathcal{D} \leftrightarrow \mathtt{X} \downarrow \mathtt{Y}))$$

where $\mathtt{IsCertDom}(\mathcal{D}, \mathtt{X})$ means that $\mathcal{D}$ is the certification domain of $\mathtt{X}$.

This definition does not require that $\mathtt{X}$ must be a CA; it can actually be any agent in the CMS. However, if $\mathtt{X}$ is a user, using the axiom AA3, it is easy to show that $\mathcal{D}$ is empty. That is, the certification domain of any user is the empty set.

Thus, the certificate issuing policy can simply be stated as follows:

– In a CMS, an agent can certify only those who belong to its certification domain.

This policy can be formalized as the following axiom:

(PA1) $\quad \forall \mathtt{X} \bullet \forall \mathtt{Y} \bullet (\mathtt{CanCertify}(\mathtt{X}, \mathtt{Y}) \rightarrow \mathtt{Y} \in \mathcal{D} \wedge \mathtt{IsCertDom}(\mathcal{D}, \mathtt{X}))$

An obvious corollary of the axiom PA1 is that users cannot certify anybody, because their certification domains are empty.

As an alternative of PA1, we may have

(PA1′) $\quad \forall \mathtt{X} \bullet \forall \mathtt{Y} \bullet (\mathtt{CanCertify}(\mathtt{X}, \mathtt{Y}) \rightarrow \mathtt{X} \downarrow \mathtt{Y})$

It is easy to show that PA1 and PA1′ are logically equivalent. However, we prefer PA1 due to a technical reason: the amount of work in checking if someone is an eligible member to whom an entity can issue a certificate based on the the certificate domain of this entity is usually less than that based on the relation $\downarrow$ if the certificate domain has already been calculated.

3.2 Access Control Policy

If a user requests a CMS service, he may be able to provide a valid certificate; but in some case he may not have one. Also attackers may try to use forged or stolen certificates to access the system. In the state-based model, to exclude unauthorised users, the CMS follows a basic rule – the access control policy stated as follows.

– No one without a valid certificate can be allowed to use security applications (CMS services).

This policy is formalized as the axiom:

(PA2) $\quad \forall \mathtt{X} \bullet \forall a \bullet (\mathtt{CanTake}(\mathtt{X}, a) \rightarrow$
$\mathtt{ActType}(a, \mathtt{other}) \vee \exists \mathtt{C} \bullet (\mathtt{Holds}(\mathtt{X}, \mathtt{C}) \wedge \mathtt{Valid}(\mathtt{C})))$

This says: for any agent $\mathtt{X}$ and any operation a, if $\mathtt{X}$ can perform the operation a, then the type of a is $\mathtt{other}$ or $\mathtt{X}$ holds a valid certificate.

3.3 The Validity of Certificates

According to the access control policy, any user who is applying for CMS services should be able to provide a valid certificate issued by a CA. In other words, the CMS must verify the validity of every certificate that a security application uses. Therefore, the CMS should provide support for the verification process to check if a certificate is valid. The CMS policy regarding the validity of certificates include the following items:

- The certificate PAAcert is held by the PAA and it is valid.
- A certificate held by someone other than the PAA is valid if and only if the following conditions are all satisfied: (1) the issuer of the certificate is a parent of this person, (2) the current time belongs to the time interval shown on the certificate, (3) the signature of the issuer is valid, and (4) the certificate has not been revoked by the issuer.
- The signature on a certificate is valid if and only if the issuer of the certificate holds a valid certificate.

In the state-based model, all these items have been formalized as the following axioms:

$$\begin{array}{ll}
\text{(PA3)} & \mathtt{Holds}(\mathtt{PAA}, \mathtt{PAAcert}) \wedge \mathtt{Valid}(\mathtt{PAAcert}) \\
\text{(PA4)} & \forall \mathtt{X} \bullet \forall \mathtt{C} \bullet (\mathtt{Holds}(\mathtt{X}, \mathtt{C}) \wedge \neg(\mathtt{X} = \mathtt{PAA}) \rightarrow (\mathtt{Valid}(\mathtt{C}) \leftrightarrow \overline{\mathtt{I}}(\mathtt{C}) \downarrow \mathtt{X} \wedge \\
 & (\mathtt{D}_s(\mathtt{C}) \leq \mathtt{Today} \leq \mathtt{D}_e(\mathtt{C})) \wedge \mathtt{Valid}(\overline{\mathtt{SIG}}(\mathtt{C})) \wedge \neg \mathtt{HasRevoked}(\overline{\mathtt{I}}(\mathtt{C}), \mathtt{C}))) \\
\text{(PA5)} & \forall \mathtt{C} \bullet (\mathtt{Valid}(\overline{\mathtt{SIG}}(\mathtt{C})) \leftrightarrow \exists \mathtt{C}' \bullet (\mathtt{Holds}(\overline{\mathtt{I}}(\mathtt{C}), \mathtt{C}') \wedge \mathtt{valid}(\mathtt{C}')))
\end{array}$$

where Today is variable representing the current time.

3.4 Trust Policy Axioms

From the above policy axioms, we can see that in general a single certificate is not enough to establish assurance that the certificate subject is authorised to access a CMS service it is requesting. Usually, a certificate path that starts from the certificate held by the trusted issuer is needed.

We make the following assumptions concerning trust within the CMS:

(1) CAs and users trust all CAs to faithfully execute their CA operations; and
(2) CAs and users trust that it is not viable to tamper with CMS certificates.

These assumptions can be well founded and supported by CMS practices. Firstly, assurance is provided for (1) through the use of accreditation of CAs, Certificate Practice Statements published by CAs and the implementation of appropriate policy[1]. Assurance is provided for (2) through the use of digital signatures, and good control of private keys.

[1] Note that policy for CAs can be listed and checked in much the same way as in which certificates are checked and can even be included as an extension in certificates.

The only assumption made concerning trusted certificates is that the certificate PAAcert, held by the PAA, is trusted by all CAs and users within the CMS. In fact, since the CMS is a top-down hierarchy, trust of all CMS certificates can be determined using the PAA certificate and by checking the validity of certificates along the certification path. For example, suppose that a CA's certificate is valid and its issuer is the PAA whose certificate is trusted then, from our assumptions, the CA's certificate is trusted. It is seen that for the CMS trust must always be referenced back to the single trust point, namely, the PAA's certificate (as is the case in top-down hierarchies).

Trust is transferred throughout the CMS using the trust policy stated as follows:

- Everybody trusts the certificate PAAcert.
- One trusts a certificate other than the PAAcert if and only if the certificate is valid and the certificate's issuer holds a certificate that the person trusts.

We formalize the trust policy with the following axioms:

(PA7) $\forall \mathtt{X} \bullet \mathtt{Trusts}(\mathtt{X}, \mathtt{PAAcert})$

(PA8) $\forall \mathtt{X} \bullet \forall \mathtt{C} \bullet (\neg(\mathtt{C} = \mathtt{PAAcert}) \rightarrow (\mathtt{Trusts}(\mathtt{X}, \mathtt{C}) \leftrightarrow \mathtt{Valid}(\mathtt{C}) \wedge \exists \mathtt{C}' \bullet (\mathtt{Holds}(\overline{\mathtt{I}}(\mathtt{C}), \mathtt{C}') \wedge \mathtt{Trusts}(\mathtt{X}, \mathtt{C}'))))$

In our model, the trust policy of the CMS consists of only the two axioms as given above. These axioms restrict the CMS to have only one trust point from which all trust is inferred. There are other ways to distribute trust within a CMS, such as multiple trust points, but these will not be considered here.

4 The Formalization of CMS Functions

In this section, we consider three CMS functions: certificate issuing, certificate revocation and certificate rekeying. All these functions of a CMS may lead to dynamic changes of state. We have already stated that the actions which change states of a CMS can be viewed as transitions within the system. Therefore, all these functions can formally be described through transitions.

4.1 Transitions

Let $\langle \Omega, \downarrow, \mathcal{C}, \mathcal{S}, \mathcal{T} \rangle$ be a CMS. We now discuss transitions at a given state s, where $s \in \mathcal{S}$.

A simple transition is defined as one of the following actions:

- Issuing of a certificate C to an agent X by a CA Y;
- Revoking of a certificate C by an agent X.
- Rekeying of a certificate C held by an agent X.

We denote the simple transitions involving the above actions as `Issues(Y,C,X)`, `Revokes(X,C)`, and `Rekeys(X,C)` respectively.

Without loss of generalisation, a transition of a CMS may be a simple transition as above, or a compound transition formed from several simple transitions that occur in parallel at the same time. That is, if t_1, ..., t_n are simple transitions, $t_1 \| \ldots \| t_n$ is a compound transition, where $\|$ is called the parallel operator over transitions.

4.2 Certificate Issuing

CMSs support two types of certification requests: *self-registration* and *agent-registration*. In a self-registration request, an Organization Registration Authority (ORA) may provide a secret message to the prospective certificate holder. The entity itself generates its own key pair, forms a certificate request, signs it with the corresponding private key material, and includes authentication information based on the secret message provided by the ORA. The CA receives the request, and verifies the requester's identity through the authentication information. If accepted, the CA will generate a new certificate and send it to the certificate holder.

In an agent-registration request, the agent vouches for the prospective certificate holder's identity and the binding to the public key. When a certificate request comes from an accredited agent, the receiving CA will process the request and, if accepted, generate a new certificate and send it to the agent. The CA may also send the new certificate to the holder. The CA may reject the agent-generated certification request due to the following reasons: the authentication of the identity of the certificate's subject fails, or the agent does not possesses a valid certificate. If the CA rejects the request, it will report the failure to the agent stating the reason of the failure.

In any case, the action "issuing a certificate to `X` by `Y`" can happen only when the checking of certificate request is successful. We use the predicate `SucReqCert(X,Y)` to denote that the checking of certificate request "`X` requests a certificate issued by `Y`" is successful. Then the transition `Issues(Y,C,X)` has the form given in Section 2.3.

The *certificates issuing* function must follow the certificate issuing policy. Therefore, the action "issuing a certificate to `X` by `Y`" can be performed only when the following conditions are satisfied:

- `Y` can certify `X`, i.e., $\mathtt{Y} \downarrow \mathtt{X}$; and
- `Y` holds a valid certificate.

Thus, the value of the guard `SucReqCert(X,Y)` can be obtained by the formula:

$$\mathtt{SucReqCert}(\mathtt{X},\mathtt{Y}) \leftrightarrow (\mathtt{Y} \downarrow \mathtt{X}) \wedge \exists \mathtt{C}' \bullet (\mathtt{Holds}(\mathtt{Y},\mathtt{C}') \wedge \mathtt{Valid}(\mathtt{C}'))$$

By axiom PA1, if the prospective certificate holder in a certification request does not belong to the certification domain of the issuer required, the request must not be accepted. Usually, when a CA receives a certification request, it will

first check if that the prospective certificate holder appears in its certification domain. If not, it may immediately reject the request.

In the case that a number of issuing certificate actions happen at the same time, a compound transition may be needed. The definition of a compound transition can simply be formed by conjunction of the formulas appearing in the corresponding parts of all simple transitions constructing the compound transition. For instance, suppose "issuing a certificate $\mathtt{C}_1$ to $\mathtt{X}_1$ by $\mathtt{Y}_1$", ..., "issuing a certificate $\mathtt{C}_n$ to $\mathtt{X}_n$ by $\mathtt{Y}_n$" happen at the same time, then the certificate issuing behavior can be described as the compound transition

$$\mathtt{Issues}(\mathtt{Y}_1, \mathtt{C}_1, \mathtt{X}_1) \| \ldots \| \mathtt{Issues}(\mathtt{Y}_n, \mathtt{C}_n, \mathtt{X}_n),$$

whose definition as follows can directly be obtained from the simple transitions $\mathtt{Issues}(\mathtt{Y}_1, \mathtt{C}_1, \mathtt{X}_1)$, ..., and $\mathtt{Issues}(\mathtt{Y}_n, \mathtt{C}_n, \mathtt{X}_n)$:

$$\begin{array}{ll}
\mathsf{Let:} & \mathtt{C}_1 = \mathtt{Cert}\left(\mathtt{Y}_1, \mathtt{D}_{s_1}, \mathtt{D}_{e_1}, \mathtt{X}_1, \mathtt{PK}_{\mathtt{X}_1}\mathtt{SIG}_{\mathtt{Y}_1}\right), \\
 & \qquad\qquad \ldots\ldots \\
 & \mathtt{C}_n = \mathtt{Cert}\left(\mathtt{Y}_n, \mathtt{D}_{s_n}, \mathtt{D}_{e_n}, \mathtt{X}_n, \mathtt{PK}_{\mathtt{X}_n}, \mathtt{SIG}_{\mathtt{Y}_n}\right) \\
\mathsf{Pre:} & \mathtt{SucReqCert}(\mathtt{X}_1, \mathtt{Y}_1) \\
 & \qquad \ldots\ldots \\
 & \mathtt{SucReqCert}(\mathtt{X}_n, \mathtt{Y}_n) \\
\mathsf{Act:} & \zeta_{\mathtt{X}_1} \leftarrow \zeta_{\mathtt{X}_1} \cup \{\mathtt{C}_1\} \\
 & \qquad \ldots\ldots \\
 & \zeta_{\mathtt{X}_n} \leftarrow \zeta_{\mathtt{X}_n} \cup \{\mathtt{C}_n\}
\end{array}$$

However, we should note that, if there is more than one CA issuing a certificate to the same agent, the actions involved in those issues could be combined to a single action. For instance, suppose a compound transition contains two simple transitions:

$$\mathtt{Issues}(\mathtt{Y}_1, \mathtt{C}_1, \mathtt{X}) \text{ and } \mathtt{Issues}(\mathtt{Y}_2, \mathtt{C}_2, \mathtt{X}),$$

where both

$$\mathtt{C}_1 = \mathtt{Cert}\left(\mathtt{Y}_1, \mathtt{D}_{s_1}, \mathtt{D}_{e_1}, \mathtt{X}, \mathtt{PK}_{\mathtt{X}}, \mathtt{SIG}_{\mathtt{Y}_1}\right) \text{ and}$$
$$\mathtt{C}_2 = \mathtt{Cert}\left(\mathtt{Y}_2, \mathtt{D}_{s_1}, \mathtt{D}_{e_1}, \mathtt{X}, \mathtt{PK}'_{\mathtt{X}}, \mathtt{SIG}_{\mathtt{Y}_2}\right)$$

are issued to $\mathtt{X}$. We have the action $\zeta_{\mathtt{X}} \leftarrow \zeta_{\mathtt{X}} \cup \{\mathtt{C}_1\}$ in the simple transition $\mathtt{Issues}(\mathtt{Y}_1, \mathtt{C}_1, \mathtt{X})$, and the action $\zeta_{\mathtt{X}} \leftarrow \zeta_{\mathtt{X}} \cup \{\mathtt{C}_2\}$ in the simple transition $\mathtt{Issues}(\mathtt{Y}_2, \mathtt{C}_2, \mathtt{X})$. In this case, the two actions should be combined to a single action $\zeta_{\mathtt{X}} \leftarrow \zeta_{\mathtt{X}} \cup \{\mathtt{C}_1, \mathtt{C}_2\}$.

4.3 Certificate Revocation

At the initial time when the CMS starts, we may assume that the sets of revoked certificates for all CAs (as well as all users) are empty. At any time after the initialization, every CA in the CMS must periodically issue a CRL, which contains all the certificates that it has revoked.

Certificates are revoked by a CA who issued them when the CA has reason to believe that they can no longer be trusted. At any time, when a CA, say X, issues a certificate C, X should of course trust C. That is, the formula Trusts(X,C) automatically obtains the truth value "True" at the same time when C is issued, and this value will keep until C will not be trusted by X due to some reasons. Therefore, if a CA X revokes a certificate C, then preconditions should include: the issuer of C is X, and X does not trust C. We define

$$\mathtt{ApvRevCert}(\mathtt{X}, \mathtt{C}) \leftrightarrow (\overline{\mathtt{I}}(\mathtt{C}) = \mathtt{X}) \wedge \neg \mathtt{Trusts}(\mathtt{X}, \mathtt{C})$$

where ApvRevCert(X,C) means that a need "X revokes the certificate C" is approved. Thus, "X revokes C" is the simple transition Revokes(X,C), which can be defined as follows:

Let:
Pre: ApvRevCert(X,C)
Act: $\eta_{\mathtt{X}} \leftarrow \eta_{\mathtt{X}} \cup \{\mathtt{C}\}$

A CA periodically issues a CRL, which may revoke a number of certificates. For instance, suppose that, at the current time, the certificates that X is revoking include $\mathtt{C}_1, \ldots, \mathtt{C}_n$, then the revocation action can be represented as the compound transition $\mathtt{Revokes}(\mathtt{X}, \mathtt{C}_1) \| \ldots \| \mathtt{Revokes}(\mathtt{X}, \mathtt{C}_n)$. In the same way we used for the representation of certificate issuing functions, we can easily obtain the definition of the compound transition as follows:

Let:
Pre: $\mathtt{ApvRevCert}(\mathtt{X}, \mathtt{C}_1)$
......
$\mathtt{ApvRevCert}(\mathtt{X}, \mathtt{C}_n)$
Act: $\eta_{\mathtt{X}} \leftarrow \eta_{\mathtt{X}} \cup \{\mathtt{C}_1, \ldots, \mathtt{C}_n\}$

The definition of a compound transition involving the case where several CAs revoke a number of certificates can also be obtained from the definitions of simple transitions contained in the compound transition.

4.4 Certificate Rekeying

At any time after the CMS is initialized, it is possible that some CA may need to change the key pair (the public key and the corresponding private key) involved in a certificate held by itself due to an increased risk of the current keys being compromised.

The process of changing the keys of a CA for a certificate, performed by the certificate rekeying function, is as follows. When a new key pair is generated for a certificate by the CA who wants to change its key pair, a new certificate is created and a certificate signature request is sent to the parent who signed the old certificate. If the certificate rekeying request is accepted, the parent signs the new certificate and returns it to the CA. Rekeying a certificate must affect the

certificate hierarchy, since some certificates held by the CA's children may have been signed with the old private key of the CA. Therefore, upon receiving the new certificate, the CA needs to re-sign all those certificates of its subordinates with the new private key, which were signed with that old private key.

The certificate rekeying function can also formally be described using transitions. To do this, we first introduce two new predicates as follows:

$\texttt{AcpReKey}(\texttt{X}, \texttt{C})$:	The request "$\texttt{X}$ wants to rekey the certificate $\texttt{C}$" is accepted.
$\texttt{SIGKEY}(\texttt{C}_1, \texttt{C}_2)$:	The certificate $\texttt{C}_1$ is signed with the private key for which the corresponding public key is issued in the certificate $\texttt{C}_2$.

We now assume that $\texttt{X}$ changes the key pair for a certificate $\texttt{C}$ held by itself, and $\texttt{C}_1, \ldots, \texttt{C}_n$ are all the certificates signed with the private key for which the corresponding public key is issued in the certificate $\texttt{C}$. Then this action can be expressed as the simple transition $\texttt{Rekeys(X,C)}$, which has the form as follows:

$$
\begin{array}{ll}
\textsf{Let:} & \texttt{C} = \texttt{Cert}\,(\texttt{Y}, \texttt{D}_s, \texttt{D}_e, \texttt{X}, \texttt{PK}_\texttt{X}, \texttt{SIG}_\texttt{Y}) \\
& \texttt{C}' = \texttt{Cert}\,(\texttt{Y}, \texttt{Today}, \texttt{D}'_e, \texttt{X}, \texttt{PK}_\texttt{X}, \texttt{SIG}'_\texttt{Y}) \\
& \texttt{C}_1 = \texttt{Cert}\,(\texttt{X}, \texttt{D}_{s_1}, \texttt{D}_{e_1}, \texttt{X}_1, \texttt{PK}_{\texttt{X}_1}, \texttt{SIG}_\texttt{X}) \\
& \texttt{C}'_1 = \texttt{Cert}\,(\texttt{X}, \texttt{Today}, \texttt{D}'_{e_1}, \texttt{X}_1, \texttt{PK}_{\texttt{X}_1}, \texttt{SIG}'_\texttt{X}) \\
& \qquad\qquad \cdots\cdots \\
& \texttt{C}_n = \texttt{Cert}\,(\texttt{X}, \texttt{D}_{s_n}, \texttt{D}_{e_n}, \texttt{X}_n, \texttt{PK}_{\texttt{X}_n}, \texttt{SIG}_\texttt{X}) \\
& \texttt{C}'_n = \texttt{Cert}\,(\texttt{X}, \texttt{Today}, \texttt{D}'_{e_n}, \texttt{X}_n, \texttt{PK}_{\texttt{X}_n}, \texttt{SIG}'_\texttt{X}) \\
\textsf{Pre:} & \texttt{AcpReKey(X,C)} \\
& \texttt{SIGKEY}(\texttt{C}_1, \texttt{C}) \\
& \quad \cdots\cdots \\
& \texttt{SIGKEY}(\texttt{C}_n, \texttt{C}) \\
\textsf{Act:} & \zeta_\texttt{X} \leftarrow (\zeta_\texttt{X} \setminus \{\texttt{C}\}) \cup \{\texttt{C}'\} \\
& \zeta_{\texttt{X}_1} \leftarrow (\zeta_{\texttt{X}_1} \setminus \{\texttt{C}_1\}) \cup \{\texttt{C}'_1\}, \\
& \quad \cdots\cdots \\
& \zeta_{\texttt{X}_n} \leftarrow (\zeta_{\texttt{X}_n} \setminus \{\texttt{C}_n\}) \cup \{\texttt{C}_n\}
\end{array}
$$

where $\texttt{Today}$ is a variable representing the current time, and $\mathcal{A} \setminus \mathcal{B}$ stands for the difference set of the sets $\mathcal{A}$ and $\mathcal{B}$.

Note that, when the certificate $\texttt{C}$ is rekeyed to become the new certificate $\texttt{C}'$, the old certificate $\texttt{C}$ is automatically revoked by its issuer at the same time. For simplifying our discussion, we omit such an action.

Similarly, we can consider those cases where a number of CAs change the key pairs for a number of certificates at the same time. We may also consider a compound transition consisting of different types of simple transitions. There should not be any difficulty obtaining the definition of a compound transition from the definitions of simple transitions.

5 Certificate Verification

In a CMS, the certificate verification function is a major CMS client function, which is responsible for verifying the validity of every certificate that a security

application uses. Certificate verification proves that no one has tampered with the contents of the certificate, i.e. the public key, the validity date, and the identity of the certificate issuer and owner have not been changed. This relies on the properties of the digital signature[2]. This section discusses the certificate verification principle, and presents verification rules.

5.1 The Certificate Verification Principle

Verifying a given certificate involves verifying the identity of the certificate issuer and owner, verifying the validity dates of the certificate, verifying the signature on the certificate, and verifying the certificate against the latest issuer's CRL list to make sure it has not been revoked. In order to verify the signature on the certificate, the certificate that the issuer holds should also be verified in the same way using the next higher certificate in the certificate path. Therefore, the verification process is iterative and terminates when a trusted certificate (by the verifier) is reached, or a invalid certificate is encountered.

When an invalid certificate is encountered, the certificate is not accepted as valid by the verifier. The verification process may begin by verifying the signature on the next certificate on the certification path with the PAA's public key from `PAAcert`. This process continues down the certification path until the required certificate is verified or an invalid certificate is encountered.

The certificate verification principle, which must be followed by all CMS users, can be stated as below:

- In a certificate verification process, when the verifier has found a certificate path constructed for verifying a required certificate in which all certificates are valid he may accept this certificate as valid, and in all other cases the certificate is regarded as invalid.

Note that, according to the certificate verification principle, it can happen that a certificate may actually be valid but the verifier did not find a corresponding certificate path in which all certificates are valid. In such a case, the verifier cannot accept this certificate as valid. This is the correct choice on security grounds.

5.2 Verification Rules

According to the above discussion, the process of obtaining and verifying the certificates from the trusted certificate to a required certificate is referred to as *certificate path development* and *certificate path verification*, respectively. An

[2] The digital signature provides a way to detect if any of these values have been changed. The original certificate issuer calculates a digital signature over the values that require protection and then issues the values and the signature which form the certificate. This signature can be checked by anyone who has a valid copy of the issuer's public key (obtained from the issuer's certificate), at a later stage. The details of the signature algorithm are beyond the scope of this paper.

efficient way to develop a certificate path is to start with the required certificate and build a certificate chain back towards a certificate trusted by the verifier. Certificate path verification can be done using certificate verification rules shown below. These rules are directly derived from the policy axioms.

A rule is usually expressed in the following form:

$$\frac{A_1 \quad A_2 \quad \ldots\ldots \quad A_n}{A}$$

which means that if formulas $A_1, A_2, \ldots, A_n$ are all proved to be true, then the fact "the formula A is true" is proved.

Verification Rules

(R1) $$\frac{\overline{\mathtt{I}}(\mathtt{C}) \downarrow \overline{\mathtt{S}}(\mathtt{C}) \quad \overline{\mathtt{D}_s}(\mathtt{C}) \leq \mathtt{Today} \leq \overline{\mathtt{D}_e}(\mathtt{C}) \quad \neg(\mathtt{C} \in \eta_{\overline{\mathtt{I}}(\mathtt{C})}) \quad \mathtt{valid}(\overline{\mathtt{SIG}}(\mathtt{C}))}{\mathtt{valid}(\mathtt{C})}$$

(R2) $$\frac{\mathtt{C}' \in \zeta_{\overline{\mathtt{I}}(\mathtt{C})} \quad \mathtt{valid}(\mathtt{C}')}{\mathtt{valid}(\overline{\mathtt{SIG}}(\mathtt{C}))}$$

(R3) $$\frac{\neg(\overline{\mathtt{I}}(\mathtt{C}) \downarrow \overline{\mathtt{S}}(\mathtt{C}))}{\neg\mathtt{Valid}(\mathtt{C})}$$

(R4) $$\frac{(\mathtt{Today} < \overline{\mathtt{D}_s}(\mathtt{C})) \vee (\mathtt{Today} > \overline{\mathtt{D}_e}(\mathtt{C}))}{\neg\mathtt{Valid}(\mathtt{C})}$$

(R5) $$\frac{\mathtt{C} \in \eta_{\overline{\mathtt{I}}(\mathtt{C})}}{\neg\mathtt{Valid}(\mathtt{C})}$$

(R6) $$\frac{\neg\mathtt{Valid}(\overline{\mathtt{SIG}}(\mathtt{C}))}{\neg\mathtt{Valid}(\mathtt{C})}$$

(R7) $$\frac{\neg\exists\mathtt{C}' \bullet (\mathtt{C}' \in \eta_{\overline{\mathtt{I}}(\mathtt{C})} \wedge \mathtt{valid}(\mathtt{C}'))}{\neg\mathtt{valid}(\overline{\mathtt{SIG}}(\mathtt{C}))}$$

(RX) $$\frac{\mathtt{Trusts}(\mathtt{X}, \mathtt{C})}{\mathtt{Valid}(\mathtt{C})}$$

Rules (R1) and (R2) together form the basis for the verification of certificates; while rules (R3) – (R7) can be applied to promptly exclude unauthorised users by finding some flaw in their certificates. The rule (RX) depends on a particular verifier ($\mathtt{X}$) and is related to trust.

Suppose a certificate path, say $\langle \mathtt{C}_0, \mathtt{C}_1, \ldots, \mathtt{C}_{n-1}, \mathtt{C}_n \rangle$ has been developed for verifying the certificate $\mathtt{C}_0$, where $\mathtt{C}_n$ is a certificate trusted by the verifier. Then certificate path verification, i.e., proving that all the certificates in the path are valid, is required for verifying $\mathtt{C}_0$. There are two ways to perform the verification process: one is *top-down* verification, and the other is *bottom-up* verification. In bottom-up verification, the verifier starts to consider the validity of $\mathtt{C}_0$. He first checks if $\overline{\mathtt{I}}(\mathtt{C}_0) \downarrow \overline{\mathtt{S}}(\mathtt{C}_0)$ holds, checks if $\overline{\mathtt{D}_s}(\mathtt{C}) \leq \mathtt{Today} \leq \overline{\mathtt{D}_e}(\mathtt{C})$ holds, and checks

if $\mathtt{C}_0$ does not belong to $\eta_{\overline{\mathtt{I}}(\mathtt{C}_0)}$. If any of the three checks fails, the verification process terminates with the result that $\mathtt{C}_0$ cannot be accepted as valid from this path. If all of these checks are successful, the verifier then checks validity of the signature on $\mathtt{C}_0$. He then needs to consider the validity of $\mathtt{C}_1$. In the same way, he may terminate with the result that $\mathtt{C}_0$ cannot be accepted as valid from this path because $\mathtt{C}_1$ is not valid, or proceed to to consider the validity of $\mathtt{C}_2$. Continuing the process, at the last, the verifier may only need to consider the validity of $\mathtt{C}_n$. Because $\mathtt{C}_n$ is a certificate trusted by the verifier, $\mathtt{valid}(\mathtt{C}_n)$ holds. Thus, the verifier has proved that all certificates on the path are valid, and may therefore accept the certificate $\mathtt{C}_0$ as valid.

With top-down verification, the verifier starts to consider the validity of $\mathtt{C}_{n-1}$ by accepting $\mathtt{C}_n$ as a valid certificate, because he trusts $\mathtt{C}_n$.

6 Conclusion

We have presented an approach to the formal specification of the structure, functions and the security policy of a CMS. A state-based model for CMSs has been proposed. With this model, all CMS functions, such as certificate issuing, certificate revocation and certificate rekeying, have formally been specified as transitions that change the states of a CMS. For the major CMS client function – certificate verification, we have in particular discussed the verification principle, certificate path development, and given the verification rules which can be used for certificate path verification.

The correctness and effectiveness of security mechanisms for a system (particularly a computer network) usually depend on the understanding of the developer and the evaluator to the security requirements of the system and their analysis. Many mechanisms, presented in somewhat descriptive form, are not suitable for formal analysis and practical implementation. Our model provides an approach to the formal specification of a CMS, the security policy and CMS functions are all formally specified. Based on this model, the security mechanisms, which are used to enforce the policy, can also be presented in a formal form. Such a formal presentation can help the developer and the evaluator as well as the user to understand the security-relevant behavior of a system, and guide them in the design and analysis of the security architecture, and in the implementation of the system. The formal approach proposed in this paper and the state-based model itself may be used as the basis for other activities in the secure computer network area, such as research into new mechanisms, services and secure network applications areas, and integration of individual mechanisms into large secure and reliable network architectures.

As a future work, we have planed to mechanise our theory in a general theorem prover, Isabelle [9]. Once a reasoning system for CMSs has been developed, certificate verification and the proof of security properties of a CMS can be automatically performed.

Future work also includes investigating the certificate verification algorithms based on the state-based model. It may also be interesting to look at the different

distributions of trust points within a CMS in more detail. Joining CMS's, with so called cross-certificates, is another important issue. Comparisons of solutions and suggestions as to how distribution of trust points could be implemented in these extended CMS structures also needs to be considered.

Acknowledgements

We would like to thank Dr. Brendan Mahony and Dr. Jim McCarthy for helpful discussions on the theory and techniques aspects. Thanks are also due to anonymous referees for their valuable comments and suggestions.

References

1. K. A. Eastaughffe, M. A. Ozols, and A. Cant. Proof tactics for a theory of state machines in a graphical environment. In *Proceedings of the 14th International Conference on Automated Deduction (CADE-14)*, Lecture Notes in Artificial Intellegince, pages 366–379. Springer-Verlag, 1997. 80
2. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transaction on Information Theory*, 31:469–472, 1985. 75
3. W. Ford. Advances in public-key certificate standards. *ACM SIGSAC Security Audit & Control Review*, 13(3), 1995. 75
4. W. Ford and M. Baum. *Secure Electronic Commerce: Building the Infrastructure for Digital Signatures and Encryption*. Prentice-Hall, 1997. 75
5. R. Housley, W. Ford, W. Polk, and D. Solo. Internet X.509 public key infrastructure certificate and CRL profile. IETF X.509 PKI (PKIX) Working Group (Draft), January 1999. 76, 77
6. N. Kapidzic. Extended certificate management system: Design and protocols. Technical report, DVS, 1997. 76
7. N. Kapidzic. Creating security applications based on the global certificate management system. *Computers & Security*, 17:507–515, 1998. 76
8. S. Kent. Privacy Enhancement for Internet Electronic Mail, Part II: Certificate-Based Key Management, Request for Comments 1422. Network Working Group, 1993. 77
9. L. C. Paulson. *ML for Working Programmer*. Cambridge University Press, 1991. 91
10. L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1-2):85–128, 1998. 76
11. Denis Trcek. Organization of certification authorities in a global network. *Computer Security Journal*, 10(1):72–81, 1994. 76
12. T. Y. C. Woo and S. S. Lam. Authorization in distributed systems: A new approach. *Journal of Computer Security*, pages 107–136, 2(1993). 76