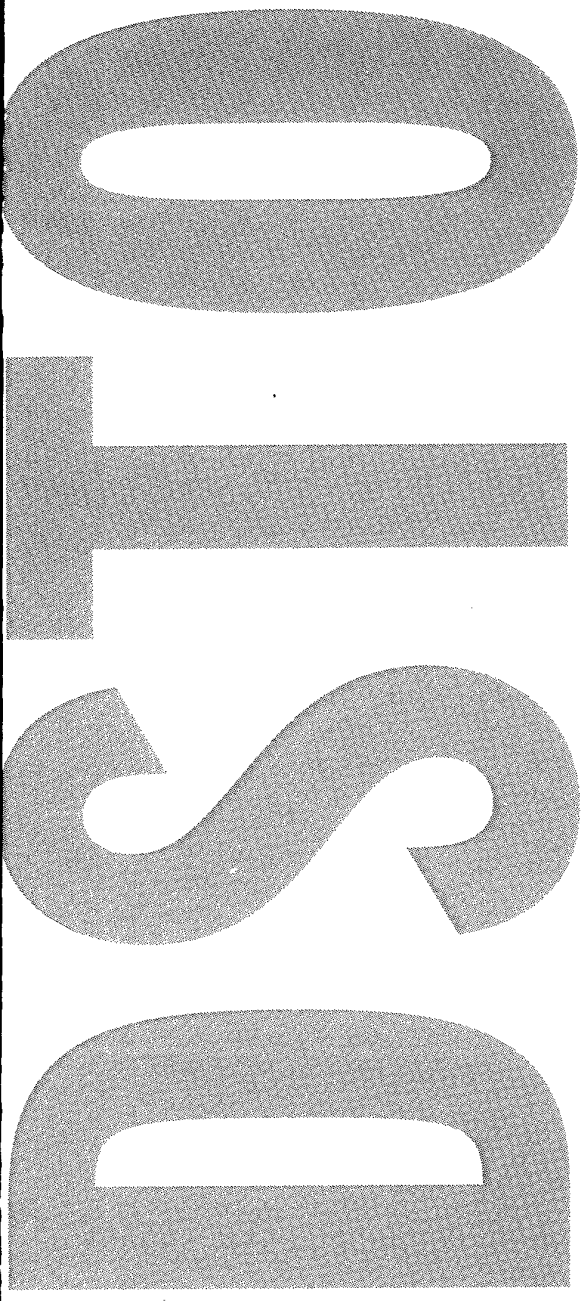
DEPARTMENT OF DEFENCE
DEFENCE SCIENCE & TECHNOLOGY ORGANISATION
DSTO



Formalization of Public Key Infrastructures

Maris Ozols, Tony Cant,
Chuchang Liu and Marie Henderson

DSTO-RR-0202

Formalization of Public Key Infrastructures

Maris Ozols, Tony Cant, Chuchang Liu and Marie Henderson

Information Technology Division
Electronics and Surveillance Research Laboratory

DSTO–RR–0202

ABSTRACT

Public key technology within a Public Key Infrastructure (PKI) has been widely promoted to support secure digital communications. However, imprecise specifications for PKIs, which are usually written in a natural language, have led to varying implementation and interpretations of conformance. There have also been cases where defects have been identified years later, some of which were serious and could cause incorrect acceptance of certification paths. In this paper we provide a formal solution to the PKI specification dilemma by introducing a state-based model for the description of the architecture of a PKI and related functions. We propose a formal approach to the representation of, and reasoning about, the behavior and security properties of PKIs, and also give a framework for mechanizing our theory in the Isabelle theorem prover. With our method, the essential aspects of PKIs can be clearly formulated, facilitating the testing and analysis of their implementations in a more rigorous and well defined way.

Published by

DSTO Electronics and Surveillance Research Laboratory
PO Box 1500
Salisbury, South Australia 5108 Australia

Telephone: (08) 8259 5555

Fax: (08) 8259 6567

AR No. AR–011–780

April, 2001

APPROVED FOR PUBLIC RELEASE

Formalization of Public Key Infrastructures

EXECUTIVE SUMMARY

There has been a global movement towards using public key technology managed by a Public Key Infrastructure (PKI) to support secure digital communications, such as electronic commerce and secure military messaging. However, imprecise specifications for PKIs, which are usually written in a natural language, have led to varying implementations and interpretations of conformance. There have also been cases where defects have been identified years later, some of which were serious and could cause incorrect acceptance of certification paths.

Formal methods can not only give a specification for a system in a precise and unambiguous way, but also provide mathematical proof technologies to verify critical properties of the system for guiding the developer towards a design of the security architecture of the system and its implementation. This report provides a formal solution to the PKI specification dilemma. The authors introduce a state-based model for the description of the architecture of a PKI and related functions, and propose a formal approach to the description of, and reasoning about, the behavior and security properties of PKIs.

Certificate verification is a central PKI client service. In a PKI, the integrity of each certificate that the security application relies upon must be verified. Verification proves that no one has tampered with the contents of the certificate, so that the verifier may accept the certificate as valid and would trust the public key bound to the subject of the certificate. The certificate verification service performs the verification process and may store verified certificates in the local database for later use. With the certificate verification function, this paper in particular discusses its two components – the certificate *path development* and *path validation*. Path development is used to find a certification path and provide it to path validation; while path validation takes the responsibility to verify the path as valid or identify where and why the path validity fails. This report proposes an algorithm for path development which is based on a given PKI state and the trusted certificate set of the verifier, and presents a framework for path validation based on a method that separates the checks of certificates into three categories – single checks, pair checks and path checks.

The authors also propose a framework for mechanizing their theory in Isabelle, a generic theorem prover, and in particular present the essential Isabelle theories that are used for the certification path validation.

In the method proposed by the authors of this paper, the essential aspects of PKIs can be clearly formulated, which would facilitate the testing of many implementations in a more rigorous and well defined way.

Authors

Maris Ozols
Information Technology Division Division

Maris Ozols is a Senior Research Scientist and he leads the Security Architectures section within the Trusted Computer Systems group. His research interests include information security, information architectures, formal modelling approaches, and interactive and automated reasoning systems. Previous work has covered safety critical systems and interfaces to theorem provers, whilst current activity is focused on security architectures, particularly for distributed systems. Prior to joining DSTO in 1990, Maris completed a BSc (Hons) at the University of Adelaide, followed by mathematical research in algebra at Monash University, obtaining his PhD in 1991.

Tony Cant
Information Technology Division Division

Tony Cant is a Senior Research Scientist within the Trusted Computer Systems Group. He leads research into high assurance methods and tools for safety and security critical systems, focusing on approaches to machine-assisted reasoning about their critical properties. He is also involved in policy formulation and the development of an Australian Defence Standard for the procurement of safety critical systems. Tony studied at the University of Adelaide, receiving a BSc(Hons) in 1974, and a PhD in Mathematical Physics in 1978. He held a number of academic research positions in mathematics and theoretical physics, as well as working in science policy, and joined DSTO in 1990.

Chuchang Liu
Information Technology Division Division

Dr. Chuchang Liu joined DSTO in 1998 and is a Research Scientist with the Trusted Computer Systems group. His research interests include computer network security, formal methods for software specification and verification, and temporal logics and their applications. Dr. Liu obtained a PhD in computing science from Macquarie University in 1998. Dr. Liu's BSc(Hons) and MEcomp degrees were obtained from Wuhan University, China, where he worked as a Lecturer/Associate Professor for 9 years.

Marie Henderson
Information Technology Division Division

Dr. Marie Henderson is a Research Scientist with the Information Security Reseach Centre at Queensland University of Technology. Previously she worked for the Trusted Computing Systems group at the Defence Science and Technology Organisation. She obtained her PhD from the School of Information Technology at the University of Queensland in 1996.

Contents

1 Introduction

1.1 Motivation

There has been a global movement towards using public key technology managed by a Public Key Infrastructure (PKI) to support secure digital communications (for example: electronic commerce and secure messaging). In order to use public key cryptography [13, 51] securely, it is essential to make the public key available to those one wishes to communicate with in a trusted way. The main threat is "theft of identity", which results in masquerade as another party, or access to confidential material of another party.

Certificates are created by CAs (Certificate Authorities) and are used as the vehicle by which public keys may be stored, distributed or forwarded over unsecured media without danger of undetectable manipulation. The CA attests to the correctness of the information in the certificate (in particular, the name of the party who controls the corresponding private key) by signing the certificate (this also protects the information from being altered thereafter). The certificate is verified by checking the signature, in which case a valid copy of the public key of the CA is required. In a large PKI, multiple CAs may be required with chains of certificates and even possibly multiple trust points, i.e., multiple trusted public keys. Multiple trust points especially occur in the case where users come from different PKIs.

The Internet Engineering Task Force (IETF) is involved in guiding the evolution of the Internet and is the main organisation involved in developing Internet Standard specifications. Agreement on content and meaning is arrived at through a process involving both informal and formal discussions along with Working Groups (WGs) and a series of draft publications from which the final Request For Comment (RFC) is distilled. The process is sometimes described as forming a "rough consensus" on the appropriate content and meaning. The RFCs that form part of an effort towards standardisation need to be clear as well as being inclusive and accessible requirements that are often at odds. These types of concerns also arise with most other standardisation efforts.

In particular, the PKI profile drafted in RFC 2459 is concerned with standardising a Public Key Infrastructure (PKI) for the Internet [22] and is a very influential PKI standard. The PKI described in that document is known as PKIX because it uses the ITU-T X.509 version 3 (X.509) certificate format [57]. The path validation protocol proposed in Section 6 of [22] consists of an algorithm used as a specification in the following way: "implementations are required to derive the same results but are not required to use the specified procedures". Such a specification could lead to varying definitions or interpretations of conformance. In fact, there are already a number of problems, such as those we mention below, arising from this approach.

The English language is wonderfully expressive: a single word or phrase is able to convey many different meanings. This is why it is unsuitable on its own for providing a precise specification. When only a natural language specification is given, it is probable that there will be different interpretations which all meet the specification, although they may logically be different. General examples of this type of problems are widespread. Such problems should be avoided for PKI documents that are expected to become global standards involving the transfer of trust.

In mid 1999 a defect report [10] was filed concerning the X.509 standard. A number of defects were identified, some of which were serious and could cause incorrect acceptance of certification paths. Due to the wide acceptance of X.509 this affects a number of other standardisation efforts, like PKIX. The PKIX WG is working on a new version of RFC 2459 [23], hoping to anticipate the changes to X.509. The PKIX document is therefore important as it indicates the likely direction for all standards relying on X.509. In order to avoid such problems, the IETF has adopted the following strategy: a specification must have at least two independent and interoperable implementations from different code bases, for which sufficient successful operational experience has been obtained in order to be elevated to the "Draft Standard" level [3]. These procedures are explicitly aimed at recognizing and adopting generally-accepted practices. However, there is still a lack of sold theoretical foundations in the development of PKI specifications.

It has also been recognised that interoperability problems have occurred with S/MIME products [38]. It is argued that the looseness of the S/MIME specification has led to the development of a number of compliant but nonetheless non-interoperable products. It is likely that such problems will occur with PKIX as well, the extent of which will be realised as products being to appear. Although example conforming certificates are given with PKIX there have been no example conforming certification paths found yet. It could be argued that example certification paths would provide such narrow coverage of the possible cases so as not to warrant inclusion. Whether the reader agrees with this or not it is clear that the absence of examples makes a clear specification even more important. Most specifications of the IETF are supported by descriptions in a natural language, as well as more formal descriptions and examples. This approach is supported by the IETF document guidelines. Requiring technical knowledge of the audience is largely unavoidable in these areas, thus making the exclusion of technical descriptions less relevant [20].

Experience with implementing PKIX path processing, in particular with regard to the more exotic parts such as policy and constraints, is not yet well developed. There are no test data provided with the path processing algorithm against which implementations can be checked. Due to the slower than anticipated progress in PKI development, most implementations can, at present, ignore these more difficult areas and focus on basic path processing. However, in this case they do not necessarily meet PKIX compliance requirements. As an example, Cryptlib [8] does implement the policies and constraints extensions. The following comment is taken from the Cryptlib code: "Policy constraints are the hardest of all because, with the complex mishmash of policies, policy constraints, qualifiers, and mappings it turns out that no-one actually knows how to apply them." Perhaps the PKIX specification is silent in areas where it should not be.

As is well known, formal methods can not only give a specification for a system in a precise and unambiguous way, but also provide mathematical proof technologies to verify critical properties of the system for guiding the developer towards a design of the security architecture of the system and its implementation. In fact, theorem proving tools and techniques now allow the possibility of high-assurance yet practical reasoning about the behavior of critical systems [7, 9, 41, 44]. Formal methods and techniques would satisfy the requirements in evaluation for higher level of confidence/assurance in PKI liability.

In this paper, we attempt to provide a formal solution to the PKI specification dilemma. We introduce a state-based model for the description of the architecture of a PKI and

related functions, and propose a formal approach to the description of, and reasoning about, the behaviour and security properties of PKIs. We also give a framework for mechanizing our theory in Isabelle [43], a generic theorem prover. In particular, we present the essential Isabelle theories that are used for the certification path validation. In our method, the essential aspects of PKIs can be captured and clearly formulated, which would facilitate the testing of their implementations in a more rigorous and well defined way.

1.2 Related Work

The research on PKIs is recently focused on the following aspects: (1) certificate format and content, (2) certificate revocation, (3) certificate management, and (4) trust models.

For certificate formats, the X.509 standard [22] constitutes a widely-accepted basis for a PKI, defining data formats and procedures related to distribution of public keys via certificates digitally signed by CAs; while PGP (Pretty Good Privacy) certificates are based on an "introducer-model" which depends on the integrity of a chain of authenticators, the users themselves. Both PGP and X.509 define their central role to be played by the verifier regarding certificate acceptance. The SPKI (Simple Public Key Infrastructure) is intended to provide mechanisms to support security in a wide range of Internet applications based on trust models. In a SPKI certificate, the public key is a globally unique identifier of the key-holder. Recently, some researchers have argued that X.509, PGP and SPKI certificates are not well-suited for electronic commerce [29]. To solve the problem, several new certificate formats have been proposed [12, 17, 31]. More recently, Rea [47] proposed a concept of *high value certificates*. High value certification services provide mechanisms that allow agents to go beyond proving who they are and enable them to prove relevant aspects of their character and ability. Also, Levi and Caglayan [32] introduced the *nested certification* concept. A nested certificate is defined as "a certificate for another certificate". Nested certificates can be used together with classical certificates in the public key infrastructures (PKIs). From such a PKI, the authors claim, verifiable nested certificate paths instead of classical certificate paths can efficiently be extracted.

Certificate revocation is a particular area of interest. Current research work such as Naor and Nissim [39] focus on investigating efficient mechanisms for withdrawing a public key or cancelling a certificate when it has become invalid for a certain reason. Revocation of public key certificates is controversial in every aspect: methodology, mechanics, and even meaning. As Fox and LaMacchia [14] have point out, in fact, when it comes to revocation, we cannot even agree on who should do it, how it should work or even what it means. Indeed, in this area there is still a lot of work required to be done.

PKIs are used to generate, distribute, store and verify certificates. The main concern with certificate management of PKIs is what kind of structure a PKI should have for a specific secure communication and how the PKI provides services for managing public key certificates to enable secure applications. Practical discussions on the structure of a PKI, 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 [26, 27] and Trcek [55]. There are also a few attempts to adopt formal methods and techniques for specifying the structure of a PKI and reasoning about its properties [34, 35]. The research on certificate

management also involves the study of techniques for key recovery [18, 19], which is also an interesting area.

The notion of trust is fundamental in PKIs. Suppose that an agent *Alice* wishes to use the key held by *Bob*, then she should have a basic trust that can be applied as the basis for verifying that the key-owner is really Bob himself and other things she may consider. Usually, we say that an agent A trusts another agent B in some respect, which informally means that A believes that B will behave in a certain way – perform (or not) some action(s) in certain specific circumstances. As Jøsang *et. al.* [24] have pointed out, PKIs simplify key management but create trust management problem. With regard to information security, Blaze *et. al.* [1] have identified such trust management problems as a distinct and important component of security in network services. Recently, several trust models with PKIs have also been proposed, which involve the development of effective formalisms used to define and express trust relations between entities involved in a PKI [58], and the investigation of techniques for dealing with trust management [1, 21, 33] and the uncertainty in a trust model [25, 30, 37].

For reasoning about security properties of a system, it is important to provide precise notions that can be used for expressing the structure and behavior of the system. To reach this objective, an appropriate logical language is needed. With regard to information security, there has been a substantial interest in formal methods applied to analysing and designing protocols or security mechanisms. In fact, there have been a number of logics proposed for analysing authentication [6, 9, 52, 56, 54], describing security architectures [2], specifying and verifying cryptographic protocols [44, 53], reasoning about security policies [15], defining trust [46], and so on.

With PKI specification, although there have been a number of documents which involve technical specifications for PKIs [4, 5, 23], there is a distinct lack of formal techniques for describing the structure and behavior of PKIs [42]. Reasoning about a PKI usually relies on the formal representation of the structure and behavior of the PKI. Technical specifications provide a basis for implementing the functions of PKIs, but they cannot well handle reasoning about the security properties of a PKI. Therefore, investigating formal approaches to PKI specifications is still highly desirable.

1.3 Our Approach

In our previous work [34], we have proposed a state-based model for describing the structure and behaviour of a PKI. In this view, a PKI is seen as a state machine where at any state each entity possesses a number of public key certificates issued by some entities (who are potentially permitted to do this) and keeps further information (such as a revocation list). The state may be changed by transitions relevant to PKI functions, e.g, certificate issuing, rekeying and revocation.

In our context, a PKI consists of the entire, generally heterogeneous, set of components that are involved in issuing, rekeying, revoking and managing public key certificates as well as the corresponding private keys. It provides mechanisms for agents to retrieve and possibly to add information to it. Typically, an agent, Alice, can retrieve another agent Bob's public key certificate together with evidence for verifying that certificate.

We identify two essential relations in a PKI: the *certification* relation and *trust* relation. The certification relation is a binary relation that can be informally stated as follows: two agents are certification-related if and only if the policy of the PKI would potentially permit the first agent to issue a certificate to second agent. This relation is usually defined based on the roles that agents (or participants) play in the PKI. For instance, RFC 1422 [28] define a rigid hierarchical structure for the Internet Privacy Enhanced Mail (PEM) [22]. 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 to 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). Users cannot certify anybody. In our method, the concept of a certification topology (see Section 3) of a PKI is used to describe the certification path architecture of the PKI, which consists of an agent set and a certification relation defined over the set.

We provide an approach to the formalization of the architectures for a PKI by axiomatizing the certification topology (defined in Section 3.1), and propose a state-based model for specifying the structure and behavior of the PKI. In principle, the certification path architecture of any given PKI can be formalized as a certification topology that satisfies a certain set of axioms, and the PKI can be specified as a state machine. At any given state, agents in the PKI possess a number of certificates, and also keep information about revoked certificates. Therefore, reasoning about the PKI can be based on the state. This method is generic; it would be suitable for specifying any kind of PKIs.

The trust relation is somewhat different from the certification relation, which is based on the role each participant (agent) plays in the PKI. The trust relation captures agents' beliefs and can be seen as related to a particular type of belief such as the BAN logic [6]. In other words, trust depends on the observer (agent), and there is no absolute trust. Two different agents may not equally trust any received information. A message may carry some information, and different people (agents) may act differently depending whether they believe this information or not.

Linguistically, "trust" is closely related to "true" and "faithful", with a usual dictionary meaning of "assured reliance on the character, the integrity, justice, etc., of a person, or something in which one places confidence." So, in common English usage "trust" is what one places his confidence in, or, expects to be truthful. In a PKI, one of the main concerns for an agent is whether a certificate is trustworthy or not. In managing public key certificates, a PKI provides mechanisms allowing an agent to determine whether a needed certificate can be trusted (or, in the agent's view, that the certificate is valid).

In a PKI, what makes a public key certificate trustworthy? How can one specify and reason about trust? We have to deal with these sorts of questions, so we propose a method for specifying and reasoning about trust in a PKI. The trust assumptions for a given PKI are formalised, with rules for deriving conclusions regarding trust. In our method, an agent can obtain trust in a required certificate through verifying its validity based on the initial trust assumptions. In particular, we discuss the initial **TA** (the *trust axioms*), and initial **TCS** (the *trusted certificate set*). In our model, the initial assumptions with the

trust relation in a PKI is formalized by **TA**, a trust axiom set. Based on trust axioms and by inference rules, an agent can derive and extend his trusted certificate set, which is essential for the certificate verification.

Certificate verification is a central PKI client service. In a PKI, the integrity of each certificate that the security application relies upon must be verified. Verification proves that no one has tampered with the contents of the certificate, so that the verifier may accept the certificate as valid and would trust the public key bound to the subject of the certificate. The certificate verification service performs the verification process and may store verified certificates in the local database for later use.

Certificate verification has two components – certificate *path development* and *path validation*. Path development is used to find a certification path and provide it to path validation, while path validation takes the responsibility to verify the path as valid or identify where and why the path validity fails. An essential principle we give for certificate verification is that an agent will accept a target (required) certificate as valid if the agent can find a certification path starting with a target certificate and ending with a certificate trusted by itself and can prove that all certificates on the path are valid, otherwise the certificate is regarded as invalid. We propose an algorithm for path development which is based on a given PKI state and the trusted certificate set of the verifier, and present a framework for path validation based on a method that separates the checks of certificates into three categories – single checks, pair checks and path checks.

We also propose a framework for mechanizing our theory in Isabelle, a generic theorem prover [43], and the essential Isabelle theories for path validation are provided. We also take the PKIX path validation algorithm in RFC 2459 and give a formal description of some of the actions and properties it defines. In this way, the essential aspects can be captured and clearly formulated, which would facilitate the testing of other implementations in a more rigorous and well defined way. The formal description of precisely what is required for conformance could also assist implementers during the design phase. We are not suggesting that our specification replace the one already given in RFC 2459 but as a supplement that may be used to improve its clarity. The framework for path validation we develop would also be applicable to other PKIs in general.

In summary, the contributions of this paper are as follows:

- A state-based model for specifying and reasoning about the properties of a PKI is proposed, and a formalization approach to the description of the structure and behavior of a PKI is also provided.
- A method for representing and reasoning about trust in a PKI is proposed. In particular, we propose a set of certificate verification rules which an agent can apply for verifying a certificate it requires.
- For the certificate verification function, we propose an algorithm for path development and present a framework for path validation based on a particular method.
- We also propose a method for mechanizing our theory in Isabelle, which would allow us to automatically perform reasoning processes, supporting certificate verification.

The rest of the paper is structured as follows. Section 2 provides a brief background to PKI elements relevant to our discussion, and a standard public key certificate format is discussed. Section 3 introduces our state-based model for PKIs. We discuss the PKI topology, PKI states and transitions, and an approach to the formalization of PKIs. Section 4 discusses the certificate verification process, and presents a set of certificate verification rules. Section 5 discusses two components of the certification verification function: path development and path validation. In Section 6, we discusses mechanization of our theory in Isabelle/HOL, and present the essential theories used for path validation. Section 7 proposes a formal technique applied for modelling PKIX path validation that involves processing policy extension fields. The last section concludes this paper with a brief discussion about possible future work.

2 Public Key Certificates

The PKI entities, which we sometimes call agents, are classified into two classes: *Certification Authorities* (CAs)[1] and *Users*. CAs can have their own certificates, and they also issue certificates for others within the PKI. Users, also called *End Entities* (EEs), are people or devices that may hold certificates issued by some CAs, but may not issue valid certificates themselves.

We assume that the reader is familiar with the use of certificates to convey public keys. For the details on public key cryptosystems, we refer the reader to Ford and Baum [13] and Schneier [51]. Here, we present a formal definition for public key certificates, giving a "standard" format of a certificate. As we can see, our definition is particularly appropriate for modelling cryptographic functions involved in public key certificates.

Public key certificates can be viewed as a vehicle by which public keys may be stored, distributed or forwarded over unsecured media without danger of undetectable manipulation. The objective is to make one entity's public key available to others such that its authenticity (i.e., its status as the true public key held by that entity) and validity are verifiable. Formally, we have: A *public key certificate* is a data structure consisting of a *data part* and a *signature part*. The data part contains cleartext data including, as minimum, a public key, and a name aiming to identify the party (subject) controlling the associated public key. The signature part consists of the digital signature by a certification authority over the data part, thereby binding the subject's identity to the specified public key. The data part may include additional information as follows: the serial number of the certificate, the validity period of the public key, the issuer's name, additional information about the subject or the issuer (e.g. where to find issuer's certificate), additional information about the key, the quality measures (which are related to the identification of the subject, the generation of the key pair, or other policy issues, e.g. certification policy), information facilitating verification of the signature (e.g. a signature algorithm identifier, issuing CA's name etc), the status of the public key (is revoked or not), and so on.

Without loss of generality, we assume that PKI certificates have a "standard" public-

[1]We do not consider *Registration Authorities* (RAs) as separate entities. RAs carry out parts of the CA function, and are logically part of the CA, but are implemented elsewhere for performance, cost and usability reasons.

key certificate format, which contains the basic information that most kinds of public key certificates should provide as follows: the name of the certificate issuer, the start and expiry dates, the subject (i.e., the name of the holder of the private key for which the corresponding public key is being certified), the value of the public key, the extension field, and the signature of the issuer. Formally, we define a PKI certificate to have the following form:

$$\texttt{Cert}\ (\texttt{I}, \texttt{DS}, \texttt{DE}, \texttt{S}, \texttt{PK}, \texttt{E}, \texttt{Sig})$$

where I is the issuer, DS and DE are the start date and expiry date respectively, S is the subject of the certificate, PK is the value of the public key for S, E is the value of the extension field, and Sig holds the signature of the issuer I.

Given a certificate

$$C = \texttt{Cert}\ (\texttt{I}, \texttt{DS}, \texttt{DE}, \texttt{S}, \texttt{PK}, \texttt{E}, \texttt{Sig})$$

the following projection functions can be used to obtain the value of each component contained in C:

$$\overline{\texttt{I}}(C) = \texttt{I} \qquad \overline{\texttt{DS}}(C) = \texttt{DS} \qquad \overline{\texttt{DE}}(C) = \texttt{DE}$$
$$\overline{\texttt{S}}(C) = \texttt{S} \qquad \overline{\texttt{PK}}(C) = \texttt{PK} \qquad \overline{\texttt{E}}(C) = \texttt{E}$$
$$\overline{\texttt{Sig}}(C) = \texttt{Sig}$$

The public key $\overline{\texttt{PK}}(C)$ is bound to the entity $\overline{\texttt{S}}(C)$, the subject of the certificate. The private key corresponding to the public key $\overline{\texttt{PK}}(C)$ is denoted by $SK(\overline{\texttt{PK}}(C))$. Therefore, the key pair possessed by the subject is $(\overline{\texttt{PK}}(C), SK(\overline{\texttt{PK}}(C)))$.

The extension field could include

- an "`authorityKeyIdentifier`" extension providing a means for identifying the particular private key used to sign the certificate, and
- a "`subjectKeyIdentifier`" extension that enables differentiation of keys held by the subject.

There are no requirements for PKI implementations to process these extensions. However, for our purposes, we assume that in the certificate process, for a given certificate C, the identifier of the certificate authority's key and the identifier of the certificate subject's key can be identified by the extensions `authorityKeyIdentifier` and `subjectKeyIdentifier`, respectively.

In practice, a PKI may use a certificate format different from the standard format given above. However, any PKI certificate format should consist of two parts: the data part and the signature part of the certificate issuer. In the standard public key certificate format, $\overline{\texttt{Sig}}(C)$ is the signature part of the certificate C, while the data part is a combination of the values of all other components to be signed. We write `tbs` representing "to be signed", and define:

$$\mathtt{tbs}(C) = (\mathtt{I}, \mathtt{DS}, \mathtt{DE}, \mathtt{S}, \mathtt{PK}, \mathtt{E}),$$

then $\mathtt{tbs}(C)$ is just the data part of the certificate, and thus ought to have been the argument to signature function carried out by the certificate issuer.

3 The State-Based Model

In this section, we present a state-based model for PKIs, in which a PKI is formally specified by its certification topology and PKI states. We adopt an axiomatic approach to PKI specifications: the certification topology of a PKI is defined by a set of *agent axioms* and PKI states are defined by *state axioms.*

3.1 Certification Topology

In our model, we make the following assumptions about a given PKI:

- There is a static set of *agents* (CAs and EEs).
- The domain of each agent which the agent may potentially certify is defined;
- Each agent may initially possess zero or more certificates; and
- Each agent can only revoke a certificate that is issued by itself.

We use in the following notations:

- $A, B, A_1, A_2, \ldots$ agent variables, $alice, bob, \ldots$ agent constants.
- $C, C_1, C_2, \ldots$ certificate variables, $c, c_1, c_2, \ldots$ certificate constants.
- $PK, PK_1, PK_2, \ldots$ public key variables, pk, $pk_1, pk_2, \ldots$ public key constants.
- $SK, SK_1, SK_2, \ldots$ private key variables, sk, $sk_1, sk_2, \ldots$ private key constants.
- $T, T_1, T_2, \ldots$ time variables, $t, t_1, t_2, \ldots$ time constants, a special time variable, denoted as $Today$, represents the current time.

Other variables and constants may be introduced when they are needed. The functions including all projection functions: $\overline{\mathtt{I}}(C)$, $\overline{\mathtt{S}}(C)$, $\overline{\mathtt{DS}}(C)$, $\overline{\mathtt{DE}}(C)$, $\overline{\mathtt{PK}}(C)$, $\overline{\mathtt{Sig}}(C)$, and $\mathtt{tbs}(C)$ will be used through the paper.

We also use $K, K_1, \ldots$ to represent public key pairs. A key pair has the form $K = (PK, SK)$ where PK and SK are the public and private keys corresponding to the key pair. We also use a specific variable $\widehat{RK}$ to represent either a public key or a private key. Thus, we have the following notations to define encryptions and decryptions: $\{M\}_{\widehat{RK}}$ represents a message M encrypted under the key $\widehat{RK}$, and $\langle M \rangle_{\widehat{RK}}$ represents M decrypted under the key $\widehat{RK}$.

All the agents contained in the PKI will be organized based on a certain relation, called the certification relation. Thus, we formalize the certification architecture of the PKI as a certification topology defined below.

Definition 1 *The* certification topology *of a PKI is a pair of the form* $\langle \Omega, \downarrow \rangle$*, where* Ω *is the set of all agents in the PKI,* $\downarrow$ *the certification relation, a binary relation over* Ω *satisfying a certain set of agent axioms. For any* $A, B \in \Omega$*,* $A \downarrow B$ *represents the fact that the agent* B *is within the domain of agents which* A *may potentially certify (called the certification domain of* A*) under the security policy of the PKI.*

In our model, for any given PKI, the certification topology is assumed to be fixed. That is, we assume that the agents and the certification relation between agents are static. In practise, dynamic changes do occur from time to time, such as an employee leaving a company (or is yet to join). This situation is covered in our model by the assumption that an agent may not possess a certificate.

The certification relation of a PKI has an intuitive direct graphical representation: all agents (i.e., CAs and EEs) are represented as nodes and, for any $A, B \in \Omega$, there is a directed edge from A to B if and only if $A \downarrow B$ holds. In some PKIs, CAs can certify each other, i.e., there may therefore be some bidirected edges in the graphical representation of their certification topologies. Some examples of certification topologies are shown in Figure 1.

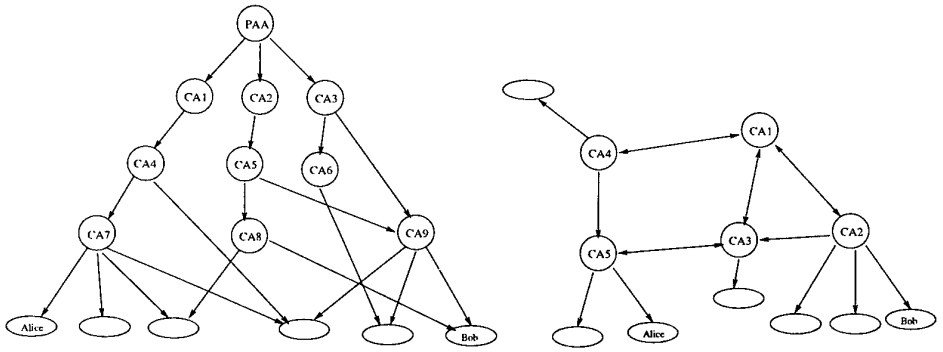


Figure 1: Examples of certification topologies

The certification relation, and associated topology, reflect the intended structure of a PKI for a given organisation, e.g. a hierarchical organisation may desire a hierarchical PKI to support their business processes.

Now the question arises: how do we obtain the agent axiom set for a given PKI? Or, in other words, how do we axiomatize the certification topology of a given PKI?

For a given PKI or class of PKIs, the agent axiom set may include:

- some facts having the form $A \downarrow B$;

- rules from which some facts of the form $A \downarrow B$ may directly be derived.
- constraint rules that define the roles of agents or describe the properties that the relation $\downarrow$ satisfies.

In obtaining the agent axiom set, we must follow two principles: one is *consistency*, i.e., the axiom set must be consistent; the other is *completeness* to guarantee that, for all $A, B \in \Omega$, $A \downarrow B$ can be derived from the axiom set if and only if B belongs to the certification domain of A.

The facts can be easily obtained based on the certificate domain of each agent in a given PKI. For instance, assume that ca_1 is a manager whose certification domain is $\{alice, bob\}$, then the facts $ca_1 \downarrow alice$ and $ca_1 \downarrow bob$ may belong to the agent axiom set. Note that we do not need to put all facts of the form $A \downarrow B$ to the agent axiom set because some of them may be derived from it using inference rules.

Those agent axioms represented as constraint rules determine the type of PKI. For instance, we now define the axiom set that characterises the class of hierarchical certification topologies. In such a PKI, all CAs and users are arranged hierarchically under a "root" CA, (called the Policy Approval Authority and denoted as *paa*), that issues certificates to subordinate CAs. These CAs may issue certificates to CAs below them in the hierarchy, or to users. In such a PKI, the topology has the following rules (agent axioms) to define the roles of agents:

(A1) $\forall A \bullet (\texttt{IsCA}(A) \vee \texttt{IsEE}(A))$
(A2) $\forall A \bullet (\texttt{IsCA}(A) \leftrightarrow \exists B \bullet (A \downarrow B))$
(A3) $\forall A \bullet (\texttt{IsEE}(A) \leftrightarrow \forall B \bullet \neg(A \downarrow B))$
(A4) $\forall A \bullet (A = paa \leftrightarrow (\texttt{IsCA}(A) \wedge \neg\exists B \bullet (\neg(B = A) \wedge (B \downarrow A))))$

Here, $\texttt{IsCA}(A)$ means A is a CA, and $\texttt{IsEE}(X)$ means A is an end entity. The meanings of all these axioms are obvious: (A1) says that every agent is a CA or an EE; (A2) says that any agent is a CA if and only if there exists an agent belongs to its certification domain; (A3) says that an agent is an EE if and only if no one belongs to its certification domain (i.e., its certification domain is empty); (A4) says that an agent is the unique top CA, *paa*, if and only if it is a CA and it does not belong to the certification domain of any other agent than itself.

The axioms describing the properties that the relation $\downarrow$ should satisfy are:

(A5) $\forall A \bullet (\neg(A = paa) \rightarrow \neg(A \downarrow A))$
(A6) $\forall A \bullet \forall B \bullet (\neg(B = paa) \wedge (A \downarrow^+ B) \rightarrow \neg(B \downarrow^+ A))$

Axiom (A5) says that, except the *paa*, there is no agent who belongs to the certification domain of itself. (A6) is applied to guarantee that there are no cycles other than self-cycle of the *paa*. The axiom (A6) includes a new relation operator $\downarrow^+$ as a transitive closure of the relation $\downarrow$, which is defined by the following formula:

(A7) $\forall A \bullet \forall B \bullet ((A \downarrow^+ B) \leftrightarrow (A \downarrow B) \vee \exists Z \bullet (A \downarrow^+ Z \wedge (Z \downarrow^+ B))$

Thus, for a hierarchical PKI, the agent axiom set consists of the axioms (A1) – (A6) and those axioms from which all facts of the form $A \downarrow B$ containted in the PKI can be derived.

It is not necessary that all these axioms are satisfied by any other kind of PKIs. For example, in a mesh PKI, independent CAs may issue certificates to each other, resulting in a general mesh of trust relationship between peer CAs. A relying party knows the public key of a CA "near" himself, generally the one that issued his certificate. That is, in general an agent trusts the CA that can issue a certificate to himself. Therefore, such a PKI does not need to satisfy the axiom (A4) and (A6).

In summary, in our model, the certification topology of a given PKI can be defined as a set of agents with a binary relation, the certification relation, which can be described by an agent axiom set.

3.2 PKI States

We define the *total certificate set* of a given PKI as the set of all certificates issued by CAs in the PKI.

At any moment in time, an agent in the PKI should possess zero or more certificates. Also, for a CA, it is at times necessary to revoke certificates, for example when the certificate holder leaves the issuing organization or when the private key is compromised. The mechanism defined in X.509 for revoking certificates is the *Certificate Revocation List* (CRL). A CRL is a list of unexpired, revoked certificates, which is signed by a CA periodically. In our model, we assume that, at any time, any agent is associated with its CRL. However, we should note that, if the agent is an EE, the CRL should be empty, because an EE does not issue any certificates to others and cannot revoke any certificates either. Thus, we define PKI states as follows:

Definition 2 *Let $\langle \Omega, \downarrow \rangle$ be a certification topology of a PKI, and $\mathcal{C}$ its total certificate set. A* state *s of the PKI is a relation from Ω to $2^{\mathcal{C}} \times 2^{\mathcal{C}}$ satisfying the axioms:*

(S1) $\forall A \bullet \exists \zeta \bullet \exists \eta \bullet (s(A, \zeta, \eta) \wedge \forall \zeta' \bullet \forall \eta' \bullet (s(A, \zeta', \eta') \rightarrow (\zeta' = \zeta) \wedge (\eta' = \eta)))$

(S2) $\forall A \bullet \forall C \bullet (s(A, \zeta, \eta) \wedge C \in \zeta \rightarrow \overline{\mathtt{S}}(C) = A)$

(S3) $\forall A \bullet \forall C \bullet (s(A, \zeta, \eta) \wedge C \in \eta \rightarrow \overline{\mathtt{I}}(C) = A)$

(S4) $\forall A \bullet \forall \eta \bullet (s(A, \zeta, \eta) \wedge \mathtt{IsEE}(A) \rightarrow \eta = \phi)$

where $2^{\mathcal{C}}$ is the power set of $\mathcal{C}$ and ϕ is the empty set. Under a state s, we call $s(A, \zeta, \eta)$ a triple, where $\zeta (\subseteq \mathcal{C})$ is a set of certificates issued to A and $\eta (\subseteq \mathcal{C})$ is a set of certificates issued by A.

(S1) – (S4) are called state axioms. According to the definition above, at any given state s (referred to as the current state), any agent A is associated with two certificate sets: ζ and η. All certificates in ζ have been issued to A because their subject is A; and all certificates in η have been issued by A itself at sometime because their issuer is A. Thus, we have

Definition 3 *Let s be a PKI state. If we have $s(A, \zeta, \eta)$, then ζ is called the* possessed certificate set *of A, which lists all certificates for A, and η is called the* revoked certificate set *of A, which represents the CRL issued by A at the state s. In particular, if $C \in \zeta$, we say that A possesses the certificate C; if $C \in \eta$, we say that A has revoked the certificate C.*

In the following, for convenience, we will use $\mathbf{PCS}_A$ and $\mathbf{CRL}_A$ to denote the possessed certificate set and the revoked certificate set of the agent A at a given PKI state, respectively.

3.3 Transitions

In the theory of state machines, transitions are usually described as actions that change states of a machine. The state of a PKI can also be changed by applications of some PKI functions, such as certificate issuing, certificate rekeying and certificate revocation. These actions could be viewed as transitions which change one PKI state into another. Thus, a PKI can be described as a state machine. Adopting the formal approach to defining a state machine proposed by Eastaughffe *et. al.* [11], we define transitions of the PKI as follows:

Definition 4 *For a given PKI, a transition 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 an agent B belongs to the certification domain of agent A and, at the current state s, we have $s(B, \zeta, \eta)$ and now a transition involving solely the issue by A of a new certificate to B, then the transition might be defined as:

Let: $\mathrm{C} = \mathtt{Cert}\,(A, \mathrm{D}_s, \mathrm{D}_e, B, \mathrm{PK}, \mathrm{E}, \mathrm{SIG}_\mathrm{Y})$
Pre: $\mathtt{SucReqCert}(B, A)$
Act: $\zeta \leftarrow \zeta \cup \{C\}$

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

We define a simple transition as one of the following actions:

- Issuing of a certificate C to an agent B by an agent A;
- Revoking of a certificate C by an agent A.
- Rekeying of a certificate C possessed by an agent A.

All simple transitions can be formalized in the same way given above. A transition of a PKI 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 $Tran_1$, ..., $Tran_n$ are simple transitions, $Tran_1 \| \ldots \| Tran_n$ is a compound transition, where $\|$ is called the parallel operator over transitions.

Dealing with PKI state changes due to application of PKI functions such as certificate issuing, certificate revocation and certificate rekeying is beyond of the scope of this paper. That is, we are not concerned with PKI state changes which result from protocol execution. In the following, we focus on discussing the certificate verification function based on a static state, so we do not attempt to discuss the formalization of transitions in details. For this, we refer the reader to Liu *et. al.* [34].

4 Certificate Verification Rules

Certificate verification is a major PKI client function, which is responsible for verifying the validity of every certificate that a security application uses. In this section, we present a set of certificate verification rules which an agent uses to derive its conclusion regarding the validity of a certificate required.

4.1 The Process

Assume that agent Alice wants to retrieve Bob's certificate together with evidence used for checking if the certificate is valid. The certificate possessed by Bob carries meaning which asserts that, for example, "I am Robert Jones and have public key pk (and control the corresponding private key sk) which is bound to a certificate that is issued by ca_1 and valid from 12th May 2000 to 31st October 2000". If Alice trusts Bob, she may believe that Bob's certificate is valid and this statement it asserts. In particular, Alice believes that Bob's public key is really Bob's, so that she can use Bob's public key to decrypt any messages signed with Bob's private key.

However, if Alice did not securely obtain Bob's certificate, she should verify Bob's certificate before she uses it. In this case, Alice may employ the following procedure to determine whether to trust Bob's certificate or not:

- verifying the identity of the certificate issuer and owner (checking if Bob is the owner, and checking if Bob belongs to the certification domain of the issuer);
- verifying the validity dates of the certificate;
- verifying the certificate against the issuer's latest CRL list to make sure it has not been revoked;

- verifying extension fields (such as `certificatePolicies`) if necessary; and

- verifying the signature on the certificate.

In order to verify the signature on Bob's certificate, Alice needs to check if the issuer holds a valid certificate, or, more accurately, Alice must verify the certificate which is held by the issuer and used to sign Bob's certificate in the same way. Therefore, the verification process Alice uses is iterative. She cannot accept Bob's certificate as valid unless she reaches a certificate in the verification procedure that she already trusts.

This indicates that, if Alice has no trusted certificates, she cannot prove that Bob's certificate is valid or her proof process can never terminate. Therefore, an agent in a PKI who is involved in certificate verification activities should keep a trusted certificate set of all certificates trusted by the agent.

4.2 Assumptions about Trust

When dealing with trust issues, we need to answer basic questions: what makes a public key certificate trustworthy? how can one specify and reason about trust? In this paper, rather than use a belief logic [6], we directly suggest several trust assumptions that might be accepted by all agents, and present a set of inference rules which agents can apply to reason about trust.

In a PKI, the operations that CAs may execute include: issuing, revoking and rekeying certificates. Within it, the following assumptions concerning trust may be accepted:

(1) All agents (CAs and users) trust all CAs to follow policy in their CA operations; and

(2) All agents trust that it is not feasible to undetectably tamper with PKI certificates.

These assumptions are well founded and supported by PKI practices and technology. Firstly, assurance is provided for (1) through the use of accreditation of CAs, Certificate Practice Statements published by CAs and the application of appropriate policy[2]. Assurance is provided for (2) through the use of digital signatures, and strong control of private keys.

We do not attempt to axiomatize the above assumptions. So as to focus on the certificate verification in the following, we only formalize those assumptions which are directly related to deriving trusted certificates for agents.

In our model, the trust assumptions regarding trusted certificates are formalized as a set of trust axioms. For instance, suppose that in a hierarchical PKI we have this assumption:

- All agents trust the certificate possessed by *paa*, the top CA.

[2]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.

We introduce a predicate Trusts: "A Trusts C" means that agent A trusts the certificate C. Let *cert_paa* be the *paa*'s certificate, then the above assumption can be formalized as the following axiom:

$$\forall A \bullet (A \texttt{ Trusts } cert\_paa).$$

Given the topology $\langle \Omega, \downarrow \rangle$ of a PKI, we define the *trusted certificate set* of an agent A in Ω, denoted as $\mathbf{TCS}_A$, as the set containing all certificates trusted by the agent. The question is: how does the agent A derive a trusted certificate set for use in the certificate verification?

Let **TA** is the set of trust axioms for a given PKI, then for any agent A and any certificate C, if A Trusts C can be derivable from **TA**, then $C \in \mathbf{TCS}_A$. Thus, the agent A can obtain its initial trusted certificate set based on the axiom set **TA**.

4.3 Alice's Verification Rules

We now discuss certificate verification rules. A rule expressed as

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

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.

Let a predicate $\texttt{Valid}_A(X)$ denote that X is valid (in an agent A's view), where X is a certificate or the signature of a certificate or a public key or a key pair. Without confusion, we may simply write $\texttt{Valid}(X)$ instead of $\texttt{Valid}_A(X)$. As we said before, if an agent A, say Alice, trusts a certificate C, then A in fact believes that C is a valid certificate and, therefore, she believes that the public key bound to the subject of C is valid. She may also therefore believes that the key pair consisting of the public key and the private key corresponding to this public key is valid. Thus, in the verification procedure, apart from those rules of inference in the first-order logic, agent Alice, may employ the following verification rules:

(R1) $$\frac{A \texttt{ Trusts } C}{\texttt{Valid}_A(C)}$$

(R2) $$\frac{\texttt{Valid}_A(C)}{\texttt{Valid}_A(\overline{\texttt{PK}}(C))}$$

(R3) $$\frac{K = (\overline{\texttt{PK}}(C), SK(\overline{\texttt{PK}}(C))) \quad \texttt{Valid}_A(\overline{\texttt{PK}}(C))}{\texttt{Valid}_A(K)}$$

If agent Alice believes that a key pair $K = (PK, SK)$ is valid, then she believes that PK and SK must satisfy that, for any message M, both $\langle\{M\}_{SK}\rangle_{PK} = M$ and

$\langle\{M\}_{PK}\rangle_{SK} = M$ hold. Therefore, she may use the public key PK to decrypt messages encrypted by the private key SK. Note that, if Alice does not control SK (i.e., she is not the ower of K), she cannot decrypt messages encrypted by the public key PK.

Thus, Alice may use the following rules to varify the certificate C when she believes that C' is valid.

$$\text{(R4)} \qquad \frac{\mathtt{Valid}_A(C') \;\; \mathtt{tbs}(C) = \langle\overline{\mathtt{Sig}}(C)\rangle_{\overline{\mathtt{PK}}(C')}}{\mathtt{Valid}_A(\overline{\mathtt{Sig}}(C))}$$

$$\text{(R5)} \qquad \frac{\mathtt{Valid}_A(\overline{\mathtt{Sig}}(C)) \;\; \mathit{Today} \geq \overline{\mathtt{DS}}(C) \;\; \mathit{Today} < \overline{\mathtt{DE}}(C) \;\; \neg(C \in \mathbf{CRL}_{\overline{\mathtt{I}}(C)})}{\mathtt{Valid}_A(C)}$$

Note that digital signature algorithms usually involve use of a hash function. However, for simplifying our discussion, we do not consider this. So, in the rule (R4), verifying the signature of the certificate C only needs to check whether $\mathtt{tbs}(C) = \langle\overline{\mathtt{Sig}}(C)\rangle_{\overline{\mathtt{PK}}(C')}$ holds when C is signed by $SK(\overline{\mathtt{PK}}(C'))$ and the agent believes that the certificate C' is valid.

Agents may also use the following rule to extend their trusted certificate sets which they have already obtained.

$$\text{(R6)} \qquad \frac{C_1 \in \mathbf{TCS}_A \;\; \mathtt{Valid}_A(C_2) \;\; \overline{\mathtt{I}}(C_2) = \overline{\mathtt{S}}(C_1)}{C_2 \in \mathbf{TCS}_A}$$

If a certificate C is trusted (in an agent A's view), then, by the rule (R1), we know that A is sure that C is valid. Conversely, if the agent has proved that a certificate is valid, then (R6) may allow the certificate to be entered into the trusted certificate set. However, A may choose not to put the certificate into this set even (R6) applies. So a trusted certificate is cached, and irrevocable for a period. What make the difference between trusted and valid certificates? Essentially, for a trusted certificate, the agent has decided to place his trust in it and he definitely believes it (at least for a certain time period) – he does not need to check the validity of this certificate when he wants to use it. On the other hand, for a valid certificate, the agent just accepts it as being valid in certain specific circumstances for temporary use. However, he does not believe that the certificate can always be trusted, so he may need to check the validity of this certificate again when he uses it in any other time. Whether or not to apply (R6) depends on other external factors outside of our model, such as evidence of the trustworthiness of the subject of A, or commercial/performance issues requiring paths to be kept short.

5 Path Development and Path Validation

Focusing on the certificate verification function, in this section we discuss how to build and validate a certificate path in the verification procedure.

5.1 The Principle of Certification Verification

A certification path is a chain of certificates, starting from a certificate which is trusted by the verifier. Each certificate in the path is signed by its predecessor's key. A relying party verifies a signature by successively verifying the signatures on the certificates in the path. The certification path is an essential architectural construct of a PKI. Formally, we have

Definition 5 *A* certification path *is a non-empty sequence of certificates* $\langle C_0, \ldots, C_n \rangle$, *where* C_0 *is the* target *certificate,* C_n *is a* trusted *certificate, and for all* i $(0 \leq i \leq n-1)$ *the subject of* C_{i+1} *is the issuer of* C_i. *The* trusted certificate *is viewed as a certificate that is trusted by the verifier and, according to Alice's verification rule (R1), it should of course be a valid certificate (in the verifier's view), and the* target certificate *is the one that the verifier wants to verify.*

The basic idea of constructing such a certification path for verifying a certificate is: if C_n is a trusted certificate, it must be valid (in the verifier's view), therefore, the public key pair associated with C_n is valid. Thus, the public key bound to the subject of C_n can be used to check if the signature on C_{n-1} is accepted as valid. Similarly, if C_{n-1} is valid, then the public key bound to the subject of C_{n-1} can be used to check if the signature on C_{n-2} is accepted as valid. The procedure will continue until some flaw on a certificte is found and the verifier cannot therefore accept the target certificate C_0 as valid, or when the validity of C_0 is proved and the verifier accepts it as a valid certificate.

The *certificate validation* procedure takes a given certification path and determines whether the target certificate is valid or invalid. A verification process for a given certificate involves obtaining and verifying the certificates from a trusted certificate to the target certificate. Obtaining the certificates is referred to as *certificate path development* and checking the validity of the certificate path is referred to as *certificate path validation.*

Central to the certificate verification is the following

Certificate verification principle: *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.

In summary, the cerificate verification function consists of two components: *path development* and *path validation*. The objective of *path development* is to find a certification path and provide it to the *path validation* process. *Path validation* must either identify the path as valid, or it must identify where the path fails. When *path validation* has identified where the path fails, it may send the information back to *path development* for finding an another alternative path. This process repeat may either until a valid path is obtained and is verified, or it gives up, perhaps because *path validation* has identified all paths provided by the *path development* as invalid and the *path development* cannot find any suitable alternative paths. Figure 2 illustrates the interaction between the two components of the verification function within a PKI state machine.

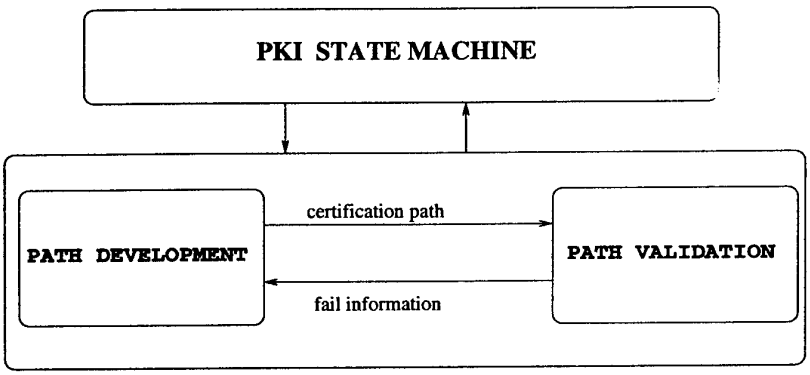


Figure 2: Two components of the verfication function

5.2 Path Development Algorithm

How the certification path is obtained is dependent on the structure of the PKI. However, in any PKI, starting with the target (required) certificate and building a certificate chain back towards a certificate trusted by the verifier is usually an efficient means for developing a certificate path.

A sequence of certificates starting with the target certificate constructed in developing a certification path may eventually be a part of some certification path; however, in some cases, it may be discarded as not being a part of any certification path if the verifier could not reach any trusted certificate along this sequence. In either case, the verifier may need to check whether there is any possibility to reach a trusted certificate along such a sequence of certificates, such that a certification path can be constructed.

We assume that, at a given state, C_0 is the target certificate required by the agent A, and $\mathbf{TCS}_A$ is the trusted certificate set of the agent A. Recalling the notations for possessed and revoked certificate sets, we have: for any agent B, $\mathbf{PCS}_B$ represents B's certificate set, and $\mathbf{CRL}_B$ represents B's revoked certificate set.

Thus, for a hierarchical PKI, the agent A, as a verifier of the certificate C_0, may adopt the following algorithm in developing a certification path by constructing candidate paths step by step.

1. Set $i = 0$, $P = \langle C_0 \rangle$ and $\zeta_0 = \mathbf{PCS}_{\overline{\mathrm{I}}(C_0)}$;
2. If $C_i \in \mathbf{TCS}_A$, return $P = \langle C_0, \ldots, C_i \rangle$ as the certification path, then stop; otherwise
3. If ζ_i is not empty, choose C_{i+1} from ζ_i, set $\zeta_i = \zeta_i - \{C_{i+1}\}$, $P = \langle C_0, \ldots, C_i, C_{i+1} \rangle$, and $\zeta_{i+1} = \mathbf{PCS}_{\overline{\mathrm{I}}(C_{i+1})}$, then reset $i = i + 1$ and go to Step 2; otherwise
4. If ζ_i is empty, and $i > 0$, delete the last element in P, i.e., reset $P = \langle C_0, \ldots, C_{i-1} \rangle$, then set $i = i - 1$, go to Step 3; otherwise
5. If ζ_i is empty and $i = 0$, return fail (which means that no certification path is found), then stop.

By this method, the verifier A may construct all possible certification paths starting with the target certificate and ending with a trusted certificate. All these certification paths can be used for verifying the target certificate. However, he may find that there are no such certification paths, in which case, he cannot accept the certificate as valid.

In particular, if A only trusts the certificate held by the PAA in a hierarchical PKI, then any certification path constructed by him is always a certification path starting with the certificate held by the PAA.

Since the certificate set of an agent may contain multiple certificates, without the use of the key-identifier information, certification path development becomes increasingly complex as the number of paths that need to developed may grow exponentially. To avoid this, in Step 3 "choose C_{i+1} from ζ_i", we may use the key-identifier information to reduce the number of choices.

5.3 A Framework for Path Validation

Given a certification path $\langle C_0, \ldots, C_n \rangle$, the path validation process needs to verify, among other things, that the path satisfies the following conditions:

1. for all i $(0 \leq i < n)$, the issuer of C_i is the subject of C_{i+1};
2. certificate C_n is a trusted certificate (in the verifier's view); and
3. for all i $(0 \leq i < n)$, the certificate C_i is valid at the time in question (usually referring to the current time). This means that: (1) no changes occur in the certificate (this proof can be done by signature checking), (2) the current time belongs to the valid time period shown on the certificate, and (3) the certificate has not been revoked.

In practice, for example, with X.509 certificates each extension field can be marked as *critical* or *non-critical*. Any extension can be ignored but in the case of critical extensions, the issuing CA will take no responsibility for use of the certificate. Note that an Object IDentifier (OID) is a unique sequence of positive integers used to distinguish objects. Each extension is associated with an OID defined in X.509. The extensions that are important to our discussion are:

- `CertificatePolicies`: contains a sequence, possible empty, of policy OIDs. The purpose, as stated in PKIX, is that in a certificate these policy information terms are used to indicate the policy under which the certificate was issued and for which the certificate may be used;

- `PolicyMapping`: is used in CA certificates to link separate certificate domains by providing a mapping of policies of one domain to policies of another domain.

- `PolicyConstraints`: consists of two further fields:

  (1) `InhibitPolicyMapping`: a positive integer which indicates how many additional certificates may occur in a path before policy mappings are no longer allowed;

  (2) `RequireExplicitPolicy`: a positive integer which indicates how many additional certificates may occur in a path before an acceptable policy OID will be required in each certificate.

An application is expected to have its own list of acceptable policy OIDs by which it can compare with the list of policy OIDs in a certificate. Some applications may not require any particular policy at all.

Therefore, in practice, the path validation process may also need to determine the set of certificate policies that are valid for this path, based on the certificate policies extension, policy mapping extension, and policy constraints extension.

The path validation process should be prepared to verify the validity of all certificates contained in the given path. It seems that we can start the process from any certificate or can process them in any order we wish. However, it is more convenient to start the process from the certificate C_n, the trusted certificate in the verifier's view, because it is directly accepted by the verifier as valid. He may then accept the signature with the corresponding private key held by C_n's subject on the certificate C_{n-1} as valid. Thus, he can verify the validity of C_{n-1}, and if the validity of C_{n-1} has been verified, then he can verify C_{n-2} in the same way, and so on. We will take this order for the path validation process.

We now introduce a path validation framework, which makes important and natural distinctions between the types of checking done for validating certification paths.

In our framework, all of the certification path validation checks can be placed into one of the three categories.

Single checks: these are checks that are performed on and only involve a single certificate (as is the case with the `Validity` field (against the start date and expiry date), for example).

Pair checks: these checks require the comparison of two possibly different fields from two certificates. For example, the `Issuer` field of the certificate C_i and the `Subject` field of the certificate C_{i+1} (where $0 \leq i \leq n-1$) need to be the same.

Path checks: for these checks the entire certification path $\langle C_0, \ldots, C_n \rangle$ may be required. If the certificate C_{i+k} holds the value k in the `requireExplictPolicy` field of the `PolicyConstraints` extension then it must be checked that the certificates C_i onwards have an acceptable policy in their `Policy` extension fields and concurrently checked whether the value k is decreased by any certificates further along the path.

The mechnisms dealing with various different checks will be discussed in the next section.

This framework is the basis for a natural separation of the entire certification path validation problem into distinct types mainly based on the different checking requirements. Actually, there are several implementations that follow this structure, although usually not emphasised in the documentation.

6 Mechanizing Our Theory in Isabelle

Isabelle [43, 45] is a generic theorem prover that can be used for implementing a range of logical formalisms. We have shown that the structure and behavior of a PKI can be formalized in a logical formalism (see Section 3), so it becomes possible to mechanize our theory in Isabelle. We use Isabelle/HOL, an instantiation of Isabelle relying on higher-order-logic (Isabelle can be instantiated to various base logics). The essence of higher-order logic is that both functions and predicates (of the appropriate type) can take functions or predicates as arguments, and return them as results [16].

In this section, we present the essential theories that are used for path validation based on a particular PKI with a hierachical architecture. The validation of certificate policies is the hardest part, which we leave and discuss later in Section 6.5. That is, in this initial presentation, path validation only deals with single checks and pair checks.

6.1 Certificate Theory

In Isabelle, a theory consists of the definition of types, functions including the fact that functions can be constants, and rules/axioms. Figure 3 gives an Isabelle theory, named `Certif.thy` or, simply, `Certif`. This theory defines the data type of certificates, and includes several rules which express the projection functions in the Isabelle theory.

```
Certif = Main +
datatype
  ('n, 'd, 'k, 'e, 's) cert = "Cert" 'n 'd 'n 'k 'e 's
consts
   issuer        :: "('n, 'd, 'k, 'e, 's) cert => 'n"
   subject       :: "('n, 'd, 'k, 'e, 's) cert => 'n"
   start         :: "('n, 'd, 'k, 'e, 's) cert => 'd"
   expire        :: "('n, 'd, 'k, 'e, 's) cert => 'd"
   public        :: "('n, 'd, 'k, 'e, 's) cert => 'k"
   extensions    :: "('n, 'd, 'k, 'e, 's) cert => 'e"
   sig           :: "('n, 'd, 'k, 'e, 's) cert => 's"
   tbs    :: "('n, 'd, 'k, 'e, 's) cert => ('n * 'd * 'd * 'n * 'k * 'e)"
rules
   issuer_def    "issuer  (Cert I ds de S PuK exts SIG) = I"
   subject_def   "subject  (Cert I ds de S PuK exts SIG) = S"
   start_def     "start  (Cert I ds de S PuK exts SIG) = ds"
   expire_def    "expire  (Cert I ds de S PuK exts SIG) = de"
   public_def    "public  (Cert I ds de S PuK exts SIG) = PuK"
   extensions_def  "extensions  (Cert I ds de S PuK exts SIG) = exts"
   sig_def       "sig  (Cert I ds de S PuK exts SIG) = SIG"
   tbs_def       "tbs c = (issuer c,start c,expire c,subject c,
                                          public c,extensions c)"
end
```

Figure 3: The certificate theory

In this theory, "`Certif = Main +`" declares that `Main` is the parent theory of the theory `Certif`. `Main` is a basic theory in Isabelle/HOL, which collects all the basic predefined theories of arithmetic, lists, sets etc. Hence `Certif` is built upon the basis theory by defining a new data type (`Cert`), new syntax for functional constants (e.g. `issuer`), and rules that give the definitions and properties about these functions (e.g. `issuer_def`). The type `Cert` is built up from 7 tuples of values from the types 'n, 'd, 'k, 'e, and 's. These types are acually type variables – so, for example, the first 'n type (representing names) may later be instantiated to string or byte arrays. Isabelle requires explicit declarations for all functions (using keyword `consts`). All functions are annotated with concrete syntax, and defined by rules (listed under the keyword `rules`).

With the theory `Certif` defined, we can prove some goals. For example, the following goal can easily be proved by rewriting with the new rules above:

```
Goal "tbs (Cert I ds de S PuK exts SIG) = (I,ds,de,S,PuK,exts)";
```

Once a goal is proved, it can becomes a theorem of the `Certif` theory, and may be used in later proofs.

6.2 Topology Theory

The topology theory describes the certification topology of a PKI. When the agent axioms of the PKI have been obtained, translating them into Isabelle is sufficiently straightforward to be automated.

Figure 4 gives an Isabelle theory named `Topo`, which describes the certification topology of a hierachical PKI, which we have disscussed in Section 3.1.

```
Topo = Main +
consts
    Is_hrc              :: "('a * 'a)set => bool"
    PAA                 :: "('a * 'a)set => 'a"
    IsAgent             :: " ('a * 'a)set => ('a=> bool)"
    IsCA                :: "('a * 'a)set => ('a=> bool)"
    IsEE            :: "('a * 'a)set => ('a=> bool)"
rules
    PAA_def             "PAA r  = (@ a . IsCA r a & ~(? x . (x, a) :r))"
    IsCA_def            "(IsCA r a) = (? x . (a,x) :r)"
    IsEE_def            "(IsEE r a) = (~ (IsCA r a) & (? x . (x,a) :r))"
    IsAgent_def         "(IsAgent r a) = (IsCA r a | IsEE r a)"
    hrc_def             "Is_hrc r = (
                           (! x y .
                            (! a . (a ~= x & IsAgent r a) --> (x,a): r^+)&
                            (! a . (a ~= y & IsAgent r a) --> (y,a): r^+) -->
                                x = y) &
                             (! x y . ~ ( ((x,y): r^+) & ((y,x) : r^+))))"
end
```

Figure 4: The topology theory

The topology theory contains a rule, translated from axioms (A6) and (A7) in Section 3.1, which defines the function `Is_hrc` intended to assume that the PKI is hierarchical.

6.3 State Theory

Path validation is based on a given PKI state. This means that at any given state the possessed certificates and the revoked certificates for every agent are defined. The state should satisfy the state axioms, (S1) - (S4), given in Section 3.2, which can easily be translated into a Isabelle theory as shown in Figure 5.

```
State = Topo + Certif +
consts
    Is_state       :: " [('a * 'a)set,  ['a , ('a,'d,'k, 'e, 's) cert set ,
                              ('a,'d,'k,'e,'s) cert set ] => bool ] => bool "
rules
  state_def          "Is_state r sta = (Is_hrc r & ( ! a. ( IsAgent r a -->
                                  (? A B. (sta a A B)) &
                                  (! U V. ((sta a U V) --> (U=A) & (V= B))) &
                                  (!c. ( c : A --> (subject c = a) )) &
                                   (!c. ( c : B --> (issuer c = a) )) &
                                   (IsEE a --> (B = empty))
                                      ) ) )"
end
```

Figure 5: The state theory

6.4 Modelling Cryptographic Functions

In order to model the cryptographic functions that are involved in public key certificates, we adopt the following principles:

- *Higher-order, property-based approach.* That is, rather than define a primitive function for signing, and another for checking signatures, we define the types and relationships that such a pair of functions must satisfy. This has the advantage of making it easy to extend the theory (and results) for multiple algorithms. It is also logically more secure, since if there are axiomatically defined functions which are inconsistent, then the theory will be unsound, while if we make the corresponding error in the property relating signing and checking, then the logic remains sound, although the property becomes uninteresting as no functions can satisfy it.

- *Purely looking at the essential properties.* In the case of our cryptographic functions, we focus on the essential connection between signing and checking, and their relationship to the public and private keys. We do not model the signing mechanism itself, so our theory does not include any discussion of encryption, and hence does not reflect the idea of signature as encryption with a private key.

Based on the above principles, we construct the signature theory named as `Sign.thy` shown in Figure 2.

The signature theory includes two rules:

- The first rule (`sign_pair_def`) indicates that checking a message `M` against a signature `S` should succeed in precisely those cases when `S` equals the result of signing `M`.

```
Sign = Main +
consts
  is_sign_pair :: "(('m => 's) * ('m => 's => bool)) => bool"
  is_PK_sign_system :: "(('x => 'm => 's) * ('y => 'm => 's => bool) *
                                ('x*'y) => bool)) => bool"
rules
  sign_pair_def  "is_sign_pair (sign, check) =
                                  (!M S. check M S = (S = sign M))"
  PK_sign_system_def "is_PK_sign_system (sign, check, keyPair) =
              (! X Y. keyPair (X, Y) --> is_sign_pair (sign X, check Y))"
end
```

Figure 6: The signature theory

- The second rule (`PK_sign_pair_system_def`) means that the signing operation using private key `X` can be checked by the operation using public key `Y`, whenever `X` and `Y` represent a valid key pair.

Using the theory `Sign`, it is easy to prove the following theorem:

```
is_PK_sign_system (sign, check, keyPair) =
   (! X Y. ( keyPair (X,Y) --> (! M S. check Y M S = (S = sign X M))))
```

This theorem is the basis of checking if there are any changes occuring in the data part of a certificate. It indicates that in a public signature system, for any key pair (a public key and the corresponding private key), any message encrypted by the private key should be able to be decrypted by the public key, and similarly, if a message is encrypted by the public key, it should be able to be decrypted by the private key. All these are consistent with the certificate verification rules given in Section 4.

6.5 Framework Function, Path_Validation Theory

As already described, the various checks used for path validation can be grouped according to what data they are calculated on. We now need to describe in Isabelle a *framework function* that may take all these individual checks, and apply them to a certification path to be verified.

In the case of single and pair predicates, it is clear that these may be iteratively applied to the list of certificates. The results of these tests are joined by conjunction (since if any test fails the entire path must fail to validate). So the pattern of iteration can be described by:

$$Q_{i+1} = Q_i \cap (\mathtt{pair}(C_i, C_{i+1})) \cap (\mathtt{single}\, C_{i+1})$$

where Q_i is the accumulating boolean result (true meaning "valid so far"), and C_{i+1} is the current certificate we are processing (the framework function is defined so that the previous certificate processed is made available to the function `pair`).

The remaining functions can depend on any aspects of the list, and hence are the most general, and potentially worst behaved (from an ease-of-modelling viewpoint). However, in all the cases we have seen in path validation algorithms, there is a fairly simple sequence of iterative structures G_i that can be constructed down the certification path which allow these predicates to be calculated.

We extend our accumulating boolean result to cover also these last predicates, represented by `path_pred` in the following:

$$\begin{aligned} Q_{i+1} &= Q_i \cap (\texttt{path\_pred}(G_i, C_{i+1})) \cap (\texttt{pair}(C_i, C_{i+1})) \cap (\texttt{single}C_{i+1}) \\ G_{i+1} &= f(G_i, C_{i+1}) \end{aligned}$$

We introduce the framework function to implement these iterative calculations, using the various predicate arguments passed to it. It is called as follows:

```
framework path_state path_pred pair single state_init trusted_cert certs
```

where the meanings of arguments are as follows:

- `path_state` – the function that calculates the new state from the old state and the new cert (certificate). Needed for path properties that use the whole path.
- `path_pred` – The path property calculating whether the validation of the whole path (up to here) has failed yet.
- `pair` – pred for checking neighbours in the certificate path.
- `single`: pred for checking single certs (e.g. for time).
- `state_init` – the initial values in the state used for whole path pred. Often either a "null" of some type, or derived from the initial trusted certificate.
- `trusted_cert` – the "top cert" that is trusted by the verifier.
- `certs` – an argument to the resulting validation function - the list of certs to be verified. Note that the head of `certs` is the highest in the tree.

Thus, the path_validation theory can take `State` and `Sign` as the parent theories to peform the path validation process. Having the parent theories `State` and `Sign`, the verfier is allowed to do single checks (check the time) and pair checks (identities of subject and issuer, signature, and CRL list), and to determine if a certificate is trusted. The main rules included in this theory are shown in Figure 7.

The rule `single_def` defines a single check that verifies the validity dates of the certificate (checking the validity of the time `t` against the start date and expiry date of the

```
single1_def   "single1 t c = (((start c) <= t) & (t < (expire c)))"
pair1_def     "pair1 (c1, c2) = (subject c1 = issuer c2)"
pair2_def     "pair2 check (c1, c2) = ((pair1 (c1, c2))
                & (check (public c1) (tbs c2) (sig c2)))"
pair3_def     "pair3  r sta check (c1,c2) = (
                    Is_CMS_state r sta & (pair2 check (c1, c2)) &
                   (sta (subject c1) A B --> ~(c2: B) )) "
path_validation1_def "path_validation1 T  =
          framework path_state1 path_pred1 pair1 (single1 T) ()"
path_validation2_def "path_validation2 check T  =
       (framework path_state1 path_pred1 (pair2 check) (single1 T) ())"
path_validation3_def "path_validation3 r sta check T  =
  (framework path_state1 path_pred1 (pair3 r sta check) (single1 T) ())"
```

Figure 7: The rules of path_validation theory

certificate c). The rules pair1_def, pair2_def and pair3_def are all pair checks. Concretely, the rule pair1_def is used to check the identities with the pair of certificates (c1,c2); pair2_def is used to check not only the identities with the pair of certificates (c1,c2), but also the signature on c2 to look if there is any change on this certificate; for the rule pair3_def, apart from all these checkes which pair2_def does, it also checks if the cerificate c2 is revoked by the subject of c1. Correspondingly, there are three rules representing different levels of the path validation function. Usually, for most security applications path_validation3_def would be required.

6.6 Remarks

Having the essential Isabelle theories, we may build a system applied for performing verification tasks. In particular, after we put all the facts describing a real PKI into the theory Topo and nominate the signature algorithms in the theory Sign, the system could be applied for verifying real certification paths of the PKI. Figure 8 illustrates the architecture of a PKI certificate verification system.

Once a verification system for modelling the path validation is defined, it becomes possible to define paths of certificates, and to reason about certificate validity. In this way we can exhibit particular paths that illustrate unexpected behaviour of a given path validation algorithm. It will also be possible to prove that a particular algorithm may have certain desirable security properties.

7 Modelling PKIX Path Validation

So far, we did not discuss the extension fields involving certificate policies. In this section, we present an approach to modelling a specific path validation algorithm – the

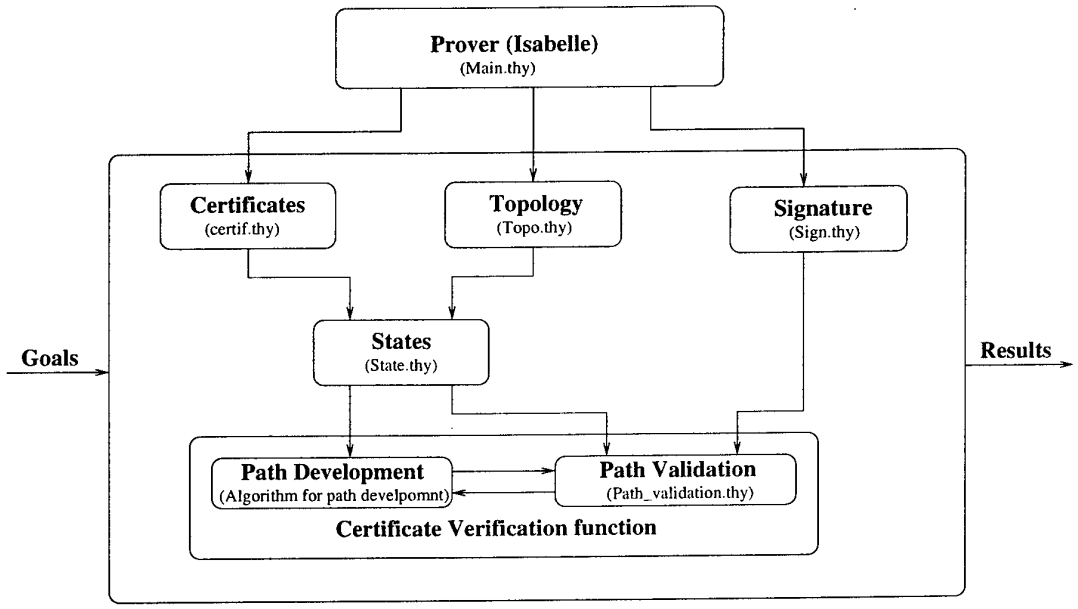


Figure 8: A model of the PKI verification system

PKIX path validation algorithm, which involves constructing a "valid policy tree" to determine if the set of certificate policies are valid for a given path.

PKIX path validation is quite complex, so we do not aim yet to model all aspects. Separate consideration of the processing regarding various extensions should be a valid technique. The reason is that most processing of extensions is independent of each other. It is, however, useful to look at interactions when they occur, and this would be a good area for further study.

The aspects we will be focusing on are the PKIX processing rules for dealing with "certificate polices". These are defined as:

> *A named set of rules that indicates the applicability of a certificate to a particular community and/or class of application with a common security requirement* [22].

Informally we interpret the role of certificate policy processing in PKIX as determining for the users the set of policies under which a given certificate may be validly used. A given certificate may have been issued under policies P_1 and P_2, but it may be that it is only valid for use under policy P_2 (perhaps because of restrictions applied by CAs higher in the path, or by the application carrying out the validation).

Closely related to certificate policies is the *policy mappings* extension, which is intended to associate related policies from different domains when cross-certifying between them. Although they play an important role in the PKIX algorithm, and have a major part in the discovered flaws, we do not yet include them in the modelling presented in this paper.

Consider now the predicates for checking certificate policies in PKIX. Our accumulating structures G_i need to track various information: the set of policies acceptable to the

user(user policy set: ups), the set of policy constraints from CAs earlier in the certification path (authority policy set: aps), a counter for the number of certificate we are up to in processing (cnt), and the certificate number at which certificate policies must be present (policy *limit*: lim). These are grouped together in a tuple: (ups, aps, lim, cnt). In the following, we consider the path predicate for path predicate checking the policies of the next certificate (c) using the data of this tuple.

Given the current values for (ups, aps, lim, cnt), we use a particular path predicate named as path_pred_cp (cp for certificate policy) to process the next certificate, say c. The path predicate is defined in the following form:

$$\texttt{path\_pred\_cp\_def} \quad \text{"}\texttt{path\_pred\_cp}((\texttt{ups}, \texttt{aps}, \texttt{lim}, \texttt{cnt}), \texttt{c}) = \ldots\text{"}$$

where the detail of the right hand side of the equation is not given yet.

We now consider how the right hand side in the above definition is to be defined. From PKIX we have the following process in the path validation algorithm:

> "(e) (1) *if the certificate policies extension is marked critical, the intersection of the policies extension and the acceptable policy set shall be non-null.*"

This can be expressed in the path predicate provided to the framework as follows:

$$((\texttt{certPoliciesCritical}(\texttt{extensions c}) \rightarrow (\texttt{aps} \cap (\texttt{certPoliciesExt}(\texttt{extensions c})) \neq \emptyset)))$$

where $\emptyset$ is the empty set.

The following check is also done by the PKIX algorithm:

> "(d) (1) *if the explicit policy state variable is less than or equal to i, a policy identifier in the certificate shall be in the initial policy set.*"

The algorithm has the following representation in Isabelle:

$$((\texttt{lim} <= \texttt{cnt}) \rightarrow (\texttt{ups} \cap (\texttt{certPoliciesExt}(\texttt{extensions c})) \neq \emptyset))$$

The two preceding predicates are conjoined to complete the definition of path_pred_cp and are used in the framework. Also defined and used are functions that define how the various components lim, cnt and aps are updated, and these functions can again be directly linked to clauses in the PKIX definition.

Once a theory describing path validation involving certificate policies (in our case for a subset of PKIX path validation) is provided, we will be able to reason about certificate validity with considering policy extension fields and, possibly, to prove that a particular algorithm may satisfy certain desirable security properties under a given security policy, though we have not yet carried out such proofs.

8 Discussion and Concluding Remarks

We have seen how an absence of formal specification in a more intricate part of a technical standard has caused problems with implementations, and resulted in a general interpretation apparently diverging from that intended by its authors (hence leading to the need for revisions of these parts). By adding formality in the specification process, we suggest that some of these problems may be avoided.

Our approach can be used to detect or highlight inconsistencies and errors in standards and/or implementations. One of the ways that this is achieved is by the type-checking Isabelle carries out on the theory, which can reveal basic kinds of inconsistencies.

Formal specification may also be used to compare various alternatives to fixing a particular problem, and potentially to verify that some alternatives are indeed free of error. For example, we are now looking at the suggested changes to PKIX, to see whether they are indeed free of some of the undesirable properties.

One of the major issues is that many standards, such as PKIX and X.509, are effectively specifying a *solution* (or a class of solutions). Although we have seen that the relative informality of these specifications may cause problems, the issue becomes even more serious when considering the *problems* that these solutions are meant to address. Usually the problems that various features of the solution are meant to address are not described, or only very informally. Effectively many features have no clear requirement, and the advice to someone developing a PKI reduces to "well, this is the effect extension X has on path validation, so if that is what you need, use it".

An example is certificate policies, where their intended meaning in end-entity certificates is perhaps clear (though what happens if a certificate has multiple policies, and their policy meanings are inconsistent?), but in CA certificates their effect is so complicated that no simple requirement is ever given for them. Perhaps another example is policy qualifiers; these allow general information to annotate certificate policies, but their uses to date have been very restricted (e.g. pointers to where the policy definition may be obtained), and it is completely unclear what applications should do with them, and what would be acceptable ways for CAs to populate them (thankfully the path validation decision is not to be dependent upon policy qualifiers; the new PKIX algorithm does specify how to process qualifiers for "any-policy", and hence be generating a "solution" based on semantics for qualifiers, which may ultimately be inappropriate).

Even when standards are correct, we can often provide an accurate and more abstract re-expression of the standard. This can be particularly useful for implementations that seek to use algorithms or data structures which differ from those used in the standards (although, of course, aiming to be behaviourly equivalent).

There exists the potential to prove certain properties about a PKI on the basis of the standard it implements. This could serve as a valuable input to the development of any future "higher-grade" PKI developments.

In the specific case of path validation, we have explained how a simple model (our framework) can be used to bring order to the many checks required in path validation, and provide a basis for formalization. We have, furthermore, explained some aspects of

just such a formalization, linking it back to statements within the standard. Thus the feasibility of such an approach to specification in this domain area is demonstrated.

Our model is a state-based model, in which the PKI state may change from time to time. Therefore, in order to effectively describe dynamic changes of the PKI states, temporal logics would be helpful. Also, as we said before, for modelling the trust relation in a PKI, we may need to investigate a specific belief logic with some modal operators applied for specifying and reasoning about trust. Therefore, a further theoretical study is needed to provide a solid foundation for specifying and reasoning about security properties of a PKI.

In future work we would also like to extend further into the areas of PKIX path validation not yet modelled, and also look at modelling and comparing some alternatives for resolving the current problems with this validation algorithm.

Our approach to the formalization of a signature theory is quite general. Note that we allow the types of the private and public keys to be different, in contrast to much work that has been done in the authentication protocol verification area, such as Paulson [44], Burrows *et. al.* [6], Roscoe [50] and Lowe [36] etc. However, in order to accommodate DSA [40], we would need to extend our model to allow non-deterministic parameters. We could also extend the work further to cover more of the total certificate lifecycle (e.g. registration authority functions, rekeying, and revocation). Closely related to these aspects are the underlying trust models; clearly this would relate to work by Maurer [37], Reiter and Stubblebine [48, 49], and others.

Acknowledgements

We would like to thank Dr. Brendan Mahony and Dr. Jim McCarthy for helpful discussions on Isabelle theories. Thanks are also due to Peter Drewer, the Head of Trusted Computer Systems Group, who carefully read all our working papers on this subject and gave useful comments. Thanks also to Peter Gutmann for valuable information about the "Cryptlib" implementation and useful comments.

References

1. M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proceedings of the 1993 IEEE Computer Society Symposium on research in Security and Privacy*, pages 164–173, 1996.

2. C. Boyd. Security architecture using formal methods. *IEEE Journal on Selected Areas in Communication*, 11(5):694–701, 1993.

3. S. Bradner. The internet standard process – revision 3 (PRC 2026). Available from `http://www.ietf.org/rfc/rfc2026.txt`, October 1996.

4. W. E. Burr. Public Key Infrastructure (PKI) Technical Specifications: Part A – Technical Concept of Operations. Available from `http://csrc.nist.gov/pki`, September 1998.

5. W. E. Burr, D. Dodson, N. Nazario, and W. T. Polk. MISPC: Minimum Interoperability Specification for PKI Components, Version 1. Available from `http://csrc.nist.gov/pki`, September 1997.

6. M. Burrows, M. Abadi, and R. M. Needham. A logic of authentication. In *Proceedings of the Royal Society of London* **426**, pages 233–271, 1989.

7. A. Cant, K. A. Eastaughffe, and M. A. Ozols. A tool for practical reasoning about state machine designs. In *Proceedings of 1996 Australian Software Engineering Conference*, pages 16–26. IEEE Computer Society Press, 1996.

8. Cryptlib. [Online]<http://www.cs.auckland.ac.nz/~pgut001/cryptlib>, Technical Contact: Peter Gutmann.

9. A. Dekker. C3PO: A tool for automatic sound cryptographic protocol analysis. In *Proceedings of the 13th Computer Security Foundations Workshop*, pages 77–87, Cambridge, UK, 3-5 July 1996. IEEE Computer Society.

10. DR 222. *Defect Report Number DR 222: Certificate Policy Mapping.* Available from ftp://ftp.bull.com/pub/OSIdirectory/DefectResolution/DefectReports/X.5 09/, June 1999.

11. 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.

12. C. Ellison. Establishing identity without certification authorities. In *Proceedings of Sixth Annual USENIX Security Symposium: Focusing on Applications of Cryptography*, pages 67–76, July 22 -25, 1996.

13. W. Ford and M. Baum. *Secure Electronic Commerce: Building the Infrastructure for Digital Signatures and Encryption.* Prentice-Hall, 1997.

14. B. Fox and B. LaMaccia. Certificate revocation: Mechnismss and meaning. In R. Hirschfeld, editor, *Financial Cryptography*, volume 1465 of *LNCS*, pages 158–164. Springer-Verlag, Berlin, 1998.

15. J. Glasgow, G. MacEwen, and P. Panagaden. A logic for reasoning about security. *ACM Transaction on Computer System*, 10(3):226–264, 1992.

16. M. J. C. Gordon and T. F. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic.* Cambridge University Press, 1993.

17. MCG Group. MCG – Internet Open Group on Certificate and Security. Available from http://mcg.org.br, 1998.

18. S. Gupta. A common key recovery block format: Promoting interoperability between dissimilar key recovery mechanisms. *Computers & Security (UK)*, 19(1):41–47, 2000.

19. S. Gupta and S. M. Matyas. Public key infrastructure: Analysis of existing and needed protocols and object formats for key recovery. *Computers & Security (UK)*, 19(1):56–68, 2000.

20. P. Gutmann. X.509 style guide. [Online]<http://www.cs.auckland.ac.nz/~pgut001/>, 2000.

21. A. Herzberg, Y. Mass, and J. Mihaeli. Access control meets public key infrastructure, or: Assigning roles to strangers. In *Proceedings of the 2000 IEEE Computer Society Symposium on research in Security and Privacy*, pages 2–14, 2000.

22. R. Housley, W. Ford, W. Polk, and D. Solo. RCF 2459, *Internet X.509 Public Key Infrastructure - Part I: Certificate and CRL Profile.* Internet Request for Comments 2459, January 1999.

23. 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). Available from http://www.imc.org/draft-ietf-pkix-new-part1, January 1999.

24. A. Jøsang, I. G. Pedersen, and D. Povey. PKI seeks a trusting relationship. In *Proceedings of the 5th Australasian Conference on Information Security and Privacy (ACISP 2000)*, volume 1841 of *Lecture Notes in Computer Science*, pages 191–205. Springer, 2000.

25. A. Jøsang and S.J. Knapskog. A metric for trusted systems. In *Proceedings of the 21st National Security Conference*, NSA 1998.

26. N. Kapidzic. Extended certificate management system: Design and protocols. Technical report, DVS, 1997.

27. N. Kapidzic. Creating security applications based on the global certificate management system. *Computers & Security*, 17:507–515, 1998.

28. S. Kent. Privacy Enhancement for Internet Electronic Mail, Part II: Certificate-Based Key Management, Request for Comments 1422. Network Working Group, 1993.

29. R. Kohlas and U. Maurer. Reasoning about public-key certification: On bindings between entities and public keys. In *Proceedings of Financial Cryptography 99 (FC99)*, LNCS. Springer-Verlag, Berlin, 1999.

30. R. Kohlas and U. Maurer. Confidence valuation in a public-key infrastrucutre based on uncertain evidence. In *Proceedings of the 3rd International Workshop on Practice and Theory in Public Key Cryptosystems (PKC 2000)*, volume 1751 of *Lecture Notes in Computer Science*, pages 93–113. Springer, 2000.

31. I. Lehti and P. Nikander. Certifying trust. In H. Imai and Y. Theng, editors, *Proceedings of the First International Workshop on Practice and Theory in Public Key Cryptosystems (PKC'98)*, pages 83–98, 1998.

32. A. Levi and M. U. Caglayan. An efficient, dynamic and trust preserving public key infrastructure. In *Proceedings of the 2000 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 203–214. IEEE Computer Society, 2000.

33. N. Li, B. Grosof, and J. Feigenbaum. A practically implementable and tractable delegation logic. In *Proceedings of the 2000 IEEE Computer Society Symposium on research in Security and Privacy*, pages 27–42, 2000.

34. C. Liu, M. A. Ozols, M. Henderson, and T. Cant. A state-based model for ceritificate management systems. In *Proceedings of the 3rd International Workshop on Practice and Theory in Public Key Cryptosystems (PKC 2000)*, volume 1751 of *Lecture Notes in Computer Science*, pages 75–92. Springer, 2000.

35. C. Liu, M. A. Ozols, M. Henderson, and T. Cant. Towards certificate verification in a ceritificate management system. In Jenny Edwards, editor, *Proceedings of the 23rd Australasian Computer Science Conference, ACSC2000*, pages 150–157, Canberra, Australia, 31 January – 3 February 2000. IEEE Computer Society.

36. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocols using CSP and FDR. In T. Margaria and B. Steffen, editors, *Tool and Algorithms for the Construction and Analysis of Systems. Second International Workshop, TACAS'96, LNCS 1055*, pages 147–166. Springer, 1996.

37. Ueli Maurer. Modeling a public-key infrastructure. In E. Bertino, H. Knurth, G. Martella, and E. Montolivo, editors, *Computer Security - ESORICS'96 (LNCS 1146)*. Springer-Verlag, 1996.

38. J. Mulvenna, L. Keys, D. Walters, S. Ganta, and S. Gupta. Characteristics and attributes that affect S/MIME product interoperability. Draft available from `http://csrc.nist.gov/pki/smime/welcome.htm`.

39. M. Naor and K. Nissim. Certificate revocation and certificate update. In *Proceedings of Usenix'98*, pages 217–228, January 1998.

40. NIST. *Digital Signature Standard* (DSS). Federal Information Processing Standards Publication 186, November 1994.

41. M. A. Ozols, A. Cant, and K. A. Eastaughffe. XIsabelle: A system description. In *Proceedings of the 14th International Conference on Automated Deduction (CADE-14)*, Lecture Notes in Artificial Intellegince, pages 400–403. Springer-Verlag, 1997.

42. M. A. Ozols, M. Henderson, C. Liu, and T. Cant. The PKI specification dilemma: A formal solution. In *Proceedings of the 5th Australasian Conference on Information Security and Privacy (ACISP 2000)*, volume 1841 of *Lecture Notes in Computer Science*, pages 206–219. Springer, 2000.

43. L. C. Paulson. *ML for Working Programmer.* 2nd edition, Cambridge University Press, 1996.

44. L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1-2):85–128, 1998.

45. Lawrence C. Paulson. The Isabelle Reference Manual. University of Cambridge, Computer Laboratory, available from `http://www.in.tum.de/ isabelle/dist`.

46. P. V. Rangan. An axiomatic basis of trust in distributed systems. In *Proceedings of the 1988 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 204–211, 1988.

47. T. Rea. High value certification – trust services for complex eCommerce transactions. *BT Technology Jounal (Netherland)*, 17(3):50–56, 1999.

48. M. K. Reiter and S. G. Stubblebine. Path independence for authentication inlarge-scale systems. In *Proceedings of the 4th ACM Conference on Computer and Communication Security*, pages 57–66, April 1997.

49. M. K. Reiter and S. G. Stubblebine. Toward acceptable metrics of authentication. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 1997.

50. A. W. Roscoe. Modelling and verifying key-exchange protocols using csp and fdr. In *8th Computer Security Foundation Workshop*, pages 98–107. IEEE Computer Society Press, 1995.

51. B. Schneier. *Applied Cryptography.* 2nd edition, John Wiley & Sons, Inc., 1996.

52. S. Stubblebine and R. Wright. An authentication logic supporting synchronization, revocation, and recency. In *SIGSAC: 3rd ACM Conference on Computer and Communication Security.* ACM SIGSAC, 1996.

53. P. Syverson and C. Meadows. A logic language for specifying cryptographic protocol requirements. In *Proceedings of the 1993 IEEE Computer Society Symposium on research in Security and Privacy*, pages 165–177, 1993.

54. P. F. Syverson and P. C. van Oorschot. On unifying some cryptographic protocol logics. In *Proceedings of the IEEE Society Symposium on Research in Security and Privacy*, pages 234–248, Oakland, CA USA, 1994. IEEE Computer Society Press.

55. Denis Trcek. Organization of certification authorities in a global network. *Computer Security Journal*, 10(1):72–81, 1994.

56. T. Y. C. Woo and S. S. Lam. Authorization in distributed systems: A new approach. *Journal of Computer Security*, pages 107–136, 2(1993).

57. X.509. *Information Technology - Open systems Interconnection - The Directory: Authentication Framework.* ITU-T Recommendation X.509, June 1997.

58. R. Yahalom, B. Klein, and Th. Beth. Trust relationships in security systems - A distributed authentication prespective. In *Proceedings of the 1993 IEEE Computer Society Symposium on research in Security and Privacy*, pages 151–164, 1993.

DISTRIBUTION LIST

Formalization of Public Key Infrastructures

Maris Ozols, Tony Cant, Chuchang Liu and
Marie Henderson

	Number of Copies
DEFENCE ORGANISATION	
Task Sponsor	
DGISREW	1
S&T Program	
Chief Defence Scientist FAS Science Policy AS Science Corporate Management	1
Director General Science Policy Development	1
Counsellor, Defence Science, London	Doc Control Sheet
Counsellor, Defence Science, Washington	Doc Control Sheet
Scientific Adviser to MRDC, Thailand	Doc Control Sheet
Scientific Adviser Policy and Command	1
Navy Scientific Adviser	Doc Control Sheet
Scientific Adviser, Army	Doc Control Sheet
Air Force Scientific Adviser	1
Director Trials	1
Aeronautical and Maritime Research Laboratory	
Director, Aeronautical and Maritime Research Laboratory	1
Electronics and Surveillance Research Laboratory	
Director	(1 copy of Doc Control Sheet and 1 Distribution list)
Chief, Information Technology Division	1
RLMIE, Information Technology Division	1
Research Leader Advanced Computer Capabilities	Doc Control Sheet
Research Leader Joint Systems	1
Head, Information Warfare Studies Group	Doc Control Sheet
Head, Enterprise Visualisation, Instrumentation and Synchronisation Group	Doc Control Sheet
Head, Trusted Computer Systems Group	1
Head, Systems Simulation and Assessment Group	Doc Control Sheet
Head, C3I Operational Analysis Group	Doc Control Sheet
Head, Information Exploitation Group	Doc Control Sheet
Head, Intelligence Group	Doc Control Sheet
Head, Human Systems IntegrationGroup	Doc Control Sheet
Head, C2 Australian Theatre Group	1
Head, Distributed Systems Group	Doc Control Sheet
Head, C3 Concepts Group	1

Head, Military Systems Synthesis Group	Doc Control Sheet
Head, Systems of Systems Group	Doc Control Sheet
Head, Advanced Network Integrity Group	Doc Control Sheet
Maris Ozols, Task Manager	1
Authors	4
Publications & Publicity Officer, ITD / Executive Officer, ITD	1
DSTO Library and Archives	
Library Fishermens Bend	1
Library Maribyrnong	Doc Control Sheet
Library Salisbury	1
Australian Archives	1
Library, MOD, Pyrmont	Doc Control Sheet
US Defence Technical Information Center	2
UK Defence Research Information Centre	2
Canada Defence Scientific Information Service	1
NZ Defence Information Centre	1
National Library of Australia	2
Capability Development Division	
Director General Maritime Development	1
Director General Land Development	1
Director General Aerospace Development	Doc Control Sheet
Knowledge Staff	
Director General Command, Control, Communication and Computers (DGC4)	Doc Control Sheet
Director General Defence Knowledge Improvement Team (DGDKNIT) R1-5-A165, Canberra ACT 2600	Doc Control Sheet
Navy	
SO(Science), Catherine Morgan, COMAUSNAVSURFGRP, Bldg 95, Garden Island, Locked Bag 12, Pyrmont NSW 2009	Doc Control Sheet
Army	
Stuart Schnaars, ABCA Standardisation Officer, Tobruck Barracks, Puchapunyal VIC 3662	4
SO(Science), Deployable Joint Force Headquarters (DJFHQ) (L), MILPO, Gallipoli Barracks, Enoggera, QLD 4052	Doc Control Sheet
NPOC QWG Engineer NBCD Combat Development Wing, Tobruk Barracks, Puckapunyal, 3662	Doc Control Sheet
Intelligence Program	
DGSTA Defence Intelligence Organisation	1
Manager, Information Centre, Defence Intelligence Organisation	1
Corporate Support Program (libraries)	
Library Manager, DLS-Canberra	1

DSD	
ASINFOSEC	1
DSB	
DSEC-TECH	1
UNIVERSITIES AND COLLEGES	
Australian Defence Force Academy Library	1
Head of Aerospace and Mechanical Engineering, ADFA	1
Deakin University Library, Serials Section (M List)	1
Senior Librarian, Hargrave Library, Monash University	Doc Control Sheet
Librarian, Flinders University	1
OTHER ORGANISATIONS	
NASA (Canberra)	1
AusInfo	1
The State Library of South Australia	1
Parliamentary Library of South Australia	1
ABSTRACTING AND INFORMATION ORGANISATIONS	
Library, Chemical Abstracts Reference Service	1
Engineering Societies Library, US	1
Materials Information, Cambridge Science Abstracts, US	1
Documents Librarian, The Center for Research Libraries, US	1
INFORMATION EXCHANGE AGREEMENT PARTNERS	
Acquisitions Unit, Science Reference and Information Service, UK	1
Library – Exchange Desk, National Institute of Standards and Technology, US	1
SPARES	
DSTO Salisbury Research Library	5
Total number of copies:	60

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION DOCUMENT CONTROL DATA				1. CAVEAT/PRIVACY MARKING
2. TITLE Formalization of Public Key Infrastructures			3. SECURITY CLASSIFICATION Document (U) Title (U) Abstract (U)	
4. AUTHORS Maris Ozols, Tony Cant, Chuchang Liu and Marie Henderson			5. CORPORATE AUTHOR Electronics and Surveillance Research Laboratory PO Box 1500 Salisbury, South Australia, Australia 5108	
6a. DSTO NUMBER DSTO–RR–0202	6b. AR NUMBER AR–011–780		6c. TYPE OF REPORT Research Report	7. DOCUMENT DATE April, 2001
8. FILE NUMBER N9505/21/45	9. TASK NUMBER JNT 00/131	10. SPONSOR DGISREW	11. No OF PAGES 35	12. No OF REFS 58
13. URL OF ELECTRONIC VERSION http://www.dsto.defence.gov.au/corporate/reports/DSTO–RR–0202.pdf			14. RELEASE AUTHORITY Chief, Information Technology Division	

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved For Public Release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE, PO BOX 1500, SALISBURY, SOUTH AUSTRALIA 5108

16. DELIBERATE ANNOUNCEMENT

No Limitations

17. CITATION IN OTHER DOCUMENTS

No Limitations

18. DEFTEST DESCRIPTORS

Secure communications
Computer information security
Path validation
Computer security
Public key infrastructure

19. ABSTRACT

Public key technology within a Public Key Infrastructure (PKI) has been widely promoted to support secure digital communications. However, imprecise specifications for PKIs, which are usually written in a natural language, have led to varying implementation and interpretations of conformance. There have also been cases where defects have been identified years later, some of which were serious and could cause incorrect acceptance of certification paths. In this paper we provide a formal solution to the PKI specification dilemma by introducing a state-based model for the description of the architecture of a PKI and related functions. We propose a formal approach to the representation of, and reasoning about, the behavior and security properties of PKIs, and also give a framework for mechanizing our theory in the Isabelle theorem prover. With our method, the essential aspects of PKIs can be clearly formulated, facilitating the testing and analysis of their implementations in a more rigorous and well defined way.