Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 235 (2009) 55–70
www.elsevier.com/locate/entcs

Formal Modelling of PKI Based Authentication

Ali Nasrat Haidar[1] Ali E. Abdallah[2]

E-Security Research Centre
Institute for Computing Research
London South Bank University
103 Borough Road
London, UK

Abstract

One of the main aims of certificate based Public Key Infrastructure (PKI) is to provide authentication in distributed systems. Through its functions, PKI authentication can be viewed as a re-usable component that can be integrated with other systems to offer strong authentication, scalability, and mobility, particularly for large organizations. PKI has been used to describe authentication in various types of applications ranging from e-commerce and web services applications to large scale systems such as Grid computing. This paper presents a formal approach for modeling certificate based PKI authentication. The approach makes use of two complementary models: one is state-based, described in Z, and the other is event-based, expressed in the Process Algebra of Hoare's Communicating Sequential Processes (CSP). The former will be used to capture the state of PKI key components used in the authentication process, the relationships between them, and model "back-end" operations on these components. Whereas the latter, CSP, will be used to model behavior, and in particular, "front-end" interactions and communications. Only when this authentication mechanism is properly formulated, reasoning about its correctness, vulnerabilities and usability can be scrutinized and possibly aided by automation.

Keywords: Formal Methods, *Z*, CSP, Security, Authentication, Distributed systems, Correctness.

1 Introduction

Certificate based public Key Infrastructures (PKI) [2] have been the source of many of the radical advances in the evolution of security solutions to: authentication, authorization, confidentiality, integrity, and accountability. PKI has been used in a wide variety of distributed applications ranging from e-commerce and web services applications to complex systems such as Grid computing and virtual organizations [25]. Also, PKI has been used in the design of security protocols such as Secure Socket Layer (SSL/TLS) [24] and Secure Electronic Transaction (SET) [4] with the

[1] Email: Ali.Haidar@lsbu.ac.uk
[2] Email: A.Abdallah@lsbu.ac.uk

1571-0661/© 2009 Elsevier B.V. Open access under CC BY-NC-ND license.
doi:10.1016/j.entcs.2009.03.005

main aim is to provide authentication. This refers to the ability to demonstrate the identity of an entity (human user, server, or a service) to any interested party [5].

Despite its widespread adoption, however, certificate based PKI still "suffers from certain ambiguity and lack of understanding and precision" [11]. The ability to have a clear and rigorous understanding of PKI-authentication requirements is particularly significant when building secure, reliable, and reusable PKI authentication components. However, poor implementations have been the main factor that has badly influenced the use of PKI based authentication on a large scale [18]. This is due to several reasons according to [1]: (1) PKI can be viewed as a complex distributed information system in which there is a potential risk that design errors and undesirable properties emerge causing considerable costs for failures to meet the intended requirements; (2) difficulty of integration into existing applications; (3) and lack of clear and rigorous approaches that enable reasoning about correctness of PKI systems' administrative side and their security and reliability. Successful PKI implementations have been restricted to "closed" environments where a conservative security policy can be applied in order to produce an effective implementation [19].

The primary aim of this paper is to present a formal model for certificate based PKI systems by combining and customizing existing formal frameworks for state-based and event-based systems. The model is formulated in specification notation **Z** [26,8,22], which is particularly suited for concisely describing state-based systems and reasoning about them. The specification consistency is checked using *ZTC* tool [10]. Then, the behavior of the PKI system components is captured using Hoare's Communicating Sequential Processes (*CSP*) [6]. When a certificate based PKI system is properly formulated, reasoning about its correctness, vulnerabilities and usability can be scrutinized and possibly aided by automation. This approach can be refined to deal with a wide range of components such as authorization and auditing, and would be useful in helping developers to have a clear and rigorous understanding of components during the design, analysis and implementation phases.

In this paper, we do not attempt to consider all the functions provided in a typical PKI system such as key generation and revocation-list management. We are interested in authentication related operations, security knowledge of the participants (user, server), security tokens, and administrative operations such as adding/removing users and trusting certificate authorities. We assume that: this PKI system will work in a closed environment, where all users can be identified and every user holds one certificate and its corresponding private key; also, PKI entities (i.e. client, server) do not need to negotiate which cryptographic algorithms they will all use.

The paper begins with an overview of certificate based PKI. In Section 3, a formal model for a PKI authentication system is constructed by stating its key components and describing the mathematical relationships between them using **Z** notation. Section 4 presents the process architecture for PKI authentication expressed in *CSP* notation. Section 5 describes typical administrative operations performed by a system administrator on the authentication server, which are for-

mally described in **Z**. Section 6 gives an overview of related work. Section 7 is the conclusion.

2 Overview of PKI

PKI is based on asymmetric cryptography [21] concept in which each user has a related pair of keys: a *public* key and a corresponding *private* key. When such a key pair is generated, the public key is intended to be made public, whereas the private key should only be known and protected by the user. One of the most currently used public key cryptographic algorithm is RSA [15] since it is suitable for both encryption and digital signatures.

Public Key encryption is used to maintain the privacy of data communicated over a public network (Fig. 1). When a message is transmitted using public key encryption, it is the public key of the recipient that is used to encrypt. For example, consider a user A with a public key pk and private key sk respectively that can be used with RSA algorithm. Another user, say B, wishing to send a message m to A, obtains A's public key, uses RSA to obtain the encryption $c = RSA\, pk\, m$, and transmits c to A. To decrypt c, A applies RSA to obtain the original message $RSA\ sk\ (RSA\, pk\ m) = m$. Public Key encryption also provide digital signature to ensure data integrity. When the private key is used for encrypting a message (known as signing) then any recipient who can obtain the corresponding public key can decrypt it. The relationship between the public key and the private key using RSA algorithm can be summarized by the following expressions:

$$RSA\, pk\, (RSA\, sk\, m) = m \qquad (1)$$
$$RSA\, sk\, (RSA\, pk\, m) = m \qquad (2)$$

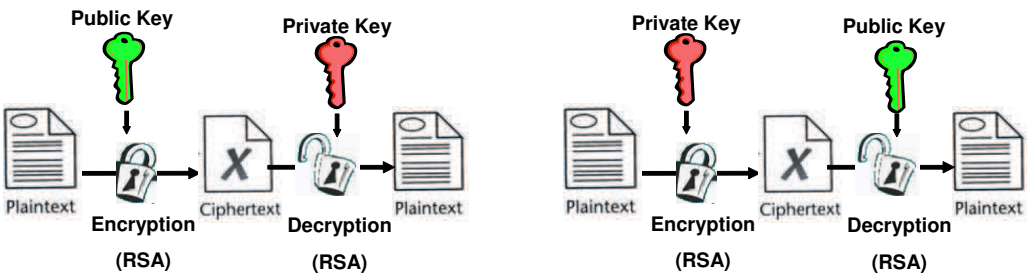

Fig. 1. Public Key Encryption and Data Signing

In order to ensure the authenticity of public keys, digital certificates are used to enable the binding of a public key to identifying information about a subject on the certificate (i.e people, servers, organizations) in such a way that its integrity and validity can be verified [2]. Digital certificates can be viewed as a security token in which public keys may be stored, distributed or forwarded over a public network such as the Internet. In PKI, the binding between subject and key is established

by a trusted third party called Certification Authority (CA) (also known as Issuer). The primary function of the CA is to generate, publish, revoke, and archive the public key certificates that binds the user identity with the user's public key. When a CA issues a certificate to a user, it signs it with its private key to ensure that any modification on the certificate can be detected. The CA also issues a certificate for itself (called root certificate). Anyone who wants to use a certificate must have a valid copy of the public key of the CA who issued the certificate, and must trust the CA (by having a copy of the root certificate). One can also choose where the key pair is generated. The keys can either be generated by the CA for the client or the client can generate the keys for itself and provide a copy of the public key to the CA to certify (in this work we assume that the client already has a key pair certified by a trusted CA).

The main characteristics of a typical certificate consists of: names of the subject and issuer, a public key associated with the subject, a validity period, an identifier for the cryptographic algorithms used by the CA to sign this certificate and another identifier for the public key algorithm with which the public key on the certificate is used (i.e RSA, Diffie-Hellmann). X509 version 3 (Fig. 2) is the most currently used PKI standard for digital certificates [7].

```
Certificate:
   Data:
       Version: 3 (0x2)
       Serial Number: 1 (0x1)
       Signature Algorithm: md5WithRSAEncryption
       Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
               OU=Certification Services Division,
               CN=Thawte Server CA/Email=server-certs@thawte.com
       Validity
           Not Before: Aug  1 00:00:00 1996 GMT
           Not After : Dec 31 23:59:59 2020 GMT
       Subject: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
                OU=Certification Services Division,
                CN=Thawte Server CA/Email=server-certs@thawte.com
       Subject Public Key Info:
           Public Key Algorithm: rsaEncryption
           RSA Public Key: (1024 bit)
               Modulus (1024 bit):
                   00:d3:a4:50:6e:c8:ff:56:6b:e6:cf:5d:b6:ea:0c:
                   68:75:47:a2:aa:c2:da:84:25:fc:a8:f4:47:51:da:
                   85:b5:20:74:94:86:1e:0f:75:c9:e9:08:61:f5:06:
                   6d:30:6e:15:19:02:e9:52:c0:62:db:4d:99:9e:e2:
                   6a:0c:44:38:cd:fe:be:e3:64:09:70:c5:fe:b1:6b:
                   29:b6:2f:49:c8:3b:d4:27:04:25:10:97:2f:e7:90:
                   6d:c0:28:42:99:d7:4c:43:de:c3:f5:21:6d:54:9f:
                   5d:c3:58:e1:c0:e4:d9:5b:b0:b8:dc:b4:7b:df:36:
                   3a:c2:b5:66:22:12:d6:87:0d
               Exponent: 65537 (0x10001)
       X509v3 extensions:
           X509v3 Basic Constraints: critical
               CA:TRUE
   Signature Algorithm: md5WithRSAEncryption
       07:fa:4c:69:5c:fb:95:cc:46:ee:85:83:4d:21:30:8e:ca:d9:
       a8:6f:49:1a:e6:da:51:e3:60:70:6c:84:61:11:a1:1a:c8:48:
       3e:59:43:7d:4f:95:3d:a1:8b:b7:0b:62:98:7a:75:8a:dd:88:
       4e:4e:9e:40:db:a8:cc:32:74:b9:6f:0d:c6:e3:b3:44:0b:d9:
       8a:6f:9a:29:9b:99:18:28:3b:d1:e3:40:28:9a:5a:3c:d5:b5:
       e7:20:1b:8b:ca:a4:ab:8d:e9:51:d9:e2:4c:2c:59:a9:da:b9:
       b2:75:1b:f6:42:f2:ef:c7:f2:18:f9:89:bc:a3:ff:8a:23:2e:
       70:47
```

Fig. 2. A Sample X509 Digital Certificate.

The main assumption when using public key cryptography is that only the owner of the certificate knows the private key corresponding to the public key on the certificate and that users (also CA management) are responsible for ensuring the confidentiality of their private keys.

3 Formal Specification

In the following paragraphs, formal models of a Certificate Authority (CA), digital certificate, a user and an authentication server are constructed in order to develop an understanding of the whole PKI authentication system. An appropriate abstraction of the CA, certificate, user and the server can be well formulated as a state-based model. The model assumes the existence of the following types:

$$[Subject, Key, Data, SerialNb, CipherAlgName, CertAuthorityName, Date]$$

to denote the set of all possible users, public/private keys, encrypted and plain data, serial number of each digital certificate, names of cryptographic algorithms, names of the certificate authorities, and dates respectively. We assume that there are standard implementations of the cryptographic algorithms supported by the CA, the client and the server.

Let $[\![algo]\!]$ denotes the semantic of the cryptographic algorithm, *algo*. For example, one of the most common used cryptographic algorithms is *AES* [15]. The semantic of *AES* is a function that takes a key and data to be encrypted, and returns the data in encrypted form.

$$[\![\_]\!] : CipherAlgName \nrightarrow Key \rightarrow (Data \rightarrow Data)$$

Let *validDate* be a global relation that relates a date d and a pair of dates $(d1, d2)$ if and only if d lies between $d1$ and $d2$. This relation will be used to check whether a certificate has expired or not.

$$\begin{array}{|l} [Date] \\ validDate : Date \leftrightarrow (Date \times Date) \\ \hline \forall\, d, d1, d2 : Date \bullet d\ validDate\ (d1, d2) \Leftrightarrow d1 \leq d \land d \leq d2 \end{array}$$

Let *validPublicKeyPair* be a relation that associates a public to its corresponding private key. This relation is used to ensure that key generated for the user are valid.

$$\begin{array}{|l} [CipherAlgName, Data, Key] \\ validPKIKeyPair : Key \leftrightarrow Key \\ pkiAlgoName : CipherAlgName \\ m : Data \\ pk, sk : Key \\ \hline pk\ validPKIKeyPair\ sk \Leftrightarrow \\ \quad [\![pkiAlgoName]\!]\ pk\ ([\![pkiAlgoName]\!]\ sk\ \ m) = m \land \\ \quad [\![pkiAlgoName]\!]\ sk\ ([\![pkiAlgoName]\!]\ pk\ \ m) = m \end{array}$$

3.1 Specifying Certificate Authority

The role of the CA is to create and sign digital certificates with its secret key and maintain a list of certificates that are not valid anymore, also known as certificate revocation lists (CRL). Some of this information is made public so that clients and servers can verify certificates issued by the CA. The CA can be modeled as a data type which can be formulated as follows:

$$
\begin{array}{|l}
\textit{PublicCAInfo} \\
\textit{name} : \textit{CertAuthorityName} \\
\textit{publicKey} : \textit{Key} \\
\textit{ca\_revocationList} : \mathbb{P}\, \textit{SerialNb} \\
\textit{ca\_SupportedCrypto} : \mathbb{P}\, \textit{CipherAlgName}
\end{array}
$$

where *name*, is the the name of the CA; *publicKey*, denotes the unique public key of the CA; *ca_revocationList*, is a set of revoked certificates; and *ca_SupportedCrypto*, is a set representing the names of the supported cryptographic algorithms used for signing certificates and verification.

The CA also has information that are private and can only be known by the CA management such as the secret key corresponding to the CA's public key and a documentation of the issued certificates. This private information can be captured by defining additional fields, *secretKey*, and *issued*, a function that relates a serial number to a certificate.

$$
\begin{array}{|l}
\textit{PrivateCAInfo} \\
\textit{PublicCAInfo} \\
\textit{issued} : \textit{SerialNb} \pfun \textit{Certificate} \\
\textit{secretKey} : \textit{Key}
\end{array}
$$

The CA, *CertAuthority*, is then modeled as a data type which can be captured using the conjunction of the private and public CA information:

$$\textit{CertAuthority} \mathrel{\widehat=} \textit{PublicCAInfo} \land \textit{PrivateCAInfo}$$

3.2 Specifying Digital Certificate

A digital certificate typically consists of: *issuer*, the name of the certificate authority that issued the certificate; *serial* , serial number of the certificate; *subject*, subject's name to be associated with the public key on the certificate; *publicKey*, public key of the subject; *validity*, the validity date of the certificate; *pkiAlgoName*, the name of the public key algorithm with which the public key on the certificate is used (e.g. RSA or Diffie-Hellman); and *caSignatureAlgoName*, the name of the signature algorithm used by the CA to generate the signature on this certificate. The certificate data part can be abstracted in Z as follows:

$$
\begin{array}{|l}
\textit{CertificateData} \\
\textit{issuer} : \textit{CertAuthorityName} \\
\textit{serial} : \textit{SerialNb} \\
\textit{subject} : \textit{Subject} \\
\textit{publicKey} : \textit{Key} \\
\textit{validity} : \textit{Date} \times \textit{Date} \\
\textit{pkiAlgoName} : \textit{CipherAlgName} \\
\textit{caSignatureAlgoName} : \textit{CipherAlgName} \\
\hline
\end{array}
$$

The signature part consists of the digital signature, *caSignature*, created by the issuer, thereby binding the subject's identity to the specified public key. The data type certificate is described in the Z schema *Certificate*. In addition to these fields, there are others that we chose not to include as they are not relevant here.

$$
\begin{array}{|l}
\textit{Certificate} \\
\textit{CertificateData} \\
\textit{caSignature} : \textit{Data} \\
\hline
\end{array}
$$

3.3 Specifying User Credentials

The model of a user focuses primarily on the security knowledge that the user must possess and maintain for the purpose of authentication. The user's information that are public knowledge, *Public_UserCredrential*, comprises two components: (1) *cert*, a digital certificate issued by a trusted CA; (2) *userSupportedCrypto*, a list of supported cryptographic algorithms for key generation and ciphering data communicated with other entities. Here, for simplicity, it is assumed that the user has one certificate only. The user's information which is publicly known can be formulated in Z as follows:

$$
\begin{array}{|l}
\textit{Public\_UserCredrential} \\
\textit{cert} : \textit{Certificate} \\
\textit{userSupportedCrypto} : \mathbb{P}\ \textit{CipherAlgName} \\
\hline
\textit{cert.caSignatureAlgoName} \in \textit{userSupportedCrypto} \\
\textit{cert.pkiAlgoName} \in \textit{userSupportedCrypto} \\
\hline
\end{array}
$$

The information that can only be viewed by the user comprises two components: (1) user's private key, *secretKey*; (2) the list of trusted CAs, *trustedCA*, a set of CA's trusted by the user, *caKey*, a relation that associates CA's with their corresponding public keys. The user's private information can then be described in Z as follows:

$$
\begin{array}{|l}
\textit{Private\_UserCredrential} \\
\textit{secretKey} : \textit{Key} \\
\textit{trustedCA} : \mathbb{P}\, \textit{CertAuthorityName} \\
\textit{caKey} : \textit{CertAuthorityName} \pfun \textit{Key} \\
\hline
\textit{trustedCA} \subseteq \operatorname{dom} \textit{caKey}
\end{array}
$$

As a result, the state of a user can then be constructed of public and private information as shown in the *UserCredential* schema. The invariant states that the issuer of the certificate is trusted by the user.

$$
\begin{array}{|l}
\textit{UserCredential} \\
\textit{Public\_UserCredrential} \\
\textit{Private\_UserCredrential} \\
\hline
\textit{cert.issuer} \in \textit{trustedCA} \land \\
\textit{cert.publicKey}\ \textit{validPKIKeyPair}\ \textit{secretKey}
\end{array}
$$

3.4 Authentication Server

In this paper, it is assumed that the authentication server operates in a closed environment such as in a Bank or a University. The server administrator is responsible for maintaining a set of current *registered_users*. Therefore, authenticating users can be achieved by validating their certificates, verifying that they have knowledge of the private key corresponding to the one on the certificate, and by checking that their subject name is in the list of registered users. In open systems, there is no need for the *registered_users* set; any user who can pass the validation and verification phases can be considered authenticated. The abstract state of an authentication server consists of the following six components:

- *registered_users*, a set of known users
- *key_association*, a partial function that associates each subject with its public key
- *trustedCA*, a set of trusted CAs
- *caKey* relation that relates each CA trusted by the server to its public key
- *revoked*, a set of certificates that have been revoked
- *today*, the current date when the authentication is taking place
- *serverSupportedCrypto*, a list of cryptographic algorithms supported by the server

This can be formulated in as a schema as follows:

$$
\begin{array}{|l}
\textit{AuthenticationServer} \\
\textit{registered\_users} : \mathbb{P}\, \textit{Subject} \\
\textit{trustedCA} : \mathbb{P}\, \textit{CertAuthorityName} \\
\textit{key\_association} : \textit{Subject} \pfun \textit{Key} \\
\textit{caKey} : \textit{CertAuthorityName} \pfun \textit{Key} \\
\textit{revoked} : \mathbb{P}\, \textit{SerialNb} \\
\textit{today} : \textit{Date} \\
\textit{serverSupportedCrypto} : \mathbb{P}\, \textit{CipherAlgName} \\
\hline
\mathrm{dom}\, \textit{key\_association} = \textit{registered\_users} \land \textit{trustedCA} \subseteq \mathrm{dom}\, \textit{caKey}
\end{array}
$$

The invariant states that every user must have a public key. The authentication server decision whether the authentication has failed or succeeded is reflect by an output response drawn from the following type *Report*:

$$\textit{Report} ::= \textit{Auth\_Success} \mid \textit{Auth\_Failure}$$

Let the schema *Success* (*Failure* respectively) indicates the successful completion of (failure to complete respectively) the authentication operation.

$$
\begin{array}{|l}
\textit{Success} \\
\textit{resp}! : \textit{Report} \\
\hline
\textit{resp}! = \textit{Auth\_Success}
\end{array}
$$

$$
\begin{array}{|l}
\textit{Failure} \\
\textit{resp}! : \textit{Report} \\
\hline
\textit{resp}! = \textit{Auth\_Failure}
\end{array}
$$

3.5 Validating Certificate

An authentication server considers a digital certificate as valid if: (1) the certificate is issued by CA trusted by the server; (2) the certificate has not been revoked by checking it against a revocation list; (3) the certificate dates are still valid (hasn't expired); (4) the certificate has not been modified since it was created: this can be achieved by checking that the signature on the certificate is valid. It is important to know that verifying a certificate requires possession of the issuer's public Key, which is computed by $\textit{caKey}(\textit{cert}?.\textit{issuer})$). The verification process can be captured as follows:

$$
\begin{array}{|l}
\textit{CertValidationOk} \\
\Xi \textit{AuthenticationServer} \\
\textit{cert?} : \textit{Certificate} \\
\textit{certData} : \textit{Certificate} \pfun \textit{Data} \\
\hline
\textit{cert?.issuer} \in \textit{trustedCA} \land \textit{cert?.serial} \notin \textit{revoked} \land \\
\textit{today validDate cert?.validity} \land \\
[\![\textit{cert?.caSignatureAlgoName}]\!] \\
(\textit{caKeycert?.issuer})\ \textit{cert?.caSignature} = \textit{certData}(\textit{cert?})
\end{array}
$$

Where *certData* is a function that returns the data from a certificate. This data is usually signed by the CA to ensure the certificate integrity. This function is used in the certificate signature validation in order to compare the certificate signature with the data on the certificate. In practice, the *certData* function can be a hash function and the signature is the encryption of the hashed certificate using the issuing CA's private key.

3.6 Verifying User Knowledge of the Secret Key

The authentication server can establish that a user knows the private key corresponding to a public key on a certificate by having the following as inputs: a certificate, *cert*, a challenge *nonce* generated by the server, and what is believed to be the nonce signed with the user's private key, *signed_nonce*. The verification operation succeeds when the subject is a registered user, and the decryption of the *signed_nonce* with the public key on the user's certificate matches the original challenge, *nonce*, sent by server. The verification operation can be captured in Z as follows:

$$
\begin{array}{|l}
\textit{VertificationOk} \\
\Xi \textit{AuthenticationServer} \\
\textit{cert?} : \textit{Certificate} \\
\textit{nonce?} : \textit{Data} \\
\textit{signed\_nonce?} : \textit{Data} \\
\hline
\textit{cert?.subject} \in \textit{registered\_users} \land \\
[\![\textit{cert?.pkiAlgoName}]\!]\ \textit{cert?.publicKey}\ \ \textit{signed\_nonce?} = \textit{nonce?}
\end{array}
$$

3.7 Authenticating Users

A successful authentication operation can be viewed as a combination of successful certificate validation and client signature verification. This is modeled as a conjunction of the following operations:

$$\textit{AuthenticationOk} \mathrel{\widehat=} \textit{CertValidationOk} \land \textit{VertificationOk}$$

The whole authentication operation can then be captured as follows:

$$Authentication \mathrel{\widehat{=}} (AuthenticationOk \land Success) \lor Failure$$

4 Process Architecture for PKI-Based Authentication

So far, we have described an authentication server and a user as completely independent systems. The CSP events depends on the Z operation defined earlier. The authentication server (denoted AS) interacts with the clients by sending and receiving the above messages on designated communication channels. For example, sIn (from client) is the input channel on which the certificate and client's response are communicated to the server; and $sOut$ (to client) is the output channel on which the server's challenge and authentication response are sent. The authentication server behavior can be informally described as follows: (1) the AS receives a certificate from a client, say c; (2) sends a plain challenge to the client, say t; (3) receives a signed challenge from the client, m; (4) and checks validity of c (using the $Authentication$ operation described in the previous section in Z). The interface of AS and its behavior are described below:

$$\alpha AS = \{sIn?req, sOut!resp\}$$
$$AS = sIn?c \rightarrow sOut!t \rightarrow sIn?m \rightarrow$$
$$(\ sOut!Auth\_Success \rightarrow$$
$$THREAD(c, session\_Key) \ |||\ AS \ \not< \text{pre}\ (Authentication(c, nonce, reply)) \not>$$
$$authentResp!Failure \rightarrow AS)$$

The CSP expression pre $(op\ (i_1, i_2..i_n))$ (where pre is the precondition operator) denotes the precondition on the state of the system and inputs which makes successful completion of the operation op. If the precondition is true, then the server creates a thread with a session key, say $THREAD(c, session\_key)$, to serve the client while still serving other clients. The description of the thread depends on the user's role.

(Server Model in CSP) **(Client Model in CSP)**

Fig. 3. CSP models for Client and Server

4.1 Client Behavior

A typical interaction between a user, $u_0 =< cert_0, issuer_0, publickey_0, secretkey_0 >$, with a valid certificate and an authentication server can be modelled as follows:

$$CLIENT(u_0) = cOut!u_0.cert_0 \rightarrow cIn?t \rightarrow cOut!m \rightarrow cIn?resp \rightarrow SKIP$$

Where, t is the random challenge, $m = [\![u_0.cert_0.pkiAlgName_0]\!]\, u_0.secretKey_0\ t]$ is the encryption of the challenge with the user's secret key using the PKI algorithm named on the certificate, and *resp* is the authentication result. If the authentication succeeds, the client will behave as authenticated

The result of $CLIENT(u_0)$ sequence of interactions with the authentication server is calculated using a parallel composition (Figure 4) of $CLIENT$ and AS processes as follows:

$$CLIENT(u_0)[rep/cOut, req/cIn] \parallel AS(state)[rep/sIn, req/sOut] \\ = req!Success \rightarrow AS(state)$$

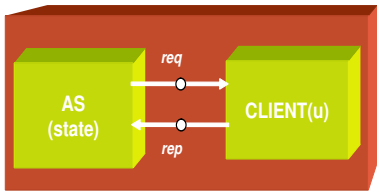

Fig. 4. PKI Authentication in CSP

5 Administrative operations

We consider some of the main administrative operations on the *AuthenticationServer* system namely: *AddUser*, *RemoveUser*, *AddTrustedCA*, and *RemoveTrustedCA* respectively.

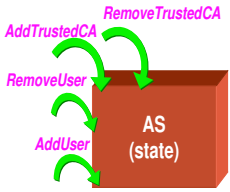

Fig. 5. PKI Administrative Operations

The first operation, *AddUser*, receives the username and public key of a user, *username*? and *publicKey*? respectively, and changes the state of *registered_users* and *key_association* respectively in the authentication server. The precondition for this operation to succeed states that the *username*? should not already be in the *registered_users* set (unique) otherwise all other components remain unchanged. The operation is captured in Z as follows:

$$
\begin{array}{|l}
\textit{AddUser} \\
\Delta \textit{AuthenticationServer} \\
\textit{username?} : \textit{Subject} \\
\textit{publicKey?} : \textit{Key} \\
\textit{resp!} : \textit{Report} \\
\hline
\textit{username?} \notin \textit{registered\_users} \Rightarrow \\
\quad \textit{registered\_users}' = \textit{registered\_users} \cup \{\textit{username?}\} \land \\
\quad \textit{key\_association}' = \\
\quad \textit{key\_association} \cup \{\textit{username?} \mapsto \textit{publicKey?}\} \land \\
\quad \textit{resp!} = \textit{Succeeded} \\
\textit{username?} \in \textit{registered\_users} \Rightarrow \\
\quad \textit{registered\_users}' = \textit{registered\_users} \land \\
\quad \textit{key\_association}' = \textit{key\_association} \land \textit{resp!} = \textit{Failed} \\
\end{array}
$$

The operation *RemoveUser* receives a username as argument and removes it with its corresponding public key from the authentication server. The precondition states that the *username?* must exists in the *registered_users* set.

$$
\begin{array}{|l}
\textit{RemoveUser} \\
\Delta \textit{AuthenticationServer} \\
\textit{username?} : \textit{Subject} \\
\textit{resp!} : \textit{Report} \\
\hline
\textit{username?} \in \textit{registered\_users} \Rightarrow \\
\quad \textit{registered\_users}' = \textit{registered\_users} \setminus \{\textit{username?}\} \land \\
\quad \textit{key\_association}' = \{\textit{username?}\} \lhd \textit{key\_association} \land \\
\quad \textit{resp!} = \textit{Succeeded} \\
\textit{username?} \notin \textit{registered\_users} \Rightarrow \\
\quad \textit{registered\_users}' = \textit{registered\_users} \land \\
\quad \textit{key\_association}' = \textit{key\_association} \land \textit{resp!} = \textit{Failed} \\
\end{array}
$$

AddTrstedCA operation receives a CA's name and public key respectively as arguments and add them to the trusted CA relation, *trustedCA* on the server. The precondition for this operation is that the CA's name should not already be in the trusted CA list.

$$
\begin{array}{|l}
\textit{AddTrustedCA} \\
\Delta \textit{AuthenticationServer} \\
\textit{ca\_Name?} : \textit{CertAuthorityName} \\
\textit{ca\_Key?} : \textit{Key} \\
\textit{resp!} : \textit{Report} \\
\hline
\textit{ca\_Name?} \notin \textit{trustedCA} \Rightarrow \\
\quad \textit{trustedCA}' = \textit{trustedCA} \cup \{\textit{ca\_Name?}\} \land \\
\quad \textit{caKey}' = \textit{caKey} \cup \{\textit{ca\_Name?} \mapsto \textit{ca\_Key?}\} \land \\
\quad \textit{resp!} = \textit{Succeeded} \\
\textit{ca\_Name?} \in \textit{trustedCA} \Rightarrow \\
\quad \textit{trustedCA}' = \textit{trustedCA} \land \\
\quad \textit{caKey}' = \textit{caKey} \land \textit{resp!} = \textit{Failed}
\end{array}
$$

RemoveTrustedCA operation received the name of a trusted CA as input and removes it from the trusted CA relation. The precondition for this operation to succeed is that the CA's name should already be on the trusted list.

$$
\begin{array}{|l}
\textit{RemoveTrustedCA} \\
\Delta \textit{AuthenticationServer} \\
\textit{ca\_Name?} : \textit{CertAuthorityName} \\
\textit{resp!} : \textit{Report} \\
\hline
\textit{ca\_Name?} \in \textit{trustedCA} \Rightarrow \\
\quad \textit{trustedCA}' = \textit{trustedCA} \setminus \{\textit{ca\_Name?}\} \land \\
\quad \textit{resp!} = \textit{Succeeded} \\
\textit{ca\_Name?} \notin \textit{trustedCA} \Rightarrow \\
\quad \textit{trustedCA}' = \textit{trustedCA} \land \textit{resp!} = \textit{Failed}
\end{array}
$$

6 Related Work

Formal methods have been used for specifying, formulating, designing, analyzing, and verifying cryptographic protocols and particularly authentication protocols [20,3,4]. The literature also contains a number of approaches for applying formal methods to the PKI problem [13,16,9,11,23,14]. In [14], the authors presented a formal specification of a certificate management management system and some basic operations such as certificate issuing, certificate revocation using a state based approach. Schneider, Ryan, and Lowe developed formal security analysis methods which were successfully used to find security flaws or prove their absence using CSP [20,3]. Woodcock and his colleagues [17] as well as Laurence [12] have used a combination of state-based Z specifications and event-based CSP models to capture and reason about distributed applications.

7 Conclusion

In this paper, key components of certificate based PKI were formalized in order to have a clear and rigorous understanding of them and to avoid ambiguities. These components were used as building block to construct a formal model for certificate based PKI authentication. The formalization of the PKI authentication system is expressed by combining some aspects of **Z** and CSP notations. The consistency of the model is checked with ZTC tool.

References

[1] Bickmore, R. and Baltimore Technologies, *Implementing a PKI*, Information Security Technical Report, **5(4)**(2000), pages 33–38.

[2] Chokhani, S., "Computer Security Hand book", chapter *Public Key Infrastructures and Certificate Authorities*, 4th Ed., Wiley, 2002.

[3] Gavin, L., *Breaking and Fixing the Needham-Schroeder Public-Key Protocol using CSP and FDR*, volume**1055** of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[4] Giampaolo, G., F. Masacc F, and L. C. Paulson, *Verifying the SET Registration Protocols*, IEEE Journal on Selected Areas in Communications, **21(1)** (2003),77–87.

[5] Gollmann, D, "Computer Security," 2nd Ed., Wiley, 2005.

[6] Hoare, C. A. R., "Communicating Sequential Processes," Prentice Hall, 1985.

[7] Housley, R., W. Ford, T. Polk, and D. Solo, *Internet X.509 Public Key Infrastructure – Certificate and CRL Profile*, RFC **2459** (RFC 3280–2002), January 1999.

[8] Houston, ISC and M. B. Josephs, *Specifying distributed CICS in Z: accessing local and remote resources*, Formal Aspects of Computing, **6(5)**(1994), 569–579.

[9] Howell, J. and D. Kotz, *A Formal Semantics for SPKI*, Technical report, Hanover, NH, USA, 2000.

[10] Jia, X. , "ZTC: A Type Checker for Z Notation – User's Guide", DePaul University, Institute for Software Engineering, Department of Computer Science and Information Systems, Chicago,Illinois, USA, version 2.03 Ed., August 1998.

[11] Kohlas R. and U. M. Maurer, *Reasoning about Public-Key Certification: On Bindings between Entities and Public Keys*, Proceedings of the Third International Conference on Financial Cryptography,1999, pages 86–103, Springer-Verlag, UK

[12] Lawrence, J., *Practical Application of CSP and FDR to Software Design*, 25 Years Communicating Sequential Processes, volume **3525** (2004) of LNCS Springer, pages 151–174.

[13] Li, N. and J. Feigenbaum, *Nonmonotonicity, User Interfaces, and Risk Assessment in Certificate Revocation*, Proceedings of the 5th International Conference on Financial Cryptography, 2002, pages 166–177, Springer-Verlag, UK.

[14] Liu, C. and M. A. Ozols and M. Henderson and A. Cant, *A State-Based Model for Certificate Management Systems*, Proceedings of the Third International Workshop on Practice and Theory in Public Key Cryptography, pages 75–92, Springer-Verlag, UK, 2000.

[15] Menezes, A., P. van Oorschot, and S. Vanstone, "Handbook of Applied Cryptography", CRC Press, 1997.

[16] Maurer, U. M., *Modelling a Public-Key Infrastructure*, Proceedings of the 4th European Symposium on Research in Computer Security, pages 325–350, Springer-Verlag, UK, 1996.

[17] Oliveira, M. V. M., A. L. C. Cavalcanti, and J. C. P. Woodcock, *Refining Industrial Scale Systems in Circus*, in I.R. East, J. Martin, P.H. Welch, D. Duce, and M. Green, editors, Communicating Process Architectures 2004, volume **62** (2004), of Concurrent Systems Engineering Series, pages 281–309, IOS Press.

[18] Palmer, T., *PKI needs good standards?*, Information Security Technical Report, **8(3)** (2003), pages 6–13.

[19] Piper, F., G. Price, and K. Paterson,*Introduction*,Information Security Technical Report, **8(3)** (2003), 6–13.

[20] Ryan, P., S. Schneider, S. Goldsmith, G. Lowe, G., B. Roscoe, "Modelling and Analysis of Security Protocol", Pearson Education, 2001.

[21] Smart, N. " Cryptography: An Introduction", McGraw-Hill, 2003.

[22] Spivey, J. M., "The Z Notation: A Reference Manual", 2nd Ed., Prentice Hall,1992.

[23] Smith, S. W., *Outbound authentication for programmable secure coprocessors*, ESORICS '02: Proceedings of the 7th European Symposium on Research in Computer Security, 2002, pages 72–89, Springer-Verlag, UK.

[24] Thomas, S., "SSL and TLS Essentials, Securing the Web", Wiley, 2000.

[25] Welch, V., F. Siebenlistand, I. Foster, J. Bresnahan. K. Czajkowski, J. Gawor, S. Meder, L. Pearlman, and S. Tuecke, *Security for Grid Services*, 12th International Symposium on High Performance Distributed Computing (HPDC-12), IEEE Press, 2003.

[26] Woodcock, J. and J. Davies, "Using Z Specification, Refinement, and Proof", C.A.R Hoare series editor, Prentice Hall International, 1996.