Certification

by

Zdeněk Říha

FI MU Report Series FIMU-RS-98-07

Contents

Chapter 1

Introduction

1.1 Motivation

Modern wide area networks allow communication between two parties that are located anywhere in the world. They are inexpensive and fast. However without using cryptography, such communication is completely open to be read by anyone else.

Traditional symmetric cryptography cannot usually be used because this requires a safe exchange of the secret key. Fortunately, cryptography has provided a solution with a technique called asymmetric cryptography (also called public-key cryptography). Asymmetric cryptography allows completely private communication between parties that have never met. Public keys of both parties are publicly exchanged and then the secret key is privately agreed upon. The combination of symmetric and asymmetric cryptography is used for efficiency reasons. All communications after the exchange of the public keys is encrypted and no one else can read (understand) them. Still, the communication problem of the previously unknown parties has not been solved entirely. The communicating parties cannot be certain that the public keys they received really belong to the other party. They also cannot be certain of the other party's characteristics, or even who or what the other party really is. One can have a perfectly private communication with a thief, which is not much safer. Public-key cryptography can make communication private, but not secure. For secure communication, one needs to know the other's party attributes, such as their name, e-mail, account number, identity and authorizations.

Insecure communication is a serious risk for most commercial applications. In banking, security is crucial for sales and the confidential exchange of data between companies. What is needed is a method of verifying if an attribute, such as a public key or an account number really belongs to the other party. Only then can two previously unknown parties safely communicate for the first time.

Authentication has been traditionally based on knowledge of a shared secret (e.g. a password), however this situation is beyond the usual concept of authentication because the parties are considered previously unknown. Instead there is a primary step called certification.

1.2 Certification

Certifications are specific procedures which compare references with measurements that allow a party to bind an attribute to an unknown party. Certification is similar to the measurement of distance. It cannot be absolute. One always needs a reference. If the reference is incorrect, spoofed or falsified, then the result will also be incorrect. For the measurements of height or length there is a standard and easy reference, but for certification there is no equally free, common, faithful and worldwide reference available to all parties that wish to communicate.

Public-key cryptography was first introduced by Diffie and Hellman in 1976. They believed they had radically solved the problem of the key management. "The public key can be made public by placing it in a public directory along with the user's name and address." This central authority was known as the *Public File*, however the problem of the secure distribution of the Public File remained. In 1978 Loren Kohnfelder [MIT] invented a new construct: "The Public File digitally signs all of its transmission, so that enemy impersonation of the Public File is precluded". He called his new construct a certificate. It is a digitally-signed data record containing a name and a public key. (it is a public-key certificate). This is the history of the first certificate.

Absolute certification methods are logically impossible because a certificate cannot certify itself. A reference is always necessary. There are three main types of methods that have been proposed to deal with this concern:

- Directory methods, based on Certification Authorities (CA): X.509 (e.g. SSL, PKIX)
- Referral methods, based on “introducers” of keys: PGP
- Collaborative methods: SKIP

Each of the above certification methods deals with the basic certification problem in a different way. All of them are described in the following chapters.

Certification is based on two different properties: trust (the qualitative property) and keys (the quantitative property). In any certification system the certificate’s trustworthiness is not magically infused from the certificate’s issuer. A certificate is trustworthy as decided by the user. He relies on the information, he is the one at risk. The decision must be based on *the user’s trust* in the certificate’s issuer and as a function of costs, risks, situation and so on.

It is not easy to define trust. Here it is necessary to mention that most protocols do not define what trust is. They only define the manner in which to convey it. Such an approach ignores the properties of trust, which can lead to security risk and a lack of standardization. This is one of the basic contradictions in the protocols commonly used today and it may cause other problems.

1.3 Bindings of a certificate

The X.509 certificate and other similar certificates bind *a name* to *a public key*. The name is unique within a space and is called *the Distinguished name* (DN). People often refer to such a certificate as an “identity certificate”, but this is not correct. It cannot bind *an identity* to a key. It binds a name to a key. This is a serious mistake. The problem lies in assuming that name implies identity. The following chain of logical implications is mistakenly used: Name $\rightarrow$ person $\rightarrow$ characteristics $\rightarrow$ identity.

The purpose of a common name is to identify someone. If there is any confusion with the name, we adjust the name so that it continues to function as an identifier. We are limited in the number of names we can remember, but for most of our lives and most of the human history we have all had relatively small communities around us. The names are just large enough to avoid confusion in communities that are normally around us. As a result, we are not normally troubled by name duplications. This leads us to the logical implication name → person. We tend to assume that if we know a person, we know their characteristics, such as marital status, financial situation, their trustworthiness and so on. Thus we tend to extend the definition to name → person → characteristics. The last implication characteristics → identity is directly a matter of definition. The Webster's dictionary defines identity as "... distinguishing character or personality of an individual ..." . Because a distinguishing character or personality is one of those characteristics expected to be known about a person, this definition of identity gives us a final implication, so that we obtain: name → person → characteristics → identity. If names were globally unique and unchanging, then we could be able to expect that, by definition, if the names are the same then the people are also the same. This is not, however, the case.

The chain of logical implications name → identity depends twice on characteristics of small communities, but modern wide area networks, such as the Internet, are not small communities.

The failure of the name → identity assumption can be easily demonstrated. I come to a CA and receive a certificate for ("Zdeněk Říha", my public key). I then go to a bank and ask for a list of all the accounts owned by Zdeněk Říha. The fulfillment of my request would certainly be a security policy violation.

In order to grant access or authorization, we must be able to identify the recipient of that authorization. Imagine a bank granting electronic access to check your account. You go to the bank and request that you be issued a certificate to give you on-line access to your account.

The request is sent to the back room where the account information (name and address) is used to look up your DN in a global directory of certificates. From a list of available certificates, the bank office will come up with a set of possible matches to that account owner. If there is none, there is no security problem, because no certificate will be issued. If there are multiple matches, then the bank policy will decide what happens. If no certificate is issued, then there is no security flaw, but if "the most likely" match is chosen and the appropriate certificate is issued, then this involves a human guess and there is a security problem. If there is only one match then the certificate is issued and is presumably correct, however the bank cannot be sure that the match is really correct.

1.4 Identifiers

What is missing in the above example which is necessary for proper identification is *a globally unique* and *universally used* identifier of persons. There are people that act as the Distinguished Names were such identifiers, but this is not correct. Distinguished Names do not guarantee that some evil-does is not capable of doing evil under one name, changing his name and starting all over again with a new DN.

It is unlikely that we will ever see a *global* identifier. In the Czech Republic we have a birth number that is assigned to a person at birth and remains the same for their whole lifetime. It works fine as a local identifier within the country, although there were a few problems with its uniqueness. In the U.S. the national social security number is used as an identifier. Again, this works fine locally within the U.S. but there is no global identifier for everyone on Earth. There are some countries where a national ID is used, but where laws exist against being required to use these numbers except for the official government purposes (e.g. social insurance numbers in Canada). In such countries the national ID uniquely identifies a person, but it cannot be universally used.

There are multiple competing CAs creating disjointed Distinguished Name spaces and therefore a person is able to drop one DN and set another, just by changing the CA. The notion of an inescapable identifier is not close to realization and is not likely ever to be.

1.5 Conclusions

As we have already learned, the application of public-key systems requires the user to be confident that the public key belongs to the correct remote party be it a person or a system, which the asymmetric cryptography mechanism will be used with. This confidence is obtained through the use of public key certificates, which are data structures that bind public key values to subjects. The binding is achieved by having a trusted certification authority digitally sign such a certificate.

Each certificate has a limited validity period which is indicated in its signed content. Because the certificate's signature and the time validity can be independently checked by a certificate-using user, certificates can be distributed via insecure communications and systems.

Chapter 2

X.509

2.1 X.509 standard

The standard know as ITU-T X.509 (formerly known as CCITT X.509) or ISO/IEC 9594-8, which was first published in 1988 along with the X.500 Directory recommendation, defines a standard for the certificate format. The standard describes two levels of authentication:

- simple authentication using only a password as a verification of claimed identity (this is not suitable for situations where the parties are considered previously unknown)
- the strong authentication that uses cryptographic techniques

The certificate format published in 1988 in the original version of the standard is called as the version 1 format (X.509v1). When the X.500 was revised in 1993, two more fields were added to the certificate structure. It is referred as the version 2 format (X.509v2). These 2 fields are used to support directory access control.

In 1993 the PEM (Privacy Enhanced Mail) proposal was first published. Its public-key infrastructure was based on the X.509v1 certificates. Experience showed that v1 and v2 certificates need to be extended in several aspects. ISO/IEC and ANSI X9 soon developed the X.509 version 3 (X.509v3) certificate format. The version 3 extends the previous version (v2) by adding provision for additional extension fields.

Particular extension field types may be specified in standards or may be defined and registered by any organization. X.509v3 was accepted as a official standard in 1996.

A user who requires knowledge of a public key needs to obtain and validate a certificate containing the required public key. It may happen that the user does not already hold an assured copy of the public key of the certification authority that signed the certificate. He will then need an additional certificate to obtain that public key. Generally, a chain of multiple certificates may be needed, consisting of the certificate of the public-key owner signed by one CA and zero or more additional certificates of CAs signed by other CAs. Such a sequence, called *certification chain*, is required as the user initially has only a limited number of assured CA public keys.

The structure of CAs is hierarchical. There are three types of certification authorities (according to X.509v1):

- Policy Registration Authority (PRA). This is the root of the certification hierarchy at the first level. This authority issues only certificates for the next level of authorities (PCAs). According to X.509v1 all the certification chains start with this root authority.

- Policy Certification Authorities (PCA). They are at the second level of the hierarchy and each PCA is certified by the root authority (PRA). A PCA must create and publish a statement of its policy with respect to certifying users or certification authorities. Distinct PCAs aim to satisfy different needs of different users.

- Certification Authorities (CA). CAs are at the third level of the certification hierarchy and can also be at lower levels. Those at level 3 are certified by PCA. Certification Authorities usually represent particular organizations or their units.

Trust associated with a certification chain is implied by the PCA name. The name rule ensures that CAs below one PCA are constrained to a limited set of entities they can certify. (e.g. a CA for an organization can only certify entities in that organization's name tree).

Privacy Enhanced Mail (PEM) as in RFC 1422 was based upon X.509v1 certificates, however the limitations of X.509v1 required a few structural restrictions. The restrictions include:

- The strict top $\rightarrow$ down hierarchy. All certification chains must start from the root certificate.

- Naming rules to restrict the names of a CA's subjects.
- Knowledge of individual PCA is required to decide if a chain can be accepted.

Version 3 of X.509 solves these problems and that's why newly designed certification systems (such as PKIX) take advantage of the X.509v3.

When a certificate is issued, it is expected to be used for its entire period of validity, however various circumstances may cause a certificate to become invalid before it expires. Such circumstances might include change of name, a change of association between the subject and the CA or compromise of the corresponding private key. Under these circumstances, the CA needs to revoke the certificate.

X.509 defines one method of certificate revocation. This method involves each CA periodically issuing a signed data structure called a certificate revocation list (CRL). A CRL is a time-stamped list of revoked certificates which is signed by a CA and made freely available in a public repository. Each revoked certificate is identified by its certificate serial number. When a user uses a certificate, it not only checks the certificate's signature and validity, but also obtains "suitably-recent" CRL and checks that the certificate serial number is not on the CRL. The interpretation of "suitably-recent" is dependent upon the local policy and usually means the most recently issued CRL. Each CA issues a new CRL on a regular basis as stated in the CPL. Entries are added to the CRL as revocations occur and an entry may be removed after the certificate expires. CRLs may be distributed by exactly the same ways as certificates themselves (i. e. via insecure communications and systems).

2.2 SSL

SSL (Secure Socket Layer) is not a socket protocol as the name might imply. It is a protocol which may be placed between a reliable connection-oriented network layer protocol (TCP/IP) and the application layer (e.g. HTTP).

SSL provides for secure communication between client and server by allowing mutual authentication, the use of digital signatures for integrity and encryption for privacy. SSL is perhaps the widest used security protocol on the Internet today and implements X.509 certificates as interpreted by the SSL's proponent, Netscape. There are also other implementations of SSL, such as the free implementation called SSleay, for example.

A certificate associates a public key with a subject which can be an individual, a server or another entity. It is important to note, that SSL claims that a certificate binds a public key to an identity. Such claims were discussed in the introduction.

The information about the subject includes identifying information (DN) and the public key. A certificate also includes the identification and signature of the CA that issued the certificate and the validity period. It may also include administrative information such as a version or a serial number.

`Subject:`	`distinguished name, public key`
`Issuer:`	`distinguished name, public key`
`Validity:`	`not before ≪date≫, not after ≪date≫`
`Administrative info:`	`version, serial number`
`Extended info`	

Distinguished names meet the X.509 standard and use the following fields:

Abreviation	Name of the field	Example
CN	Common Name	Zdeněk Říha
O	Organization or Company	Masaryk University
OU	Organizational Unit	Faculty of informatics
L	City or Locality	Brno
SP	State or Province	Moravia
C	Country (ISO code)	CZ

A certificate authority may define a policy specifying which fields of the distinguished names are required and which are optional. It may also specify requirements of the field contents. As an example, the Netscape browser requires that the Common name (CN) for a server certificate matches a regular expression for the domain name of that server.

A CA may also issue a certificate for another CA. When examining a certificate, the user may need to examine the certificate of the issuer for each parent CA, until reaching one, which he has confidence in. This presents the problem of who will issue, and sign, the certificate for the top-level authority. In this case the certificate is “self-signed”, so the issuer of the certificate is the same as the subject. As a result, the user must be very careful in trusting a self-signed certificate.

Since the CA paradigm essentially relies on an authentication chain which ends in another CA that certifies itself, the validity problem is shifted from a local perspective to a global perspective, wherein the whole chain depending on the final link. In the end, the possibility of fraud is high since one weak link may compromise a whole chain of certificates.

Most SSL-capable browsers are pre-configured to trust "well known" certificate authorities. As a result, if a certificate is acceptable by the Netscape SSL product, it does not necessarily mean it will be considered acceptable by products from Microsoft or RSA. From the beginning, the decision of trust is completely removed from the user, which contradicts the fact that the user should play the central role in decisions. Moreover, SSL does not check certificate revocation lists (CRLs). So they are practically useless.

2.3 PKIX

2.3.1 The form of the certificate

PKIX is an Internet Draft. Internet Draft are working documents of the Internet Engineering Task Force (IETF) and its working groups. The most recent version at the time of writing is described here. That is the seventh draft published on March 25, 1998.

The purpose of the PKIX is to specify a profile and certificate management system for use of the X.509 certificates within the Internet. It is intended for those, who wish to make use of X.509 technology in applications such as WWW or e-mail.

The PKIX Internet Draft defines the Internet public key infrastructures of the X.509 certificates and the corresponding CRLs. Public key certificates are based upon the X.509v3 certificate format, however the deployment of an X.500 directory system is not assumed. The use of an X.500 directory is not prohibited, but there are other ways of distributing certificates and CRLs.

Basing PKIX upon the new v3 of X.509 has a few advantages over PEM, which was based upon the version 1:

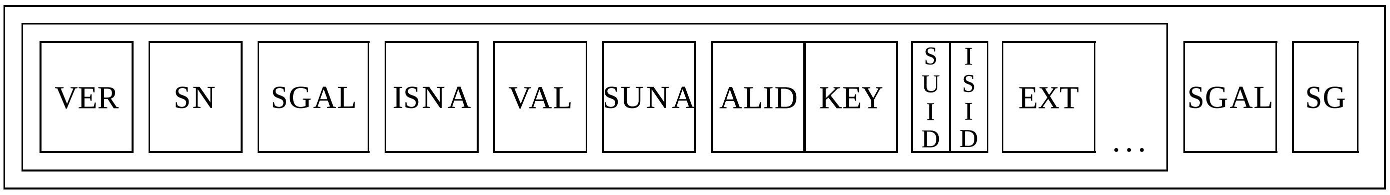
- Certification chains do not need to start with the root CA anymore.

Starting with the public key of a local CA has certain advantages. In many cases the local CA is the most trusted one.

- Name constraints may be imposed by appropriate name constraint extension in the certificate, but it is not mandatory required.
- Policy extensions and policy mappings replace the PCA concept.

This enables a greater degree of automation. The client software can decide whether the certification chain is acceptable based on the content of the certificate instead of the knowledge of the PCAs. This permits the full process of the certification chain validation to be implemented in software.

Let's take a look at the actual format of the certificate. The following picture shows its fields:



The certificate consists of 3 basic parts: of the main body, of the signature algorithm identifier (SGAL) and the signature of the certifying CA (SG). The main body of the certificate includes following fields:

- VER – version of the X.509 standard (may be v1, v2 or v3)
- SN – the serial number of the certificate (unique within the particular CA)
- SGAL – algorithm identification for the signature of the CA
- ISNA – issuer name – the name of the entity who has signed the certificate. (If the name is specified in the 'issuer name', it must contain the X.500-style DN. The issuer name may also be specified in 'Issuer Alt Name' (e.g. Internet-style address.)
- VAL – period of validity. Consists of two dates, the first and the last on which the certificate is valid.
- SUNA – subject name (similar conventions as for the issuer name)

- PKIN – subject public key info. Consists of the identification of the algorithm (ALID) and the key itself (KEY).

- Subject Unique Identifier (SUID) and Issuer Unique Identifier (ISID). These are part of the v3 standard, but the PKIX recommends that they not be used.

- EXT – extensions. One or more certificate extensions (for version 3 certificates only). Each extension consists of an extension object ID, extension value and the optional boolean ’crucial’.

2.3.2 Certificate Extensions

The certificate extensions defined for X.509v3 certificates provide methods for associating additional attributes with users or public keys, for managing the certification hierarchy and for managing the CRL distributions. Each extension in a certificate may be designed as crucial or non-crucial. The certificate-using system must reject the certificate if it encounters a crucial extension it does not recognize (while a non-crucial extension may be ignored).

X.509v3 defines a set of standard extensions and permits the use of additional “private” extensions. There are 14 standard extensions defined in X.509v3 and one private defined by PKIX. CAs or clients do not have to support all of them. The PKIX standard defines which extensions must be supported.

Here is the list of the standard extensions:

- Authority Key Identifier – this extension helps to identify the public key corresponding to the private key used to sign the certificate. It is useful when the issuer has multiple signing keys.

- Subject Key Identifier – identifies the particular public key used in an application

- Key Usage – defines the purpose of the key contained in the certificate (a key may be used for verifying digital signatures, providing a non-repudiation service, encrypting user data, for the key agreement, for verifying a signature on a certificate or verifying a signature on a CRL)

- Private Key Usage Period – permits the ability to specify a different validity period for the private key than the certificate.
- Certificate Policies – is a sequence of one or more policy information terms that indicate the policy under which the certificate has been issued and the purpose for which the certificate may be used. Two policy qualifiers are defined: "CPS pointer" contains a URI to a Certification Police Statement (CPS) of the CA and "User Notice" is intended to be displayed when the certificate is used.
- Policy Mappings – lists one or more pairs of object IDs. Each pair includes an "Issuer Domain Policy" and a "Subject Domain Policy". The pair indicates that the issuing CA considers its policy equivalent to the subject CA's policy.
- Subject Alternative Name – allows additional identities to be bound to the subject (e-mail address, DNS or IP address). The subject alternative name is considered to be definitively bound to the public key, thus all parts of the alternative name must be verified by the CA.
- Issuer Alternative Name – is used to associate Internet identifiers with the issuer
- Subject Directory Attribute – used only for local purposes
- Basic constraints – identifies whether the subject of the certificate is a CA and specifies how deep the certification chain may exist through this CA.
- Name constraints – indicates the name space within which all subject names in subsequent certificates in a certification chain must be located. This may apply to the subject DNs and / or subject alternative names.
- Policy Constraints – can be used in certificates issued for CAs. It constrains the validation of certification chains in two ways: They can be used to prohibit policy mappings or require that each certificate in a chain must contain an acceptable policy identifier.
- CRL distribution points – this extension defines how the CRL information is obtained

- Extended Key Usage Field – indicates for which purpose the certified public key may be used (in addition to the basic purposes defined in the “Key Usage” field).

And finally the private PKIX Internet extension:

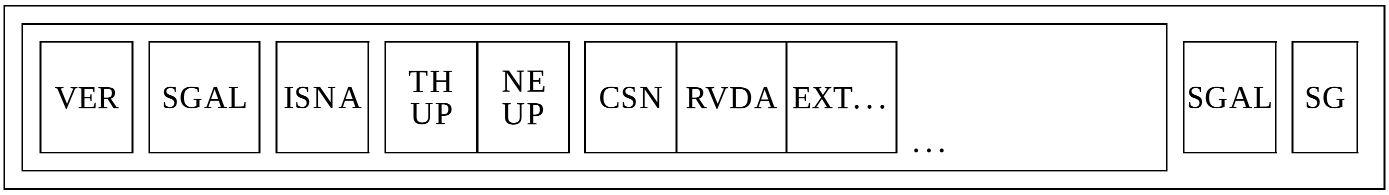
- Authority Information Access – indicates how to access CA information and services. These may include on-line validation services and CA policy data.

2.3.3 Certificate revocation lists

The PKIX draft uses X.509v2 CRLs in the standard format without any private Internet CRL extension.

CAs have to provide revocations, but they are not required to issue CRLs if they provide other revocation methods. If they do issue CRLs they are required to issue version 2 CRLs and specify the date when the next CRL will be issued.

The format of v2 CRL is shown at the following picture:



The CRL is signed by the issuing CA (SG) using the signing algorithm SGAL. The body of the CRL consists of:

- VER – version (must be v2)
- SGAL – algorithm used by the CA to sign the CRL
- ISNA – issuer name (see the previous section)
- THUP – this update, the issue date of the CRL
- NEUP – next update, indicates the time / date by which the next CRLs will be issued.

And the list of the revoked certificates follows. Each entry includes:

- CSN – Certificate Serial Number. Uniquely identifies the certificate to be revoked
- RVDA – revocation date, the date on which revocation occurred
- EXT – extensions, associate additional attributes to a CRL entry
  - Authority Key Identifier – identifies the signing key used to sign this CRL
  - Issuer Alternative Name – Internet name of the issuer of this CRL
  - CRL Number – monotonically increasing sequence number
  - Issuing Distribution Point – a crucial extension that identifies the CRL distribution point for a particular CRL
  - Delta CRL Indicator – enables the distribution of CRL updates, listing only new entries. The base used to create this difference-only CRL must be specified.
  - Certification issuer - used when the certificate issuer differs from the CRL issuer
  - Extensions – separate characteristics of particular CRL entries:
    - Reason Code – identifies the reason for certificate revocation (e.g. key compromise, CA compromise, etc.)
    - Hold instruction code – indicates an action to be taken after encountering that the certificate has already been place on hold
    - Invalidity date – provides the date when the private key is known or suspected to have been compromised.

2.3.4 Validation of a certification chain

The process followed for the certification chain verification involves checking the binding of the subject name (a DN or an Alternative Name). The binding is limited by constraints, which are specified in the certificates forming the chain. The usage of basic constrains and policy constraints enables to automate certification chain processing.

A certification chain is a sequence of n certificates $C_i, i = 1..n$ where:

- C_1 is the self signed certificate that all certification chains have to start with (the public key it contains is already trusted). The C_1 need not be the root certificate, it may also be any local certificate.
- C_n is the end entity certificate
- for $C_1, \dots, C_n$ holds that the subject of C_i is the issuer of C_{i+1}

The verification algorithm has to check that each certificate is correctly signed, has not expired and has not been revoked. It must also check for consistency in the name subtrees and process the policy information. If any of these check fails then the end entity certificate cannot be accepted.

2.4 Critical notes on X.509

The main purpose of a certificate and the job of a CA is to bind a public key to the name contained in the certificate. Third parties will believe that some measure of care was taken to ensure that the binding name $\leftrightarrow$ key is valid. The method by which this association was verified, as well as whether the DN really corresponds to a person or an e-mail address is completely outside the scope of X.509.

The CA's policy is specified in the "Certification Practice Statement (CPS)", which is defined by the CA itself. The X.509 says that "a certification authority shall be satisfied of the identity of a user before creating a certificate for it". Therefore, the identity validation procedures are to be satisfied according to the CA's own self-defined rules (CPS). These rules can be completely different for different CAs. The CPS is a governing law that the CA presents to potential clients and represents a top-down framework. The CPS mechanism was considered as a good way to introduce flexibility in X.509 because each CA can have its own rules for different needs, however such a mechanism is a "black hole" and cannot be harmonized for different CAs. This attitude leaves ample room for strong differences between CAs and for the "take-it-or-leave-it" attitude regarding what a CA subscriber can expect. This attitude also cannot be scaled to the global Internet because it is a doubtful that it could always be successfully applied between competing business or different countries. (These problems have actually appeared and are caused by independent interpretations of X.509 in different implementations).

Also, since X.509 certificates are not human readable, a user cannot easily see what is being accepted. In fact he has to take it for granted that it is correct. There is some room for doubt about what exactly an X.509 certificate is and why it is or is not acceptable, thus the X.509 certificates fail to satisfy the most important issue – what has been certified.

There are many other conceptual points in X.509 as discussed in discussion groups or the MCG:

- How the validity of a CA certificate can be verified? The CAs are often self-certified or depend on a CA that is self-certified. In this case, the validity problem is only shifted and the entire chain depends on one final link.

  The GTR [7] (Global Trust Register) group created a list of the most important keys, that are commonly used today. This book is published primarily in a non-electronic way and so helps users to obtain the root keys in a relatively safe way.

- The client software is often forced to accept signatures that are "hardwired" into the software. In such cases the decision of trust is completely removed from the user, which contradicts the rule that the user should always be central in the decision.

- There is also a question how one would distribute an updated top-level CA certificate, when the expired certificate is "hardwired" into the software. Unless there is a second trusted CA which can sign the distribution, the new certificate cannot be certified.

- X.509 is based on X.500 in order to specify the DN naming scheme, but X.509 is not completely defined. It has left room for different readings of the proposed recommendations.

- When using SSL, the clients usually accept the server CA's key signatures in a list that uses indirect (and "soft-trust") properties to link certificates such that the client can accept an unknown CA if that CA is trusted by a CA that the client trusts. This could possibly be a very long chain starting from only one trusted CA. A good certificate list can decrease the number of untrustworthy entries in such a chain. The job of providing a good certification chain is more challenging for the server than for the client. The server is not informed by the client what CAs are preferred (CAs that are directly trusted by the server or untrusted CAs directly trusted by a CA that the client trusts and so on). The client can know which CAs are acceptable to the server as the server sends a list of DNs. No such corresponding list is sent in the other direction from the client to the server.

- The life of a certificate cannot extend beyond the life of the certificate of the signing CA. After the expiration of the CA's certificate one should assume that the corresponding private key may have been cracked or compromised (e.g. discarded without enough care). Anything signed by that key thereafter should be viewed as a forgery. If someone presents a certificate today that was signed by a key, whose certificate expired last week, you have to assume that this is a forgery. If you knew that this certificate was signed during the lifetime of the signing CA's certificate you could assume that it is authentic. The problem is that there is no way to determine from the certificate exactly when it was signed in relation to lifetime.

- The lifetime of a certificate is dependent on various factors. Mathematical analysis and simulation shows that optimal certificate lifetime can be as short a few weeks, while many current commercial CAs use the lifetime of one or more years. It is evident that shorter lifetime will result in higher overheads.

- One must have multiple copies of certificates, due to the use of different non-communicating CAs which have different expiration dates. The certificate must be substituted before it expires, while the older one is still valid and is registered in someone's files somewhere.

- The Certification Authority public key may be the target of an extensive cryptoanalytical attack. For this reason, CAs should use very long keys and change them regularly. On the other side, the top-level CAs are exceptions. It is not practical for them to change keys frequently because their keys may be hard-coded into software and distributed in various ways, and used by a large number of verifiers; Thus those CAs that are most probable target for attacks, offer the least level of protection.

- Certificates usually do not include much information about the subject to whom the certificate is issued. Certificates usually do not certify phone numbers, fax numbers, account numbers or addresses, however such data may be crucial for identification or for establishing a reliable contact. Certificates also do not allow for temporary changes of personnel or other characteristics. Netscape has proposed a new type of certificate that is used together with the X.509 certificate. This new certificate is known as an attribute certificate. Such certificates have no associated key pair and they cannot be used to establish “identity”. Other than this, there is not much difference between a public key certificate and an attribute certificate and thus everything one can include in an attribute certificate could also be included in a public key certificate.

- CRLs are needed to inform that an otherwise valid certificate is not valid anymore. This was first thought to be a positive aspect of relying on CAs (as compared to PGP, for example), but it also creates several unsolvable problems. The frequency of issuing CRLs should be specified in the CA’s self-specified CPS, but there may be considerable delay between the actual need to revoke a certificate and the reflection of this need in a certification chain with *different* CAs (There is no upper limit for such delays). The user is not able to check CA’s certificates in a chain against revocation lists. Some people even claim that CRLs are a solution to a non-existent problem. The problem solved by CRLs is how to communicate that a certificate is no longer valid. If a certificate were really no longer valid, no one would need to check the CRLs to know about it.

- There are basically two approaches how to check the validity of a certificate. The first possibility is based on regular obtaining of the CRLs, as they are published by the CA. When a certificate is presented its serial number is checked against the CRL. If there’s no record in the CRL, the certificate is considered to be valid. The other possibility considers the certificate invalid until the issuing CA confirms the validity of this certificate. This requires on-line access to the CA’s database.

  The latter approach provides for better security. Its benefit depends on the frequency and reason for revoking keys. (The GTR project didn’t register any important key revocation during its this year’s survey, for example).

- When confronted with a risk situation, a normal business solution is to rely on auditing, however auditing of CA's certificates is a difficult or even impossible task. This comes from the X.509 which allows the CA's policies to be built by the CA's itself without any restrictions. The CPSs are indeed different and self-made by each CA and they are not designed to be audited. Phillip Hallan-Baker, a Verisign consultant, publicly noted that "there is not a defined standard for CA practices against which a company might be audited. In effect, each company states their own practices in their CPS. The CPS is not a document designed for auditing use, however. It describes a "specification". It does not describe DETAILS which may be checked by a third party in a systematic manner." The lack of any regulations concerning the CA's policies made a few countries (e.g. Germany and the state of Utah in U.S.) to specify such regulations in their legislative system.

- Another potential problem with CRLs is the risk of a CRL growing to an entirely unacceptable size. In the versions from 1988 or 1993 of the X.509 CRLs each CRL for the end-user certificates needed to cover the complete set of end-users from a CA. Such populations can easily grow into thousands or even hundreds of thousands of end-users. The end-user CRL is therefore at risk of growing to such sizes which might present major communication and storage overhead problem. Since the release of the version 2 of the CRL it is possible to divide the population of end users for one CA into a number of partitions. Each partition is associated with one distribution point. Therefore the maximum size of CRL is controlled by the CA.

  Separate distribution points can also exist for different reasons of revocation. The CRL of revocations resulting from key compromises might be, for example, issued more frequently than the CRL for "routine" revocations.

X.509 is essentially a bag of bytes, meaning and validity of which strongly depends on the CA. Moreover, one may trust the confirmation procedures of the CA during certificate reliance, but one cannot rely upon them for anything more than their value as a representation of the CA's policy expressed (in most cases) in the CA's own terms and rules. Therefore an X.509 certificate needn't necessarily be meaningful or valid for the user's purposes.

Chapter 3

SPKI

3.1 Introduction

SPKI (Simple Public Key Infrastructure) is an Internet draft. It is not yet an Internet standard. Before Internet drafts can become Internet standard, the IESG (Internet Engineering Steering Group) must accept them as proposed standards. To date the IESG has not put this draft among the Internet standards (neither did PKIX). The latest version of this working document was published on March 11, 1998.

The SPKI working group proposes a simple model of PKI. It defines a bare-bones certificate format that eschews all of X.509's complexity, preferring to bind keys to authority and capability rather than identity. The SPKI model is similar to the way in which people use credit cards today. A person's ability to use a credit card is the first step in authorization. When a shop authorizes the use of the credit card, neither the shop nor the credit card company uses identity to authorize such use. The user either has the card or not and the card either has been revoked or not. So the SPKI is based on capability, not identity. The SPKI proposal also eliminates the idea of a hierarchy, proposing a very “flat” architecture in which all certification, certificate retrieval and verification occur in an ad hoc manner. The SPKI is also aligned with the SDSI (Simple Distributed Security Infrastructure) created by Ron Rivest, one of the RSA inventors.

3.2 SDSI

Simple Distributed Security Infrastructure (SDSI – pronounced ”Sudsy”) is a relatively new distributed security infrastructure. It was first published in April 1996 (in version 1.0) by R. Rivest and B. Lampson, along with the work on the SPKI standard. The SDSI was redesigned to go well with the SPKI. SDSI was practically merged with SPKI and the current version of SDSI is 2.0.

SDSI combines a simple public-key infrastructure design with a means of defining groups and issuing group-membership certificates. Its design emphasizes linked local name spaces rather than a hierarchical global name space.

SDSI principals are public digital signature verification keys. These public keys are central in SDSI. Everything is based around them. The notion of an "individual" (a person, process or machine) is not required. Of course, such individuals will actually control the associated private keys, so that public/private keys can be viewed as "proxies" for such individuals.

Each principal is represented by a data structure that can be passed around, such as:

```
( Principal:
  ( Public key:
    ( Algorithm: RSA-with-SHA-1 )
    ( N : =0123456789abcde= )
    ( E : #11 ) ) )
```

A principal can also be the "value" for some name.

Each principal can make statements and requests on the same basis as any other principal. No hierarchical global infrastructure is required. In practice some principals will be more important than others, and SDSI allows for some principals to have special status as "distinguished roots", allowing SDSI to accommodate "global names", but this is for convenience only.

SDSI signatures are quite flexible. Objects may be co-signed by several signers and signatures may contain collections of relevant certificates. Signatures may also have expiration dates and may require periodic reconfirmation.

Certificates can be created and signed by anyone. In other words, everyone can be a "CA". Which policies and procedures a principal follows when issuing certificates is up to that principal to choose. They may declare that some standard procedure must be used or they may issue certificates arbitrarily.

There is no fixed “global” name space giving a unique name for principals. The principal that someone calls Alice may be different from the principal that I call Alice.

SDSI provides means for linking local name spaces. Each principal can “export” his name/value bindings by issuing so called “name/value certificate”. Thus if my local name Alice refers to a principal then I can refer to the principal that Alice calls Bob as

`(ref: alice bob)` or `Alice’s Bob`.

Alice exports her binding by issuing a signed name/value certificate that binds her local name Bob to that particular principal.

It is not necessary to have a symbolic reference as the first argument. One can have an expression such as

`(ref:` *principal* `alice)` or *principal’s* alice,

where *principal* is an explicit principal, however people generally prefer the symbolic form for clarity.

“Groups” are a fundamental part of the SDSI. For purposes of describing who is authorized to access certain data or perform an action, it is usually simplest to define the group of authorized principals in one step and then to place the group name on the appropriate access-control list as a second step. This provides efficiency and reliability when the same group of principals is authorized on many different ACL’s.

A SDSI group is typically a set of principals. Each group has a name and a set of members. The name is local to the principal, who is the owner of the group. The owner is the only one who may change the group definition.

One can define a group by listing its members:

```
(Group: Alice Bob)
```

or by giving algebraic expressions in terms of the other groups or principals:

```
(Group: ( OR: Carol teachers))
```

3.3 Certificates

Certificates in SPKI consist of the following five fields:

- ISSUER: a principal making the certificate's statement
- SUBJECT: the thing about which the statement is being made. This usually is a principal (or a name reducible to a principal).
- DELEGATION: a boolean for propagating authorizations
- AUTHORIZATION: the specific authorizations being delegated in this certificate. Authorizations in SPKI are defined *explicitly*.
- VALIDITY: data ranges and / or on-line validity test to determine the certificate validity

These fields can be expressed as a 5-tuple: (≪issuer≫, ≪subject≫,≪deleg≫, ≪auth≫, ≪validy≫). The basic SPKI certificate is such a 5-tuple, signed by the issuer.

3.4 5-tuple reduction

The process of validating a certification chain is called "5-tuple-reduction". The rules for reduction are simple:
Given $(I_1, S_1, D_1, A_1, V_1)$ and $(I_2, S_2, D_2, A_2, V_2)$ reduction is possible $\Leftrightarrow S_1 = I_2 \wedge\ D_1 = \mathit{true}$. The result 5-tuple is then (I_1, S_2, D_2, A, V) where $A = A_1 \cap A_2$ and $V = V_1 \cap V_2$. (Exact rules for evaluation of the ≪authentication≫ and ≪validity≫ intersections are specified.)

3.5 Authorization flow

There does not need to be any such thing as a "root" key ("handed down from God to be trusted for all purposes"). There is an algorithm running on the user's computer and this algorithm makes the decision. Certificates merely pass authorizations from the ≪issuer≫ to the ≪subject≫, so there needs to be a "left-most" certificate. The only usable structure is an ACL entry. So all useful authorizations flow from an ACL entry through (one or more) certificates, until a 5-tuple of the following form is reached: (self, ≪subject≫, ≪delegation≫, ≪auth≫, ≪val≫). The authorization process then receives this result and acts on it.

3.6 Discussion

PKIX and SPKI are both Internet Drafts in the process to be accepted as Internet standards, however the proposals of the PKIX working group are largely based on the existing X.509 standard for PKI. It has only been formally specified (and extended) for the Internet use by the IETF. SPKI on the other side is relatively new and needs to gain significant support yet. In each case the names behind the SPKI (B. Lampson from the Microsoft and R. Rivest from the RSA) sound promising.

Chapter 4

PGP

4.1 The web of trust

Pretty Good Privacy (PGP) is a crypto software created originally by Phil Zimmerman. The first version of PGP appeared in Internet in 1991 and it has had quite thrilling history since. Nowadays, the PGP has become a standard for privacy in e-mail communication.

PGP in its older versions (up to 2.6) used RSA (for public-key cryptography) and IDEA (for symmetric cryptography). The current situation is more complicated. Since version 5.0 PGP offers 3 algorithms for symmetric cryptography (CAST, IDEA and Triple-DES) and RSA was replaced by Diffie-Hellman algorithm. The international versions of PGP (5.0i, 5.5i and 6.0i), however, use both RSA and Diffie-Hellman algorithms to keep the compatibility with older versions of PGP.

Along with the RSA the PGP system uses MD5 as the message digest algorithm for digital signatures. In the 1996 a German cryptographer Hans Dobbertin showed several weaknesses in the design of MD5 and that why the new versions of PGP use SHA as the message digest algorithm (along with the Diffie-Hellman encryption).

A user of a PGP system generates a (public key, secret key) pair and then associates the public key with his unique ID in the recommended form (name <e-mail address>). Keys are stored in key records. A public / secret key record contains an ID, a public / secret key and a timestamp of when the key was created. Public / secret keys are stored in public / secret “rings”‘. Each user must store and manage a pair of keyrings.

If Alice has a good copy of Bob's public-key record (e.g., a copy which she has reason to be confident has not been tampered with since Bob generated it), then Alice can sign this copy of the public key and pass it to Carol. Alice thus acts as an "introducer" of Bob to Carol. A signed key record is called a key certificate. (Sometimes the word "certify" is used as a synonym for "sign".) The user must tell the PGP system which individuals he trusts as introducers and must certify (sign) the introducer's public-key records with his own secret key. Each individual associated to a public key in the public key ring is assigned his ability to act as an introducer. When a new key is added to the public key ring one of four following attributes is assigned to it:

- *completely trusted* – if any other key is signed by this key, then the new key can be added to the key ring. This means that Alice trusts Bob for the validity of any key.
- *marginally trusted* – a key signed by this key must also be signed by one (or more) other keys. In this case Alice does not trust Bob completely and needs to have his claims about keys confirmed by one or more others.
- *untrusted* – this key cannot be used in determining whether other keys can be added to the keyring. Alice does not trust Bob for validity of any key.
- *unknown* – the level of trust cannot be determined for this key. In practice, this is the same as "untrusted".

The above description bears a great deal of similarity to a description of how two CA operators issuing X.509 certificates would cross-certify each other, or how a CA would issue a certificate for a user. In the PGP case the cross-certification is done at the user level and the assurance of user identity is almost always low or anonymous. In the PGP model, each user is, in effect, its own CA with full authority over how he assigns his trust. This simplicity has allowed PGP to gain relatively widespread acceptance on the Internet compared to other PKIs.

It is, however, important to note that PGP system assumes that the only notion of "security policy" that needs to be supported is the verification of a message sender's ID. It is also necessary to note that Alice's signature on Bob's public key cannot be interpreted such that suggests Alice trusts Bob's personal integrity. Rather, the correct interpretation is that Alice believes that the binding of Bob's "identity" to the key is correct (the problem of binding a public key to an identity was also discussed in the Introduction).

Keyrings and degrees of trust are designed to allow each user to define his own policy of such a very limited form. This narrow notion of policy is appropriate to PGP, which was primarily designed to provide secure e-mail for individuals, but it is insufficient for the broader range of secure network services.

Trust is not transitive. The presumptions that Alice fully trusts Bob as introducer and Bob fully trusts Carol *do not* automatically imply anything about the degree of trust of Alice into Carol. The PGP system respects this important property of trust and does not automatically consider trust to be transitive.

A PGP certificate is not extensible. It usually contains only an e-mail address, an optional photo (since version 6.0), the public key and an attribute indicating the degree of trust; However an e-mail address is by no means an accurate method of identifying someone, thus PGP cannot provide strong authentication of "identity".

The certificate's lack of extensibility prevents PGP from being used for applications beyond the casual e-mail communication. For example, a bank cannot create a PGP bank account certificate for Alice's public key and a user signing Alice's bank key has no way to say that this is the key of the Alice's bank. PGP does not allow a user's trust to be delegated in a discriminating fashion, even if such certificates were possible.

The PGP trust model is supported by a simple infrastructure. There are *Public Key Servers* placed in well defined places on the Internet. These servers respond to requests to add a key or to retrieve a key for a named user. For server operations is (since version 6.0) used the PGP's implementation of TLS (with 128 bit encryption).

These server do nothing else than storing and retrieving keys.

4.2 Revocations

PGP allows for issuance of "revocation certificates". These are a special kind of public key certificate that says that this public key should not be used. A PGP key revocation certificate contains a copy of the public key and is signed by the secret key. When a user obtains such a key revocation certificate, he incorporates it into the keying and this prevents the user from using the public key. The problem of revocation in PGP lies in the distribution of revocation certificates, and therefore in the maintenance of the chain. Adding, deleting or changing data is done by the users themselves and in a happenstance pattern. There is no guarantee if and when the chain is up-to-date and valid. Even if PGP enforces a model of “hard-trust” to setup entries in the web of trust (and correctly recognizes that trust is intransitive), it uses “soft-trust” to upkeep entries without discussing its validity or allowing the time factors. Once a certificate is added to a user’s keying, it is considered valid until the user decides otherwise.

Another problem arises if you want to revoke a key that you no longer have access to. You cannot create a revocation certificate since you cannot control your key anymore. The older versions of PGP recommended to create a key revocation certificate when creating a new key and store it in a safe place for possible future use. This is not very helpful, because it is probable that such a certificate would be lost together with the private key.

Since version 6.0 there is another possibility. When creating a new key one can select one or more other keys for revoking the key. These “designated revokers” will be then able to create the revocation certificate for the lost key.

4.3 Discussion

The PGP approach of trust works quite well in very small groups of users (e.g., small company or department) where one person (or several people) sign the certificates for local staff and the local security policy declares that person(s) to be trusted. In effect, we obtain an organizational CA that signs everyone’s keys and which is trusted by people within the organization.

Scaling the model to larger communities is not easy, because Alice from the company A and Bob from the company B have no common point of trust. What we need now is a cross-certificate, which would be the equivalent of a higher level CA. The PGP system itself supports only a single certificate to be attached to a message and so these certification paths cannot be conveyed with the message.

Chapter 5

SKIP

5.1 Description

Simple Key Management for Internet (SKIP) is key-management scheme for network layer protocols. It is especially suited for use in conjunction with a session-less datagram protocols (e.g. IPv4 or IPv6). SKIP is an open standard and has the status of the IETF draft document.

There are a few ways the authenticated RSA public key can be used to provide authenticity and privacy for a datagram protocol. However such methods require use of a session key establishment protocol prior to communication. This approach has a few drawbacks (related to dynamic routing and crash recovery). Thus, due to the nature of IP network communication the key management scheme must operate in a sessionless and stateless manner.

The packet-encryption key can be encrypted with the recipient's public key and added to each IP packet. However this would mean considerably high overhead caused by transmitting approx. 128-bit key with each packet. The use of authenticated Diffie-Hellman (DH) public values can avoid the need for pseudo-session state management between two parties to establish and change packet encrypting keys.

Each IP source or destination has an authenticated Diffie-Hellman public value. (SKIP supports even multicast IPs) This DH public value can be authenticated in various ways (X.509, PGP and Secure DNS are supported). Thus there is a secret value i and a public value $g^i \mod p$ associated with each IP address. Once n certificates are assigned to n IP nodes,

n^2 mutually authenticated pairwise keys arise. All this simply as a result of the public value authentication process, because each pair of IP nodes can compute the shared secret g^{ij} *mod p*. The symmetric keys derivable from these shared secrets require no setup overhead except for the initial authenticated public value distribution process.

All that is required for each party to compute the pairwise symmetric key is to know the others party's authenticated public key. Since there is nothing secret about the DH public values, one natural way to obtain the needy authenticated public value is to use a directory service (secure DNS, for example).

The shared secret (*gij mod p*) is called the long-term secret and the master-key K_{ij} is derived from this shared secret. K_{ij} is an implicit pairwise shared key, it does not need to be sent in any packet. Thus it can be used as long as desired without any additional overhead.

Since it is desirable to keep K_{ij} for a relatively long period of time, the actual IP data traffic is not encrypted using the key K_{ij}. Instead, only the transient keys (K_p) are encrypted by this long-term key and the transient keys K_p are used to encrypt the data traffic.

Clear IP header	Encrypted K_p	Encrypted IP packet

Both the keys K_{ij} and K_p are used as keys for a symmetric key algorithm. If the source node changes the packet encryption key K_p, the receiving IP node can discover this fact without having to perform a public key operation. Optionally instead of using the authenticated public key infrastructure one can manually distribute the master keys K_{ij}. However this is slow and awkward.

The implicit pairwise master keys can even be used to generate an arbitrary number of implicit master keys by making the master key to be a function of a counter. This counter can be easily constructed in a stateless manner as the number of time units since an agreed-upon start time.

5.2 Discussion

SKIP is an interesting protocol, yet it has a few problems. Each node authenticator derives its information from a type of a directory service. And a user needs a certificate to obtain the other's party DH public value. It is also necessary to note that all the security features that depend on certificates depend also on the data from the application layer. When the certification of SKIP happens at the protocol level, the application program has to complement the certification in a higher layer later, too.

SKIP is transparent to the user. Therefore the user has no practical way to control the process, cannot decide which node authenticator is reliable, can not exclude nodes, which have been affected by enemies, cannot influence the choice of certificates and so on.

The use of SKIP in commercial and other serious situations is difficult, because the decisions are removed from the user, who should play the central role in the certificate acceptance.

Chapter 6

PolicyMaker

6.1 Motivation

Identity-based certificates create an artificial layer of indirection between the information that is certified (which answers the question “who it the holder of this public key?”) and the question that a secure application must answer (that is “can we trust this public key for this purpose?”).

Let’s consider the steps an application must go through to process a request based on a signed message from the holder of a traditional certificate.

obtain certificates and verify the signatures on certificates

$\Downarrow$

verify that certificates are unrevoked

$\Downarrow$

attempt to find a “trust path” to the certificate that is in question

$\Downarrow$

extract names from certificates

$\Downarrow$

lookup names in the database that maps names to actions they are trusted to perform

$\Downarrow$

determine whether the requested action is legal

$\Downarrow$

proceed, if everything is valid

A more general system would integrate the specification of policy with the binding of public keys to the actions they are trusted to perform. PolicyMaker is such a system. It is a trust management system developed by a group of people from the AT&T Laboratories[13], first presented at the IEEE Symposium on Security and Privacy in May 1996.

PolicyMaker binds public keys to predicates that describe the actions they are trusted to sign for, rather than to the names of the key holder.

Considerations such as personal identity and organizational level of the approvers, which are only incidentally relevant to the question the application is trying to answer, can be omitted altogether. This enables to express security credentials and policies without requiring the application to manage a mapping between personal identity and authority.

Simple policies and credentials can be stated simply, and existing PGP or X.509 certificates that merely bind keys to IDs can be used by PolicyMaker with only trivial modifications.

The PolicyMaker system provides a simple language to express conditions under which an individual or an authority is trusted, as well as conditions under which trust may be deferred. PolicyMaker enhances the potential scope and form of security services by implementing trust management in a distinct software system. It frees the designers of services from the need to handle security completely within applications.

PolicyMaker evaluates proposed actions by interpreting the policy statements and credentials. Depending on the credentials and form of the query, it can return either a simple yes/no answer or additional restrictions that would make proposed action acceptable.

Security policies and credentials are defined in terms of predicates, called *filters*, that are associated with public keys. Filters accept or reject action descriptions based on what the holders of the corresponding secret keys are authorized to do. Security policies and credentials consist of a binding between a filter and one or more public keys.

A local policy may trust third parties to issue credentials for others, and it is possible to use filters that limit the extent to which these third parties are trusted. Credentials themselves may also contain filters that limit the actions their holder is trusted to perform. An action is considered acceptable according to local policy if there is a “chain” from the policy to the key(s) requesting the action in which all the filters along the chain are satisfied.

6.2 The PolicyMaker Language

The basic function of a PolicyMaker system is to process queries. A *query* is a request to determine whether a particular public key (or a sequence of public keys) is permitted to perform a particular action according to local policy. Queries are of the form

$\texttt{key}_1\texttt{, key}_2\texttt{, ..., key}_n\ \texttt{REQUESTS ActionString}$

The semantics of action strings are determined by the applications that generate and interpret them.

PolicyMaker processes queries based on trust information contained in assertions. *Assertions* confer authority on keys. Each assertion binds a predicate, called a filter, to a sequence of public keys, called an authority structure. Assertions are of the form:

```
Source ASSERTS AuthorityStruct WHERE Filter
```

Source indicates the source of the assertion (either the local policy or the public key of a third party). AuthorityStruct specifies the public key or keys to whom the assertion applies. Filter is the predicate that action strings must satisfy for the assertion to hold. Each assertion states that the assertion source trusts the public keys in the authority structure to be associated with action strings that satisfy the filter.

There are two types of PolicyMaker Assertions: *certificates* and *policies*. A certificate is a signed message that binds a particular authority structure to a filter. A policy also binds a particular authority structure to a filter. Policies, however, are not signed. Because they originate locally, they are unconditionally accepted locally.

Set of local policies forms the “trust root” of the machine and defines the context under which all queries are evaluated. A query is a request for information about the trust that can be placed in a particular (sequence of) public key(s).

We may interpret the assertions as a directed graph D in which the vertices are labeled by keys or policy sources and the arcs are labeled by filters. If $v \xrightarrow{f} w$ is an arc in D that is labeled by f, then there must be an assertion whose source is the label of v, whose authority structure is the label of w, and whose filter is f.

To process a query, the PolicyMaker system must find a chain $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_x$ in D which v_1 is a local policy source and $v_x = k$. If the query contains multiple keys $k_1, k_2, \ldots, k_n$ and the assertions contain complex authority structures, then V(D) must include nodes that are labeled by keys, policy sources or complex authority structures, and the chain $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_x$ must be such that v_x is labeled by an authority structure that accepts the input $(k_1, \ldots, k_n)$.

The filters in certificate and policy assertions may take one of two forms. The simplest form is a program that accepts or rejects action strings. The second filter form not only accepts or rejects action strings, but may also append annotations to an otherwise acceptable action string that indicates restrictions.

PolicyMaker itself does not verify signatures on signed assertions or queries. Signatures are verified by some external program or function (e.g., PGP, PEM).

6.3 Discussion

The PolicyMaker approach has a few advantages compared with the traditional trust management approaches:

- certificates and policies are based on predicates written in a general programming language;
- trust descriptions can be changed without altering the trust management system;
- risks arising from one level of indirection (mapping of identities to their authority) are eliminated;
- it requires designers of secure systems to consider trust management explicitly.

Of course, PolicyMaker does not solve the entire trust management problem neither does it guarantee that systems which use it will be secure. Applications *must* define action description languages that accurately reflect the security semantics of the application.

The policy predicates and certificate assertions must be carefully written to reflect the *intentions* of the policy. There are practically no restrictions on predicates, so it is possible to construct policies that have unfortunate or unexpected consequences.

Chapter 7

Conclusions

The basic purpose of the certification is to verify that an attribute (such as a public key) really belongs to an entity. Assured knowledge of the other party's public key is needed for secure communication between two previously unknown parties.

The problem of certification is often discussed nowadays. It is rapidly developing, two new Internet drafts were presented this year (1998).

There are many standards and so-called standards for public key infrastructure. They are based on various attitudes to trust and trust management. The best known standard is the ITU-T's X.509. It is based on Certification Authorities, that bind distinguished names to public keys. X.509 is a recommendation which many implementations are derived from (e.g. SSL, PKIX). The most important fact that is reprehended to X.509 is the Certification Practice Statement of CAs. It is a self-created rule for all a CA does. The rule can be different for different CAs and is not designed to be audited. On the other side, out of all the systems offered today, the systems based on X.509 are the most acceptable for business purposes (e.g. SET is derived from X.509).

PGP is based on "introducers" of public keys. PGP is very suitable for small communities that want to exchange e-mail messages, but is not extensible for larger communities or other purposes.

SPKI is an Internet draft that prefers to bind keys to authorities rather than identities. It uses the flat (non-hierarchical) architecture called SDSI (Simple Distribution Security Infrastructure) created by Ron Rivest and Butler Lampson. The SPKI has not gained a widespread support by now, but the names behind this standard might indicate that this will soon change.

The PolicyMaker system enables the separation of security services from applications and process them in a distinct system. PolicyMaker processes the applications' queries and returns either simple yes / no answer or additional restrictions for acceptable actions. Processing queries is based on trust information contained in assertions supplied by applications. The separation of the security services brings a few advantages, but does not automatically solve the entire trust management problem. The application must supply the PolicyMaker system with correct policy predicates and certificate assertions, otherwise, PolicyMaker would not be of any help.

References

[1] S. Garfinkel: *PGP, encryption for everyone.*
O'Reilly & Associates, Inc., Sebastopol, 1995

[2] A. Aziz: *Simple Key-Management for Internet Protocols*
`http://www.skip.org/inet-95.html`

[3] A. Aziz et al.: *Simple Key-Management for Internet Protocols*
`http://skip.incog.com/spec/SKIP.html`

[4] E. Gerck: *Overview of Certification Systems: X.509, CA, PGP and SKIP*
`http://www.mcg.org.br/cert.htm`

[5] M. Branchaud: *A Survey of Public Key Infrastructures: Pretty Good Privacy*
`http://www.xcert.com/~marcnarc/PKI/thesis/pgp.html`

[6] C. M Ellison et al.: *SPKI Certificate Theory*
`http://csro.nist.gov/pki/`

[7] R. Anderson et al.: *Global Trust Register*
`http://www.cl.cam.ac.uk/Research/Security/Trust-Register/`

[8] E. Gerck: *Towards a Real-World Model of Trust: Process and Social Reliance on Received Information*
`http://www.mcg.org.br/trustdef.htm`

[9] N. Bohm: *Authentication, Reliability and Risks*
`http://www.mcg.org.br/auth_b1.htm`

[10] E. Gerck: *The Meta-Certificate Standard FAQ*
`http://www.mcg.org.br/mcfaq.htm`

[11] F. J. Hirsch: *Introducing SSL and Certificates using SSLeay*
`http://www.camb.opengroup.org/www/prism/wwwj/`

[12] R. Housley: *Internet Public Key Infrastructure X.509 Certificate and CRL Profile*
`http://csro.nist.gov/pki/draft-ieft-pkix-ipki-part1-07.txt`

[13] Matt Blaze et al.: *Decentralized Trust Management*
`http://dimacs.rutgers.edu/TechnicalReports/`
`abstracts/1996/96-17.html`

[14] A. Young: *Technologies to Support Authentication in Higher Education*
`http://www.ukoln.ac.uk/services/elib/papers/other/scoping/`

[15] C. Ellison: *Generalized Certificates*
`http://ftp.clark.net/pub/cme/html/cert.html`

[16] R. Macgregor: *SET Certification*
`http://www.redbooks.ibm.com/redbooks/SG244978/setbk24.htm`

[17] J. Lewis: *Public Key Infrastructure Architecture*
`http://www.tbg.com/samples/netsvcs/pkiarc.htm`