IP Authentication Framework over PKI

Chiache Tsai, Xiaofeng Qiu, Henning Schulzrinne
Internet Real Time Lab, Columbia University

1 Abstract

To provide an IP authentication mechanism for internal messaging in a network, we apply public key infrastructure (PKI) to build up the trust model. The problems we are solving is to securely sign and verify messages using asynchronous key pairs and to dynamically manage certificates bound to IP addresses. We try to involve Dynamic Host Configuration Protocol (DHCP) in the architecture, to make the certificates being deployed immediately after IP address asignment. Our model of the certificate management contains (1) a prover that proves the ownership of IP addresses to the certificate authority (CA), (2) a signer that signs the messages using the private key and (3) a verifier that verify the certificates related with the arriving mesages. This model provides a solution to the existing routing security issues, and works as a generic end-to-end message signing mechanism.

2 Introduction

In computer networks, IP addresses are usually used as identifications of hosts. For example, packet filters will filter the IP addresses of senders or receivers for applying the routing policy. In many web services, clients have to trust applications published by issuers. The way of identifying application issuers is often IP addresses or DNS translations of IP addresses, which makes IP addresses very sensitive information in computer network.

In order to quarantee the authenticity of IP address, we need a framework that provides methods to proof the IP address ownership of each individual hosts. Moreover, when a new host joins the network, the authentication information of the host must be dynamically updated. In order to build such a trust model, we applies *Public Key Infrastructure (PKI)* to feature this security interaction. In *Public Key Infrastructure*, entities are given unique asynchronous key pairs, to proof the integrity of signaling by signing the messages using the self-hold private key. The receiptors of messages may verify the integrity of messages by the public key either attached to the message or found in the directories of the network. For enforcing the security of cryptographic operations, PKI must maintain proper management on the deployed key pairs, in the form of protected documents of certificates.

One of the main issues of IP address authentication is routing security. Internet protocol is a protocol of network layor, which is much lower than application layor where most people intend to enforce the security. A DHCP server does not own a trust model with certificate authority, so there is no strong model to bind certificate deployment on IP address assignment. It will be extremely vulnerable if the certificate authority simply trust all certificate request, since IP address are easy to forge. Our solutions to the issue is to build a trust model among IP address assignment and certificate deployment. The simple one is to embed certificate request in DHCP request, and thus prevent other user in the network intercepts the deployment of certificates. Another solution, which leaves least modification to DHCP, is to embed a piece of authentication information, which can later be referenced by the certificate authority when the user issues certificate requests.

We start this project as a part of NetServ Project. NetServ is an innovative framework to fulfill new specification on network routing, with a new signaling model to carry code segments among routers. The architecture of NetServ has a security issue: Routers must trust the issuers of the arriving code segments. A bogus issuer may compromise a router with backdoored code. A user may also elevate the privilege by replaying a signal sent by a privileged user. We want to design a subsystem to bind the hosts with IP identification and to provide security for the source address of routing signals. The core concept of this project largely depends on the requirement of the Net-

Serv architecture, and we intend to state the security problems that might happens in the architecture and provide solutions to those problems. Somehow we are also intend to make our result to be general solution that can used to improve security in other network that requires address-based authentication.

3 Background

In this chapter, we will introduce the background knowledge that a user or developer must know about this framework.

3.1 Public Key Infrastructure (PKI)

Public key infrastructure (PKI) is an aggregation of systems that provides a reliable management and usage of cryptographic objects. In our framework, we hold a network where each entity owns a unique pair of asymmetric key (a public key and a private key) to perform cryptographic operations. Public Key Infrastructure provides mechanisms for revealing the public key of a entity to the network. The ability of signing messages using the private key can prove the authentication of senders, and everyone in the network can verify the identification using the revealed public keys.

In public key infrastructures, key pairs are be used in several ways:

- Encryption of digital messages, such as E-mail messages, sensitive information, privacy data.
- Authentication of the sender and receiver of the messages.
- Prove the integrity of the content or the attributes of the messages.
- Authentication of users to applications.
- Bootstrapping secure communication protocols, such as Internet key exchange (IKE) and SSL.

In common, public keys are stored and conveyed in the form of digital certificates, which is a piece of data being protected by some party that both the sender and receiptor trust. It is critical to have a trust model between the sender, receiptor and the certificate issuer to maintain the security of issued certificates. One of the common trust models is certificate authorities (CA). Certificate authorities are services provided by trustworthy servers that sign and issue certificates. A CA will maintain a directory of all the certificates it issues, and maintains the proper usage in the life-time of each certificate. In the case that a certificate is being abused or compromised, CA must have mechanisms to revoke the certificates and prevent them to be reused. Figure 1 shows the architecture of PKI.

Figure 1: Basic Structure of Certificate Authorities

In some case, an optional registration authority (RA) is required as a role to authenticate the incoming request and to manage the registration of user entities. A Registration Authority will be placed int the middle of the certificate authority and the network, accepting all request and forward only legal request to certificate authority. There must be a trust relationship between certificate authority and registration authority. From the view of the network, registration authority should be transparent and considered as one entity together with the certitifcate authority.

In this project, a hierarchical CA framework is used to implement the public key infrastructure. The reason that we apply public key infrastructure on IP authentication is that we need a mechanism to deploy and convey the certificates, and to ensure srelaiability and usability of key pairs. One of the main purposes of the solution is to provide a generic solution to IP authentication, so we must not try to define the clients in the network. We must not assume that each entity in the network trust each other. Obviously, a CA hierarchy should be useful for providing the trust model we want, and it is reasonable to assume that the client at least trust one of the root CAs.

3.2 X509 Certificates

X.509 standard is first issued in 1988, as an extension of X.500 standard. It is a standard format for describing the format of public key certificates, certificate revocation lists, and a certification path validation algorithm. X.509 strictly defines certificates under a hierarchical structure: a X.509 certificate has to be either self-signed (which means it is the root CA) or signed by some higher-level CA. To do the verification, it is neccesary to go through the whole signing chain to verify the certificates.

The first version of X.509 standard defines a standard certificate format. In the format, each certificate has a name to identify itself. Such a name is called *Domain Name* or *Subject*. A domain name or subject consists of different identifications, such as common names, organization names, company names or locations. Note that this subject might not be unique among all the certificates signed by the same CA, and there is a unique serial number to identify the certificates. A certificate must contain issuer information, algorithm and signature for verification.

X.509 version 2 formats is published in 1993, and version 3 formats is later published in 1996. The primary improvement of the X509 version 3 formats is a set of certificate extensions that can carry additional information for the users.

3.3 Enterprise Java Bean Certificate Authority (EJBCA)

EJBCA is one of the enterprise class certificate authority applications. It is built on J2EE platform, together with Java Bean (EJB) framework, which is a popular component framework implemented in JAVA language. Enterprise Java Bean framework provides a methodology to build an internet service with centralized server and back-end database. Those enterprise Java Bean applications will be run on certain EJB servers, such as JBoss server, Glass Fish, WebLogics or OC4J. EJBCA can be run on a recent version of all these servers.

SQL databases are used as a back-end database of the EJBCA. For each certificate issues by the EJBCA server, there is a correspondent user created, with a unique user name. The type of users will tell the purpose that this certificate is being used and the profile template that the user is generated. Three Basic types of user are SERVER, END ENTITY and SUBCA.

The architecture of EJBCA can be either an all-in-one CA/RA server or a standalone of CA server with some external RA server. In the all-in-one mode, EJBCA only accept certificate request from a user that has previously registered to the user database. Also the status of this user has to be new, which means no certificate has been generated for this user before. If configure the EJBCA with an external RA, user can be generated for any legal certificate request, and EJBCA will let the external RA responsible for the management of users.

Several certificate management standards for PKI are provided by EJBCA. EJBCA uses digital certificate, certificate request, certificate revocation list (CRL) of X.509 standard. It supports RSA key algorithm up to 4096 bits, DSA key algorithm with 1024 bits, and ECDSA key. For certificate status check, EJBCA support both Certificate Revocation List (CRL) and Online Certificate Status Protocol (OCSP). CRL and OCSP provides different mechanism to check revoked certificates, for CRL provides a list of revoked certificates and OCSP responds to the client with the status of certificates. The two mechanisms can be useful in different situations. Note that EJBCA can also be used as a standalone OCSP server that caches certificates status information from external CAs and responds to clients.

EJBCA is also one of the few open-source solutions that support both Certificate Management Protocol (CMP) and Simple Certificate Enrollment Protocol (SCEP). The two protocols are both for batch generation of external certificate request. The former, CMP, provides a more general-use management protocol for all kinds of certificate management operations, while SCEP provides only a simple messaging protocol for certificate generation. In this project, we need CMP as an interface of server for accepting certificate requests and revocation request from clients.

For futher guideness to set up and configure an EJBCA server, pease read *Appendix I. A Quick Manual for setting up EJBCA.*

3.4 Abstract Syntax Notation One (ASN.1)

In computer networking and cryptographic methods, a uniform data format is required to store and transfer information among entities. The Abstract Syntax Notation One (ASN.1) provides a standardized and flexible data structure to describe objects. ANS.1 Objects can be encoded into bit string, transferred on the network or stored on the media, and decoded back into object.

There are six kinds of encoding rule in ASN.1 standard:

- Basic Encoding Rules (BER)
- Canonical Encoding Rules (CER)
- Distinguished Encoding Rules (DER)
- XML Encoding Rules (XER)
- Packed Encoding Rules (PER)
- Generic String Encoding Rules (GSER)

Especially Distinguished Encoded Rules (DER) is widely used in X.509 standard. And we are also using DER encoding to encode the objects we defined in this solution.

3.5 OpenSSL, Java Crypto Utility and Bouncy Castle

The OpenSSL project is a set of utilities that provide representation of cryptographic objects and methods. Although the main purpose of OpenSSL is to support development of applications with SSL/TLS, it can also be used in public key/secret key encryptions, certificate generation and management, digital signatures and authentication codes. The current version of OpenSSL is 0.9.8, and it is written in C languages, expectedly used on Linux and BSD platforms.

For developer that uses Java and C# language, Bouncy Castle project provides porting of OpenSSL API to Java and C# platform. Bouncy Castle defines a set of cryptographic objects in ASN.1 format and provides methods to encode objects in DER or BER encoding rule. Bouncy Castle also ports the native ASN1 parser engine of the OpenSSL project, so developer can easily define new ASN1 objects in Java or C# language.

In fact, Java Development Kit 1.2 or later version also defines cryptographic utilities for developer of security applications. In the native cryptographic utilities of Java, the interfaces and prototype of cryptographic objects and methods are defined abstractly, and can be implemented by any providers.

3.6 Certificate Management Protocol (CMP)

Certificate Management Protocol (CMP) is an internet protocol which supports remote certificate operations on CAs. The functionality of Certificate Management Protocol includes request and renewal of X.509 certificates, update and recovery of public keys, proof of private key possession, certificate revocation, cross operation among multiple CAs, and so on. Certificate Management Protocol provides a well-formed communication channel between end entities and RAs, RAs and CAs, CAs and end entities. Currently only business solutions and a few open-source solutions, such as EJBCA, support Certificate Management Protocol. Most of the interface of CMP is built upon HTTP protocol, while some solution might provide TCP connection for CMP responders.

The format of the CMP messages are defined in Abstract Syntax Notation One (ASN.1 Format).The protocol is officially defined in RFC4210, and a primary part of the CMP messages uses CRMF (Certificate Request Message Format) described in RFC4211. An obsolete version of CMP is described in RFC 2510, the respective CRMF version in RFC 2511.

In RFC 4211, the format of CMP messages is described as following.

```
PKIMessage ::= SEQUENCE {
   header              PKIHeader,
   body                PKIBody,
   protection   [0]    PKIProtection OPTIONAL,
   extraCerts   [1]    SEQUENCE SIZE (1..MAX) OF
                       X509CertificateStructure OPTIONAL
}
```

PKIMessage has two main parts, PKIHeader and PKIBody. Note that CMP provides two way of the protection on the integrity of the message. One of the protection methods is Password-based Protection. In the Password-based Protection method, client and server share a secret key and the key can be used to create the Message Authentication Code (MAC) for ensuring the integrity. Another method is Certificate-based Protection. In the Certificate-based Protection method, client and server own their certificates and they use the private keys only known by themselves to generate the signatures as the protection. Only in the Certificate-base Protection method, extra certificates have to be attached to the message.

The PKIHeader contains information to identify the sender and the receiver of the message, the way to verify the protection and some useful information. The PKIBody provides up to 26 kinds of messages types to serve different kind of certificate operations.

In order to prevent eavesdroppers to send bogus CMP request by making replay attack, the server might ask the client to prove that it really owns the private key that correspond to the public key included in the certificate request. Such a mechanism is called proof-of-possession (POP). Since only the owner of the private key can produce the segment of proof-of-possession, the security can be ensured on the certificate requests. Several kinds of proof-of-possession protection can be used in CMP, and one of the simplest ones is to create signatures of the message. Note that this signature is different from the certificate-based protection mentioned before. The Proof-of-Possession Protection is made on each single certificate requests (A CMP message can contain multiple requests), and its purpose is to prove the ownership of the key instead of protecting the message. By using signatures generated by the clients private key, the server can prove the possession by simply verify the signature using the clients public key. Another proof-of-possession protection will be decryption key challenge.

Certificate management protocol is very critical in this solution. Since the complexity of certificate operations performed on the CAs, only Certificate Management Protocol provides all the functionalities we want in this project. The two basic message types required here are Initial Request (Certificate Request as well) and Revocation Request. The CMP interfaces of CA/RA servers have to serve the requests from trustworthy clients and create/revoke the correspondent certificates as they wishes. By following the definition of CMP, with authentication and proof-of-possession, the request from those clients can be trusted as secure.

To be more specific, the CA/CMP solution used in this project must satisfy the following requirement:

1. The client must be able to register new users in the user databases. In order to provide dynamic certificate deployment for end entities, CAs must provides user generation when the certificate request sent by the client does not match with any existing user in the user databases. Sometimes such a case is called RA mode. It means that

the CAs trust the sender of certificate request as a RA, and accept whatever valid request for the certificate generations. Somehow in such a RA mode, an additional RA is required to invoke the management and authentication upon external requests.

2. The CA/RA must authenticate the sender of CMP requests. Since the CA/RA has to trust the end entity that requests for certificate generations, there must be some authentication on the CMP messages. In fact, not only the client sending certificate requests must be authenticated, the CA/RA server that providing the CMP interface also has to be authenticated in order to prevent rogue servers or man-in-the-middle attack. As described before, there are two kinds of authentication methods for CMP messages: Password-based Authentication and Certificate-based Authentication. Both of the methods are usable in the scheme, as long as a CA/RA maintains a database for trusted passwords and certificates. Somehow some CA solutions, such as EJBCA, only provide a shared-secret authentication for RA mode, which can be extremely insecure. For those CA solutions that dont support individual authentication methods, there is definitely a necessity for building a correspondent RA.

3. The CA/RA must deal with the collision of public keys and domain names in certificate requests. In this project, one of the assumptions is that the network that an organization owns might not have its own CA/RA server. It is allowed that the certificates for this network are issued by an external trustworthy CA/RA server. As a result, there exists possibility for collision in the domain names that the clients request for. The collision can happen especially when we use the IP addresses as the Common Name for recognizing the certificates. Since the quantity of certificate requests that a CA/RA server receives might be huge, there is also a possibility that two clients use the same public key. As a solution, the CA/RA has to accept CMP requests with identical domain names or public key, and try to prevent confusion during the certificate management.

For developers, currently there are only a few implementations of CMP API suites. An unofficial extension of OpenSSL provides the definition and methods to build CMP messages in C language. For Bouncy Castle project, a Novosec extension implements a set of API supporting CMP, and it is also used in EJBCA. Both of the solutions is used in this project, for the CMP extension of OpenSSL is used in C programming, and Novesec extension for Bouncy Castle is used in a large proportion of Java programming.

3.7 Certificate Revocation List (CRL)

A certificate revocation list is a list that a Certificate Authority issues, of certificates that are revoked or no longer valid so no any entity should reuse them. The concept of certificate revocation list was defined together with X.509 certificate in RFC3280. In fact, a CRL is usually a list of the serial number of the revoked certificates instead of the full contents of certificates.

Certificate revocation lists are generated periodically by the issuers. Two attributes in the CRL can tell the validity of the CRL Object: Last Update specifies when the CRL object should be used, and Next Update tells when the CRL object should no longer be used. The period of CRL generation should be short enough to prevent a revoked certificate being abused before it is listed in the CRL. Although that we cannot renew the CRL timely after the revocation of a certificates, as long as the update period is short enough, we dont really worry about the effect of a revoked certificate in the cryptosystems. In principal, we should not absolutely depend on CRL to decide the validity of a certificate, and a reasonable expiring time should be given in case that CRL service is not available.

The largest threat of certificate revocation list will be denial-of-service attack. Since the revocation list is updated to clients passively, a denial-of-service will make the client unable to verify the certificates, or even make the revoked certificates pass through the verification. The denial-of-service attack can happens in both sides: in the server side, make CRL unable to be updated or retrieved, or in the client side, make the client unable to obtain the CRL. In some cases, an alternative of the CRL called online certificate status protocol (OCSP) might be more useful.

In the revocation list, there are two kinds of status the certificates might be given, one is *Revoked* and another is *Hold*. A revoked certificate is irreversibly revoked, so any client should no longer reuse this certificate. But a hold certificate is just a temporarily invalidated certificate, which might be reused later for some reason. When the issuer claimed that a certificate is revoked, it might also want to specify the reason why the certificate should not be used. There are 10 predefined reasons in CRL that can be given by the issuers.

Sometime it is a waste of bandwidth if we periodically download the full Revocation List. Instead, a delta CRL can be generated by the issuer to improve efficiency. A delta CRL includes only the difference of the current CRL with the previous CRL, so largely reduce the size of the CRL Object. The clients only have to download the full content of CRL at the beginning

and after the service resumes. Note that a delta CRL also has a denial-of-service problem. Temporary unavailability to the delta CRL will cause inconsistence and affect the reliability of the CRL.

4 Concept

4.1 Dynamic Certificate Deployment

One of the problems that we are challenging is how to deploy certificates to authenticate the IP addresses in a dynamically allocated network. Suppose we have a DHCP server that assigns a range of addresses to all the hosts in the local network. Assuming that all the hosts obtain the IP addresses legally, we need a mechanism to issue certificates to bind on those IP addresses and let the host can sign messages with the correspondent private key. Therefore the receivers of those messages can verify the signatures and prove the integrity of source IP addresses.

Such a mechanism is especially useful in a network where a huge amount of addresses are dynamically allocated and users of the network cannot be explicitly defined. Since the users of the network cannot get the certificate before obtaining the IP address, there is a difficulty to build a web of trust that *Secure BGP* does. We try to provide a solution that all IP address owners in the network can obtain a legally signed short-term certificate to authenticate themselves, with no need to register preliminarily.

The main challenge of this problem is how we can trust the hosts in the local network to request certificates. After the IP addresses being assigned to hosts, we can no longer prove the ownership of the IP addresses. Since an attacker or eavesdropper in the local network can always spoof the source address by manipulating the header of an IP packet, there is no secure way to authenticate the clients given the IP addresses. In order to solve this problem, we try to involve DHCP server in the process of certificate deployment. The involvement of DHCP server gives us a new advantage on trusting the certificate request from local clients, and stops the attackers from trying to spoof an IP address that they actually didnt own. During the assignment of an IP address, a piece of identity information can be given by the legal obtainer of this address, and since such an identity information cannot be forged or reused by any attacker, the security is maintained in the process of certificate deployment.

We have two solutions that both involve DHCP services in the certificate request procedure. One of the solutions has a direct involvement to the DHCP services, and we modify DHCP messages to carry the cryptographic objects. Another solution leaves the least change to DHCP messages, but use some attribute of DHCP messages to carry authentication information. The comparation between the two solutions will be discussed in later chapter.

4.1.1 Solution 1: Certificate Request and Response within DHCP Messages

This solution involves a direct modification to the DHCP message format. It has been well defined in DHCP standard that an IP request procedure consists of four DHCP messages between clients and DHCP server. All the messages are sent in the form of UDP packets and mostly broadcast to the local network. The DHCP server has to remain stateful for each single request, and keep all the information (client ID, physic address, requesting IP address, etc) for unfinished request. Note that client ID is a randomly generated 32bit integer which is used to identify the request.

The DHCP messaging for requesting consists of four phases:

- Discovering: A DHCP client will broadcast *DHCPDISCOVER* message to find local DHCP servers. Sometime a client might specific a previously allocated address in the DHCPDISCOVER message, for requesting reuse of the address.
- Offering: The DHCP server will respond to *DHCPDISCOVER* message with a pre-allocated address.
- Requesting: If the DHCP client to accept the offer of the address from a certain server, the client sends out a *DHCPREQUEST* to request this address.
- Acknowledging: If the requested address is still available to the client, the DHCP server will acknowledge the client with *DHCPACK* message.

We need to place certificate request and retrieval information within DHCP request. The basic procedure is as following:

1. The client generates a key pair consist of a private key and a public key.
2. The client sends the public key to the DHCP server by encapsulating it in the DHCP message.
3. The server will generate or ask some Certificate Authority to generate the certificate, and return to the client in another DHCP message.

The certificate object should only be put in DHCPACK messages. It should not be put in DHCPOFFER messages since the address being offered to the client might not be the address eventually being assigned. However, from the client side, the

public key can be put in either DHCPDISCOVER or DHCPREQUEST messages. Technically we prefer to put the public key in the DHCPDISCOVER messages than DHCPREQUEST messages, since the public key can be used in some other way to protect sensitive information during the DHCP request.

There might be a race between the legal user and some attacker trying to steal the address identity. Since an attacker can sniff on the local network, the requested IP address is somehow revealed to the attacker before it is actually assigned. The result is, the attacker can send out a bogus DHCPREQUEST messages with its own public key. Such a attack can trick the DHCP server to issue the certificate that actually contains a counterfeit public key. The attack also causes a denial of service on the legal user, since the signature that a legal user generated can no longer be verified.

One useful technique to stop attacker from sending bogus request is to shadow the IP address in DHCPOFFER messages. No attacker can get the requested IP address before the last acknowledge and try to race with legal clients to obtain certificates. Instead the request IP address can be encrypted by the clients public key. Since only the client owning the correspondent private key can decrypt the information, we dont have to worry that attackers might reveal the requested address and do malicious things.

However, such a mechanism does not really stop all attackers to send out bogus request. Since the range of the IP addresses in a C-class local network is usually of size smaller than 256, it is not difficult for an attacker to send out 256 DHCPREQUEST and try to obtain the certificate bruteforcedly. Things get worse when the DHCP server assigns the IP address in sequence. An attacker can easily predict the next address by eavesdropping all the DHCP requests in the local network. In fact this problem is a common security problem for DHCP service and there is no solution but some detection technique we can use to find out attacker in the local network.

Sometimes we may want to shadow the IP addresses not only in DHCPOFFER messages but also DHCPREQUEST messages. There is a concern that an attacker might still eavesdrop on DHCPREQUEST messages and try to race with legal users. As a result, not only the client has to provide its public key to the server, but server has to provide its public key to clients. The public key of the DHCP server can be carried by DHCPOFFER messages and used on the encryption of IP address in DHCPREQUEST messages.

In order to carry cryptographic objects in DHCP messages, we create a new option temporarily numbered as 254. Just like mentioned before, the YIADDR (Your Address) fields have to be wiped out as 0s, and also Option 50 (Requested IP address) must be removed in the first three messages. The cryptographic objects that the messages carrying will be as follow:

- Discovering: Client generates a key pair and sends the public key to the server.
- Offering: Server sends its public key to client, with the offered IP address encrypted with clients public key.
- Requesting: Client sends the requesting IP address encrypted with servers public key. Client may also want to update its public key.
- Acknowledging: Server returns the certificate.

The Key Pair should only be generated by the client. Since there is no previously negotiated key between server and client, it is extremely insecure to let the server generate the Key Pair and conveyed to the client. However, there are some drawbacks for letting clients to generate their key pairs. First, there might be a collision, occasionally or deliberately, among the public keys that different clients generated. Second, the server cannot prove the possession of the private keys in the CMP requests. For solving those two problems, the client should be responsible for providing a unique public key and a proof-of-possession segment to the server. The server will validate the information given by the client and deny those requests that didnt give reasonable parameters.

One of the concerns for putting cryptographic objects into DHCP messages is the size of the objects. The size of a certificate can be as large as several hundred bytes, and make the DHCP messages too large to send. For some small router, there might be a limitation on the size of the DHCP messages, and therefore truncate the messages to a fixed length. This problem is just a technical problem that can be solved by configuring the network properly. However, if we dont want the certificate to make the DHCP messages too large, there are two techniques that we can use in the implementation: One is create another channel to deploy the certificates. We can either broadcast the certificate to the network, or put the certificate on some distribute point that can be retrieved by clients later. Another way is to cut the certificates into several pieces, and carry them in multiple DHCPACK messages.

In this project, we extended the open-source UDHCP application, which is a light-weighted DHCP solution for both client side and server side. The implementation is on the both side, for removing plaintext IP address information and creating a new option to carry public keys and certificates. The server used a CMP client in the local host to communicate with an external CA to obtain the certificate.

4.1.2 Solution 2: Certificate Request and Response without DHCP Messages

The second challenge of the dynamic certificate deployment problem is to build a certificate deployment mechanism without changing the structure of DHCP messages. We want to prevent direct modification to the DHCP standard, and build a trustworthy channel for the IP address owner to obtain their certificates. An ideal communication model is like the following.

Phase 1 The client sends some preliminary messages.

Phase 2 The client obtains IP address by DHCP request.

Phase 3 The client sends request to a local server for the issue of new certificate.

Phase 4 The client gets the certificate.

As described in the previous paragraph, it is necessary to involve the DHCP server in the procedure. In order to make sure that the client doesnt spoof its IP address while requesting the certificate, we need some new advantage for the legal users to authenticate against attackers. We need two assumptions on the DHCP solution to provide the security.

1. DHCPREQUEST must carry some authentication information that can only be produced by a legal client. For example, the result of a one-way function that can only be computed by the client itself.

2. After the offering of an IP address being acknowledged to the client, the authentication information must be kept in the lease database that some other application can retrieve on the local host.

Make an example, we can make client ID (XID) as a digest of the public key to be the authentication information. Since the digest function is a one-way function that cannot be reversed by eavesdroppers, no attacker can request the certificates with some other public key. The certificate request will be served or forwarded by a daemon that runs on the same host of the DHCP server and has the power to query the lease databases. When such a daemon receives a certificate requests with some public key, it will check if the digest of the public key matches with the authentication information in the lease database. If the daemon determines that the requests cannot be proved as sent by the real owner of the IP address and the public key, it will deny the request for a new certificate.

We do not worry about replay attack in this model. An attacker that tries to send out a certificate request which was sent previously by some other legal client

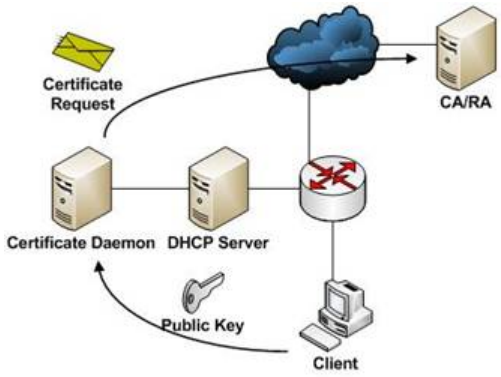

Figure 2: Architecture for Dynamic Deployment Model without DHCP messages

can only obtain the same certificate that was given to this client before. Since the attacker doesnt own the correspondent private key, it cannot spoof its IP address by signing its payload with this certificate. Since such a model guarantees that the relationship of IP address and the public key cannot be forged by any attacker, any local machine that tries to bind some public key to an allocated IP address will be denied by the certificate daemon.

For doing the implementation, a simple model is to make the certificate daemon as a proxy of certificate request. When the daemon determine a certificate request to be legal, it will sign the request with its own certificate and send it to some trusted Certificate Authority to obtain the certificate. The Certificate Authority will verify the request to make sure it doesnt come from a rogue sender or a compromised DHCP server. Figure 3 shows the architecture of deployment model.

In such a model, the certificate daemon must ne given the proof-of-possession of the certificate request. In some network that is less sensitive, the daemon can accept aggressive connections from the clients that produce the proof-of-possession segment by themselves. For a more secure way, the daemon can challenge the client with the public key and check if the client can answer the challenge. Additional messages are required to do the challenge.

There might be confusion on the role of the certificate daemon and local clients. The daemon is a local server that accepts requests from local clients. At the same time it is also a client to the CA/RA server. The workload of the certificate daemon can be large, since it has to be responsible for the validation of the public key and the deployment of the certificate request. As a result, the risk that such a server is compromised can be huge. To simplify the role of the Certificate Daemon, we defined a new role in the model called *Prover*.

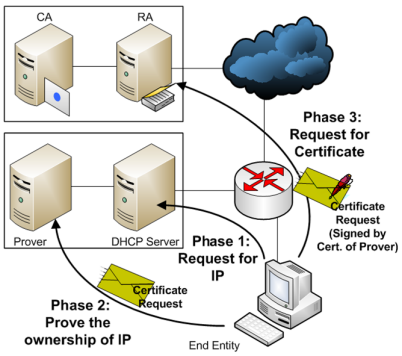

Figure 3: Phases for Certificate Deployment using Provers

The *prover* is an entity in the local network that is trusted by the DHCP server. It has the privilege to query the lease database and retrieve the pre-stored authentication information. The job of prover is easy: to prove the relationship between the allocated IP address and the public key that the owner uses. We can define the prover as an efficient algorithm that compute the following result:

```
Prover (IP address H, Public Key PK) =
    true, if the owner of H owns PK
    false, otherwise.
```

Now the certificate daemon is changed into a prover daemon. The prover daemon will validate certificate request and sign requests if it can prove the relationship of the IP address and the public key. The client will send the signed request to the CA/RA server not through the daemon but by themselves. Such a model is simpler than using certificate daemon, since there is no synchronization problem, collision problem, latency from the CA/RA responders, and it is easier to implement and extend.

We can argue that the security of certificate daemon and prover daemon is actually equivalent. Since no entity but the prover in the network has the ability to sign the certificate request, it does not matter if we let the clients to send the request by themselves. Suppose there exists an attacker that can deceive a Prover Daemon to sign a bogus request, it must also be able to deceive some Certificate Daemon to send the request for it. For these reasons, we prefer the model with a Prover Daemon than a Certificate Daemon.

Figure **??** shows the steps that the certificate is deployed in the model with a Prover.

Phase 0 The client will generate a key pair consist of a public key and a private key.

Phase 1 The client obtain an IP address from the DHCP server, an authentication code of the public key must be provided to the DHCP server and stored in the lease database.

Phase 2 The client creates a raw certificate request containing the public key. The prover will sign the request if it can prove the relationship between the IP address that the client owns and the public key that the client is using.

Phase 3 The client send the request to CA/RA server and obtain the certificate.

Sometimes a trivial prover can be used in a static network or a pre-allocated network. In a static network that each entity already owns some key pair generated by some trustworthy authority, the prover only has to maintain a database for the IP addresses and correspondent public keys, and no need to query any lease database on the DHCP servers. In some other cases, we can also use a password-based prover that authenticates the request by verifying the password. In the model that uses such a password-based prover, we assume that the clients will be responsible for the security of their own key pair.

4.2 Certificate Status Validation

We designed a certificate status validation algorithm to verify the certificates used to sign the payload. This algorithm involves a set of unverified certificates and a set of certificates of some trusted CAs. We also need a somehow certificate status checking service to check the revocations of those certificates.

Since we use a hierarchical CA structures in this model, each certificate will have a signing path that starts with some trusted root CA. To validate the certificate, we have to go through the whole signing path and validate each certificate that signs the ascendants of the end entitys certificate. This signing path can be long if we have a complicated hierarchy of CA structure. There is a risk of denial-of-service that either one of the certificates in the signing path might fail the verification and affect the usability of the system. We need an algorithm to dynamically collect status of the certificates and change the states of those certificates that share the signing paths or part of the signing paths. This algorithm has to be efficient enough to process a huge amount of certificate validation.

For relaxing the security and increasing the usability of the system, we defined the validity of a certificate to be Validity Depth which is an 8-bit integer. The definition of Validity Depth is as following:

```
If the certificate is absolutely trusted,
    Validity Depth = 0.

If the certificate is determined as invalid,
```

```
    Validity Depth = negative integer

If the certificate is determined as valid, but
we havent got enough information to verify the
signing path of the certificate,
    Validity Depth = 128.

If the certificate is determined as valid, and
we can verify the signing path of the certificate
to up to the d-th ascendant,
    Validity Depth = d.
```

By using such a relaxed definition of certificate validation, we can somehow make the system tolerant to failure of status check. Normally a Validity Depth up to 1 3 might be reasonably good enough for the system, since we assume that the end entity can only own some certificate with a very short validation time. Even if we cannot get timely status of a certificate, we still dont have to worry about the abuse of the certificate in a very short period of time.

The validation algorithm in this model is designed as following:

```
SetDepth( Certificate C , Depth D )
  If C.Depth > D then
    C.Depth <- D
    For every child C for C
    If C is checked as valid then SetDepth( C , D )
       Else SetDepth( C , D + 1 )

Certificate[] Store

Verify( Certificate[] Cs )
  For every C in Cs
    Check the status of C by using some status checking service

  If C is checked as invalid then SetDepth( C , -127 )
  If C is checked as valid   then SetDepth( C ,  127 )
  If C cannot be checked     then SetDepth( C ,  128 )

  If C in Store is found as a parent of C
    If C is checked as valid then SetDepth( C , C.Depth )
    If C cannot be checked then   SetDepth( C , C.Depth + 1 )

  If C in Store is found as a child of C
    If C is checked as valid then SetDepth( C , C.Depth )
    If C cannot be checked then   SetDepth( C , C.Depth + 1 )

  Add C to Store

  Return Cs.Depth
```

4.2.1 Certificate Status Validation with CRL

Unlike other certificate status checking service, CRL provide a list of revoked certificates for clients to look up. While retrieving CRL from a CA server, a X.509 CRL object is downloaded from the distribute point and we compare the certificates with the content of CRL. We need an additional algorithm to verify the CRL and integrate CRL objects into the certificate storage. If a CRL is not verified as a valid CRL object, the algorithm should not rely on the list given by this CRL object.

Applying the previous certificate validation algorithm, we have to assign CRL objects with the same definition of Validity Depth as certificates. Just like certificates, CRL objects can only be trusted when the whole signing path is validated. If a certificate in the storage is determined as the issuer of a CRL object, the Validity Depth of the CRL object should be set up the depth of the issuer certificate. As long as the issuer certificate of a CRL objects is found, all the certificates that are signed by the same issuer can be verify by this CRL object.

The validation algorithm is designed as following.

```
CRL[] CRLStore

VerifyCRL( CRL L )
  If L is not valid then return

  If found some Certificate C in Store that C = L.issuer then
    SetDepth( L , C.Depth )
    For each child C of C
      If C is revoked in L     then SetDepth( C , -127 )
      If C is not revoked in L then SetDepth( C , C.Depth )
    Add L to CRLStore
```

5 Discussion

5.1 Functionality

We implemented a PKI system that applies the core concepts we designed in this project. Both of the two solutions got successful results in the demonstrations. The experiment done in this project consists of a hierarchical CA structure (with a root CA and a local CA), a router with DHCP server, and two clients that each obtains an IP address in the network. For simplifying the problem, we didnt use IP forwarding or IP Masquerading on the router. The signing and verifying of the messages is only done end-to-end, but we believe the same methods can be used on edge-to-edge operations.

The implementation of the system is platform-dependent, so currently the solution cannot be used on non-Unix platforms. However the structure we designed is platform-independent, and it is feasible to build customized solution for any network and any platform. We try to build the system on existing network protocols, such as X.509, Certificate Management Protocol (CMP) and Cryptographic Message Syntax. Our solution is very friendly to developers who is looking for a suitable IP authentication framework.

5.2 Security

This solution is just a prototype of IP authentication framework, so it might be imperfect in some details of implementation. However we can prove that those security vulnerabilities are not structural vulnerabilities and can be fixed through certain improvement. We believe our system solved most of the security problems

encountered in the project and proposed a solution for most of the vulnerabilities we havent solve.

Involvement of DHCP server in the architecture gives us new advantages to authenticate against IP spoofer. We assume that all the hosts in the local network obtained the IP addresses from the DHCP server through legal DHCP requests. Even though we dont really worry about the security of DHCP messaging (It is not the purpose of this project.) it should be kept in mind that such an assumption might be too strong for the real-world network. However, skipping the security problem of DHCP messaging, our solution avoids any bogus certificate requests that come from IP spoofers and provide a method for legal users to register themselves before they obtained the address.

Note that in the second solution, for using client ID as a digest of public key, it might be too short for generating a 32-bit long digest for the public key. An attacker can build up a dictionary with 2^{32} different key pairs that try to race with the legal user to send out certificate request. One solution for this problem will be putting random nonce or timestamps into the digest to make the digest non-deterministic. (For using timestamp, there might be a synchronization problem of system time between servers and client.) Another solution, which provides even better security, is to create a new option to carry the digest.

One of the few attacks that can happen in the second solution is replay attack. However, just like described in the previous paragraph, a replay attack cannot really get any benefit from the system. Since the Certificate Authority accepts replayed certificate request, such an attack doesnt really affect the common usage of a legal user. As a result, the attacker will get a certificate with some private key that he didnt know, he cannot really make use of this certificate.

Man-in-the-middle attack is actually a larger threat in this system. We defined a mechanism for how the user can be authenticated by the server (no matter CA server, DHCP server, certificate daemon or prover daemon). However we miss the definition of the server authentication in the framework. The user that joins into the network might be misled by a rouge server, and become a victim of Main-in-the-middle attack.

Another attack we might want to worry about is denial-of-service attack. An easier attack might be some client in the local network trying to occupy all the IP addresses in the range and prevent other client to obtain IP addresses before its certificates expire. In order to solve this problem, we need a collector to collect all the unused IP addresses in the local network and release them to new users. Another Denial-of-service attack might happen on a router that verifies a large amount of certificates. The workload of the verifier on the router can be overridden by a client send out a large amount of unverified certificates.

One of the problem that we should always keep in mind is the possibility that a DHCP server/certificate daemon/prover daemon being compromised. The CA/RA server has to revoke the certificates of those compromised servers timely to reduce the risk that those servers are being abused. In principal, the CA/RA server should not accept any request from a DHCP server/certificate daemon/prover daemon related to a IP address that isnt owned by the domain. Therefore even if an attacker broke into one of the local server in the domain, it cannot spoof its IP address that belongs to another domain. Also we can assign different policy on different domains, for some domains, such as DMZ, might be more sensitive on the integrity of IP address identity.

6 Future Work

The construction of this framework is finished, but some details of the system need improvement on the security. In the dynamic deployment solution with a prover daemon, we need a better way to carry authentication information in the DHCP messages. A new DHCP option will definitely be a good solution on this problem. In addition, we need a better management on the authentication information stored in the lease database. The management system has to prevent collision and confusion on the identity of different client.

We also need an algorithm to manage the certificate in the database of the certificate authority and in the cache of router verifier. Consider about the amount of certificates in a network, the overhead of verifying or managing those certificates can be very expensive.

Finally, we want to refine the solution and the API, and make it more flexible, usable, extensible and platform independent. We believe that this solution can be a general solution to solve the authentication problem of local IP address identity.

References

[1] FC5280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile `http://tools.ietf.org/html/rfc5280`

[2] FC 4210 - Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP) `http://tools.ietf.org/html/rfc4210`

[3] FC 4211 - Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF) `http://tools.ietf.org/html/rfc4211`

[4] FC 2560 - X.509 Internet Public Key Infrastructure Online Certificate Status Protocol (OCSP) `http://tools.ietf.org/html/rfc2560`

[5] TU-T X.690 ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER) http://www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf

[6] raft of Secure BGP http://tools.ietf.org/html/draft-clynn-s-bgp-protocol-01

[7] anifests for the Resource Public Key Infrastructure http://tools.ietf.org/html/draft-ietf-sidr-rpki-manifests-07

[8] JBCA http://ejbca.sourceforge.net/

[9] penSSL Project http://www.openssl.org/

[10] ouncy Castle Project Java Home http://www.bouncycastle.org/java.html

[11] ovosec Bouncy Castle Extension http://sourceforge.net/projects/novosec-bc-ext/