PKI Services for IPv6

A recent European research project provides an ideal opportunity to migrate the Java-based UMU-PKI to IPv6 and build new security services over it.

Generally speaking, a public key infrastructure (PKI) is a set of hardware, software, people, and procedures needed to create, manage, store, distribute, and revoke public key certificates. With these in place, a PKI can provide trusted and efficient private- and public-key certificate management, thus enabling the use of authentication, nonrepudiation, and confidential security services. To provide such services, a PKI uses its base components, which include a certification authority, at least one registration authority, and a request server. Some PKIs use extra components, depending on the services their particular implementations offer.

At the University of Murcia, Spain, we developed UMU-PKI, which uses open-source software and Java for its internal components (see https://pki.dif.um.es). UMU-PKI supports LDAP directories, Java cards, and RSA smartcards; it also has policy-based management and Web-based registration methods for end users. In addition to this, it implements three advanced security services: a time-stamp authority, an online certificate status protocol (OCSP) authority, and a simple certificate enrollment protocol (SCEP) server, which can accept certificate requests from secure gateways.

The European IPv6 Internet Exchanges Backbone (Euro6IX, www.euro6ix.org) project is one of the major focuses within the EU’s 5th Framework Program and is devoted to the implementation of a European native IPv6 network. It provided us with the perfect opportunity to test UMU-PKI to ensure that it could migrate current security services from IPv4 to IPv6 (see the sidebar). The project also gives us real IPv6 experience with end users, ISPs, and Internet exchanges (IXs). This article describes how UMU-PKI fared in testing and which new services we can build over this IPv6-enabled PKI.
Promoting IPv6 in Europe

The Euro6IX project is a consortium of telecommunications operators, IT companies, and universities from all over Europe that are deploying and testing a large-scale native IPv6 network and a broad range of services for it. Within these services, security — especially a security infrastructure based on the PKI concept (see www.ietf.org/html.charters/pkix-charter.html) — represents a major point of interest. The overall objective of having a PKI-supporting IPv6 is to provide a trust point (or basic security framework) for the deployment of new security services over IPv6 (see www.ietf.org/html.charters/aaa-charter.html).

The Euro6IX project’s primary objective is to support the fast introduction of IPv6 in Europe by advancing research on the issues of network design and deployment, advanced services, and development and porting of user-validated services and applications. Its second objective is active dissemination of the results and experiences coming from this research activity.

The technical approach followed to achieve both goals is based on building a large, scalable, native IPv6 backbone of traffic exchanges with connectivity across Europe (see Figure 1). To promote IPv6 use, the project’s consortium members will develop or migrate new services and applications to this new protocol.

The EU Commission typically asks project consortiums to group a set of activities into what it calls work packages, each with a whole set of activities. The Euro6IX project is divided into five work packages, but one of the most relevant activities to this article’s scope is related to advanced services and IPv6 applications. This activity brings the developments, migrations, or enhancements of the applications needed to create a realistic set of end-user services into internal and public tests to be performed over the project’s duration (January 2001 to January 2004). UMU-PKI is one of the services migrated to IPv6 and extended with new functionalities; several new services will benefit from it.

The UMU-PKI Service

UMU-PKI supports a robust group of certification services. It lets end users perform a whole set of activities from a Web browser: requesting, renewing, or revoking a certificate, looking for another...
user’s certificate, and so on. In a general PKI, you can store your private key and certificate in your browser’s database. UMU-PKI lets you use smart cards instead, which gives users greater mobility and increases the whole system’s level of security.

UMU-PKI’s most innovative characteristics are that it

- supports the definition of a certification policy, which the administrator defines to establish security restrictions inside an organization;
- is completely developed in Java, which allows implementations of the PKI to run on any operating system;
- is based on the drafts and standards specified by the IETF’s PKI Exchange (PKIX, www.ietf.org/html.charters/html.charters/pkix-charter.html) working group;
- supports SCEP, which in turn enables router-certificate requests;
- supports OCSP; and
- implements time stamping in the system.

Figure 2 shows all of UMU-PKI’s components, including the registration authority (RA), the request server, and the certificate authority (CA). A system could have several RAs, each with a different administrator, for validating and sending different certification requests to the CA. In our system, we implemented the RA as a standalone Java application.

The request server stores all the certification, renewal, and revocation requests generated by end users or other PKI components, including RAs, processes, or active services. All these requests are stored in an internal database. This database is in the request server and is only accessible by the CA and request server, meaning RAs and final entities (such as end users, virtual private network (VPN) devices, administrators, and so on) can’t access it. There is no direct connection from this server to the CA because, for security reasons, the CA always works offline and never accepts incoming connections. The request server is implemented via an Apache HTTP server with a set of Java servlets managing all the certification requests (which are temporally stored in an internal PostgreSQL database).

The CA, which is also implemented as a standalone Java application, processes all the requests stored in the request server. For a certification request, the CA first checks the PKI’s internal policy to see if the certificate can be issued; if so, it signs the certificate, stores it in the internal database, and publishes it in the LDAP server. The CA then notifies the user via a digitally signed email message. For a renewal request, the CA simply updates the certificate in the internal database and in the LDAP server. For a revocation request, the CA marks the certificate as revoked in the internal database and includes it in the next certificate revocation list (CRL) published by the CA in the LDAP server.

Java Security Mechanisms

Once we know why we’re porting UMU-PKI to IPv6, we next need to determine which specific components must be analyzed regarding their IPv6 status.

The first (maybe most important) one is IPv6’s support of Java. We deployed two of UMU-PKI’s three main components — the RA and the CA — with Java. Furthermore, the Java security providers’ status is vital to us because, for example, the RA’s software uses them to generate new certificate requests, which the CA’s software will sign later.

Besides the RA and CA, the third component of importance in our PKI is the request server, which is based on an HTTP server that supports Java servlets and SSL communications. Thus, we must analyze IPv6’s status and the feasibility of using existing open-source HTTP servers.

To provision IPv6 support in Java and its secure components, we turned to the Java Cryptography Architecture (JCA), which is a framework for accessing and developing cryptograph—
ic functionality in Java platforms. It defines a provider architecture for multiple and interoperable cryptography implementations, including APIs for digital signatures, message digests, and so on.

The Java Cryptography Extension (JCE) builds on the JCA API and includes interfaces for encryption, key exchange, and message authentication code (MAC). Both the JCA and the JCE provide complete, platform-independent cryptographic APIs. Several JCA–JCE cryptographic providers exist, and we compared their basic functionalities to find the right one for migrating UMU-PKI to IPv6.

- **IAIK-JCE.** The Institute for Applied Information Processing and Communications' JCE (IAIK-JCE, http://jce.iaik.tugraz.at) is a set of APIs and implementations of a robust group of cryptographic functions, including symmetric, asymmetric, stream, and block encryption methods. According to our experience, it has a complete Abstract Syntax Notation One (ASN.1) library, supports Public-Key Cryptography Standards (PKCS) 1, 5, 7, 8, 9, 10, and 12, and has a good X.509 certificate and extensions implementation. It supplements the security functionality of the default Java JSDK 1.2.1 through JSDK 1.4, including digital signatures and message digests. It also includes APIs for SSL communications and secure/multipurpose Internet mail extensions (S/MIME) operations.

- **JCSI.** The Java Crypto and Security Implementation (JCSI, http://security.dstc.edu.au) applies standard security services in a manner compatible with JCA. It uses most of the security algorithms currently supporting public key cryptography and the cryptographic message syntax (CMS) standard, although its ASN.1 library is very limited. The same library supports basic SSL communications and S/MIME functions and works with JDK versions 1.2 through 1.4.

- **Bouncy Castle Crypto APIs.** The Bouncy Castle Crypto APIs (www.bouncycastle.org) consist of a lightweight cryptographic API in Java, a provider (or library that implements an API) for the JCE and JCA, generators for version-1 and version-3 X.509 certificates and PKCS #12 files, and a signed Java archive (jar) version suitable for JDK 1.4 and the Sun JCE. In addition, it has a CMS and S/MIME implementation library.

- **BeeJCE and BeeCrypt.** The BeeJCE (www.virtualunlimited.com) is an implementation of version 1.2 of the Sun JCE with a basic group of cryptographic algorithms. BeeCrypt is an open-source cryptography library supporting PKCS #1 and PKCS #5 standards and X.509v3 certificates. Both are open source, but neither supports JDK 1.4.

- **Baltimore Keytools.** Baltimore Keytools (www.baltimore.com) is a complete cryptographic library including a JCA–JCE provider and SSL–S/MIME protocol support. It supports PKCS #1, 5, 8, and 12, an ASN.1 library, an XML library, and the X.509/OCSP standard. The version we tested, KeyTools Pro v5.0.6.1, does not support JDK 1.4.

- **Phaos toolkits.** Phaos Crypto (www.phaos.com) is a JCA–JCE implementation that provides the core algorithms for cryptography in Java including the implementations of the RSA public key cryptosystem, the RC4 stream cipher, DES and triple-DES encryption, MD5 and SHA message digests, and Diffie-Hellman key agreement. Besides this, Phaos offers SSL, S/MIME, and XML libraries and supports JDK 1.4.

Table 1 compares the most interesting features of each implementation that we analyzed. Only those solutions that support JDK 1.4 are valid for use in UMU-PKIv6 components. (UMU-PKI is the basic implementation we migrated to IPv6; the final version with IPv4/IPv6 support is called UMU-PKIv6. Think of UMU-PKIv6 as UMU-PKI with more fea-

<table>
<thead>
<tr>
<th>Provider</th>
<th>Version</th>
<th>Open source</th>
<th>JDK 1.4</th>
<th>SSL</th>
<th>S/MIME</th>
<th>X.509</th>
<th>CRL</th>
<th>OCSP</th>
<th>CMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAIK</td>
<td>3.0</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>JCSI</td>
<td>2.1.1</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>BC</td>
<td>1.1.3</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Bee</td>
<td>1.02</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Keytools</td>
<td>5.0</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Phaos</td>
<td></td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
tures, including IPv6 support.)

We selected IAIK because it complies with all UMU-PKIv6 requirements, works properly with JDK 1.4, has a very good ASN.1 library, supports the X509/OCSP/CRL standard, and implements the large group of X509 extensions needed to offer a range of cryptographic services to end users and devices.

HTTP-over-IPv6 Solutions
One of UMU-PKI’s main components, the request server, is based on an HTTP server that supports a Java servlet module and SSL communications. We analyzed four HTTP server implementations that support IPv6 to select the most suitable one for our purposes:

- **Apache 2.0** (www.apache.org) is the most popular HTTP-over-IPv6 implementation. In systems with IPv6 support, it gets IPv6 sockets by default: the configuration file has been upgraded to read IPv6 addresses in the Listen, NameVirtualHost, and VirtualHost directives. Apache 2.0 also supports SSL connections by default.

- **Apache 1.3** is a broadly used HTTP server. By default, this version does not have native IPv6 support, but several patches are available for it. The SSL support is very stable with the ModSSL module, which provides strong cryptography for this Web server thanks to its use of the OpenSSL library. ApacheJServ is the best implementation for Apache version 1.3.

- **Trivial/Tiny/Turbo/Throttling HTTP server** (THTTPD, www.acme.com/software/thttpd) is a simple IPv6-ready Web server. It is fast and small but does not have native SSL support, although there is a patch for it. THTTPD’s main problem is that it does not support the servlet API that our system needs, so we had to discard it.

- **MINI_HTTPD** (http://heliso.tripod.com/download/swww/mhttp.htm) is a basic HTTP Web server with IPv6 support. It also supports SSL connections, but like THTTPD, it does not work with the servlet API.

We thus found that only the two versions of Apache can support all the characteristics the current UMU-PKI implementation requires.

Servlet API Implementations
For the final implementation of the UMU-PKI request server, we needed a suitable combination of Apache and servlet API versions. Our main options were Tomcat and ApacheJServ, but some combinations of Tomcat and Apache weren’t possible because the Web server must support SSL as well. We learned some interesting lessons along the way:

- **Apache 2.0 with Tomcat 4.0.** Installing Apache 2.0 with IPv6 and SSL is very easy and goes quite smoothly in Unix-like systems. To connect Tomcat with Apache to add servlet support, however, we need a connector — the Windows Astronomical Resource Protocol (WARP) or Apache Jserv Protocol (AJP) — to handle requests. Apache 2.0 with Tomcat 4 work fine with IPv6, and servlets can be executed with either connector, but SSL support is not fully implemented for the combination. Moreover, SSL client attributes (for example, SSL_CLIENT_CERT) cannot be retrieved inside the servlet code.

- **Apache 2.0 with ApacheJServ.** The Apache 2.0 HTTP server does not support the ApacheJServ API because it requires the API servlet version 2.3.

- **Tomcat 4.0 standalone.** Tomcat 4.0 can be used as a stand-alone HTTP server through a connector component supporting HTTP 1.1. It can also run servlets and JavaServer pages (JSP). IPv6 access is possible through Tomcat 4.0. Although Tomcat as stand-alone server supports SSL, however, it cannot retrieve SSL client attributes from servlets over a secure connection.

- **Apache 1.3 with Tomcat.** Apache 1.3 has no native support for IPv6, but a patch from the KAME project (ftp://ftp.kame.net/pub/kame/misc) adds it specifically to version 1.3.19. Tomcat can be installed easily and works fine, but the WARP and AJP connectors cannot be compiled if the IPv6 option is active.

The solution we picked was Apache 1.3.19 with the IPv6/SSL patch and ApacheJServ with servlet 2.2 API support. This solution is quite stable and satisfies all UMU-PKIv6 requirements.

UMU-PKIv6 Services in Different Scenarios
The definition of projects and associated pilots, or testbeds, on IPv6 marks a qualitative leap forward. These pilots are becoming increasingly important to prove this technology’s maturity to end users and ISPs.

Within the Euro6IX project, we’ve successful-
ly tested and validated UMU-PKIv6 [https://pki.ipv6.um.es] in conjunction with several IPv6 services. Figure 3 depicts the specific PKI architecture on which we designed and deployed several test scenarios.

**Static VPNs with IPv6**

We tested UMU-PKIv6’s performance in issuing digital certificates for the main existing IPsec and Internet Key Exchange (IKE) implementations for IPv6. We then defined an evaluation plan to analyze these implementations’ interoperability and conformance levels.

Three basic scenarios emerged:

- **Only hosts apply IPsec.** This scenario is usually defined when two hosts across an insecure network establish a secure communication via IPsec.
- **Only secure gateways apply IPsec.** This scenario tests situations in which a VPN is created between two local networks.
- **Both host and secure gateway apply IPsec.** This scenario, also known as “road warrior,” tests situations in which a VPN is created between a secure gateway and a final host.

We tested all these scenarios with a test suite composed with basic tools to measure network throughput – ping6, telnet6, and pchar – and some advanced applications – mainly Isabelv6 (http://isabel.dit.upm.es), a multimedia computer-supported cooperative-work (CSCW) application for distributed events – to evaluate network performance when we use real applications.

Based on this experience, we deployed a static VPN service in the Euro6IX communication testbed.

**Dynamic VPNs with IPv6**

IPsec is a typical policy-enabled networking service in which security functions are executed properly only if policies are correctly specified and configured. Following current practice, however, network administrators usually configure IPsec policy databases manually, which is quite inefficient and error-prone for large distributed networking systems. Also, the growing number of secure Internet applications and services is making IPsec policy deployment increasingly complex.

A policy-management system is clearly needed to enable automatic configuration and management of IPsec policies within single or multiple administrative domains.

The Euro6IX project has created such a system, called policy-based network management (PBNM). It is divided into two components:

- a distributed, automated, and coordinated system for managing network services (in our case, IPsec security services) and defining policies that rule these services, and
- a flexible service for discovering, accessing, and processing these policies.

Both subsystems have one element in common: the trust point defined by UMU-PKIv6’s CA.

With this system, administrators need not configure IPsec devices directly. The administrators just introduce how the network will behave regarding security; the PBNM system automatically enforces policies for every IPsec-enabled device in a specific communication domain.
Secure AGWSv6

We have used UMU-PKIv6 to secure several applications and services. One of the most interesting examples is the Agora Groupware Web Service for IPv6 (AGWSv6, www.agoratechnologies.com), which is a powerful e-project, e-meeting, and e-learning tool for asynchronous collaboration over the Internet.

This application’s users have their own digital certificates that let them use and access those parts of the system they’re authorized to work with. Their rights are associated with their UMU-PKIv6-issued digital certificates and stored in an internal system database.

Conclusions

A PKI is a key component for most of the current and future secure communications architectures and distributed application environments. Thus, the process of migrating UMU-PKI to IPv6 is important for the successful deployment of IPv6 as a basic component of the future Internet.

Numerous services and applications depend on this basic security framework to be deployed properly. This is the case of secure VPNs or AAAv6, for example, which the Euro6IX consortium is currently working on.

Regarding UMU-PKI, we’re currently designing and testing new services within the Euro6IX network. We’re attempting to support cross-certification between different certification authorities, to use DNSsec as a distributed certificate store, and to manage attribute certificates that define the different roles of users and processes inside an organization.

Acknowledgments

The EU Euro6IX project IST-2001-32161 partially funded this work.

References