Plug-and-Play PKI: A PKI your Mother can Use

Peter Gutmann
University of Auckland

Abstract

A common complaint about PKI is that it is simply too hard to use at the end-user level. Somewhat surprisingly, there exists no PKI equivalent of DHCP or BOOTP for automated, transparent PKI setup, leaving the certificate user experience similar to the process of bringing up an X.25 link. This paper provides a PKI equivalent of these basic bootstrap services that provides automatic, transparent configuration and setup of certificate information, with the user needing to supply no more than their user name and password. The work covers the design process involved and tradeoffs made, implementation details, and experiences with actual operation. The overall purpose of the work is to design and implement a plug-and-play certificate setup mechanism usable by even the most inexperienced user, a "PKI your mother can use".

1 Introduction

Despite many years of effort, PKI technology has failed to take off except in a few niche areas. Reasons for this abound, and include the difficulty of deploying the technology, cost, lack of interoperability, and the poor match of PKI designs to any pressing real-world problems. Probably the primary factor at the user level though is the high level of difficulty involved in deploying and using a PKI.

There is considerable evidence from mailing lists, Usenet newsgroups and web forums, and directly from the users themselves, that acquiring a certificate is the single biggest hurdle faced by users[1]. For example various user comments indicate that it takes a skilled technical user between 30 minutes and 4 hours work to obtain a certificate from a public CA that performs little to no verification, depending on the CA and the procedure being followed. Obtaining one from non-public CAs that carry out various levels of verification before issuing the certificate can take as long as a month. A representative non-technical user who tried to obtain an (unverified) certificate from a public CA took well over an hour for the process, which involved having the author tell them where to go to start the process, filling out eight (!!) browser pages of information, having to spend time finding their passport, several restarts due to values being rejected, waiting for emailed instructions, numerous questions to the author about various points of the process, cutting and pasting data values from an email message to a web page, and filling in more information over a succession of eleven further web pages (including several that were completely incomprehensible to the user, she just clicked "Next"). Eventually, a final page announced that a certificate had been issued, although the user was never able to locate it on the PC, and couldn't understand much of the information on the web page, which referred to things like "certificate Distinguished Names" and "X.509 SubjectAltName" (there was a button labelled "Fetch", but this seemed to have no effect). Eventually an emailed message provided a URL to download the certificate, which, after switching to a different web browser and providing further names and passwords yielded a file on disk which the user located after considerable searching. Clicking on the file had no effect, at which point the user gave up. In a similar study, a set of highly technical users, most with PhDs in computer science, took over two hours to set up a certificate for their own use and rated it as the most difficult computer task that they'd ever been asked to perform [1].

In contrast, obtaining the Internet connection that enabled the certificate process took around ten minutes and involved the exchange of a user name, password, ISP phone number, and a credit card number (the sample certificate was free, or the process would no doubt have taken even longer). Similarly, connecting a new system to a LAN usually requires no more than a user name and password for access to the appropriate server, with the rest being taken care of transparently via mechanisms such as DHCP. In contrast, the process of obtaining a certificate most closely resembles the experience of bringing up an X.25 link. The usability problems inherent in such a system are summed up by a book that examines the reasons for the success of peer-to-peer software: "the vast majority of users detest anything they must configure and tweak. Any really mass-appeal tool must allow an essentially transparent functionality as default behaviour; anything else will necessarily have limited adoption" [2].

[1] The use of anecdotal evidence in this paper is an unfortunate necessity because few public PKI usability studies exist, and the few that do tend to be post-mortem analyses.

The problems that this creates are demonstrated by what happens when technically skilled users are required to work with certificates. The OpenSSL toolkit [3][4] includes a Perl script `CA.pl` that allows users to quickly generate so-called clown suit certificates (ones that "have all the validity of a clown suit" when used for identification purposes [5]), which is widely-used in practice. The cryptlib toolkit [6][7] contains a similar feature in the form of Xyzzy certificates (added with some resistance and only after the author grew tired of endless requests for it), ones with dummy X.500 names, an effectively infinite lifetime, and no restrictions on usage. Most commercial toolkits include similar capabilities, usually disguised as "test certificates" for development purposes only, which end up being deployed in live environments because it's too difficult to do it the way X.509 says it should be done. Certificates used with mailers that support the STARTTLS option consist of ones that are "self-signed, signed-by the default Snake Oil CA, signed by an unknown test CA, expired, or have the wrong DN" [8]. The producer of one widely-used Windows MUA reports that in their experience 90% of the STARTTLS-enabled servers that they encounter use self-signed certificates [9]. This reduces the overall security of the system to that of unauthenticated Diffie-Hellman key exchange, circa 1976. In all of these cases, the entire purpose of certificates has been completely short-circuited by users because it's just too difficult to do the job properly. The problematic nature of X.509 is echoed in publications both technical and non-technical, with conference papers and product descriptions making a feature of the fact that their design or product works without requiring a PKI. For example, one recent review of email security gateways made a requirement for consideration in the review that the product "have no reliance on PKI" [10]. As an extreme example of this, the inaugural PKI Research Workshop, attended by expert PKI users, required that submitters authenticate themselves with plaintext passwords because of the lack of a PKI to handle the task [11][12]. In other cases, initiatives that had tried to deploy PKI switched back to standard passwords due to the volume of user complaints about complexity and usability problems [13].

The goal of this paper, then, is to design and build a certificate-handling mechanism whose use is no more difficult than the task of obtaining an ISP account. Somewhat less formally, the intent is to "build a PKI your mother can use". Since ISPs have had many years of experience in making what was originally a complex operation best left to experienced technical users accessible to everyone, this paper uses the approach used by ISPs to solve similar problems as a starting point for solving equivalent problems from the PKI field.

2 The Problem

Security texts generally recommend the following security principles for good key management [14][15][16]:

- Encryption (as opposed to signature) keys should have as short a lifetime as possible in order to shorten their window of vulnerability. Changing keys frequently reduces the damage if a key is compromised.
- Different keys should be used for different purposes. For example one (short-term) key might be used for encryption, a (different) medium-term key for ephemeral authentication and a long-term key for digital signatures that need to be validated years after signing. Even in the latter case, only the public portion of the key should be retained over the long term for signature verification, with the private portion being destroyed to reduce potential exposure.
- Keys should be carefully protected, and never distributed

A consequence of the difficulty in obtaining certificates is that users break every principle of good key management in order to avoid the pain of acquiring/replacing a certificate [17]:

- CAs use certificates with effectively infinite lifetimes (20-40 years).
- Users use certificates with effectively infinite lifetimes (even if the CA limits it to 1 year for billing purposes) by re-certifying the same key year in, year out. Many users see this as perfectly natural and sensible, and some CAs concur.
- The same key is used for everything (encryption, long-term signing, ephemeral authentication, etc, etc). This is regarded as a CA and/or software feature by many users. For example, the Windows CryptoAPI allows an encryption-only key to be used for signatures as well, since this is more convenient for users than having to manage two keys.
- The private key is copied around freely so that every application on the system and in some cases every user or machine on the network can share it. Again, the ability to freely distribute the private key is regarded as a software feature.

- In some cases, this effect can even stall the deployment of newer, more secure mechanisms to replace older ones with known security problems. One of the contributing reasons why RSA with PKCS #1 padding is still in almost universal use rather than being replaced with a more secure alternative such as RSA OAEP or RSA PSS is that it's too difficult to update the certificates involved [18].

In all of these cases, the pain of certificate acquisition and renewal has overridden all security concerns. The problem we face, then, is how to make certificate acquisition and update as transparent and simple as possible. The best way to determine what needs to be done is to examine the individual issues raised by the certificate acquisition example given in section 1, and provide representative solutions as used in the ISP world.

2.1 Initial Assumptions

Before we begin, we need to define the environment within which we're operating. The work presented here makes the following basic assumptions:

- There is an existing relationship between the user and some entity involved in the certification process that can be used for authentication purposes. For example when a user signs up for online banking or a loyalty program, the bank or vendor typically sends them an authenticator for the online enrolment in the post, or communicates it over the phone or in person. A similar out-of-band exchange of initial authentication information is assumed here. This follows existing practice and uses well-established mechanisms, which is important not only for obvious practical reasons but also because it comes with valuable legal precedent in case of any dispute.
- There is a server available that provides the certification services discussed further on, or at least the PKI service-locator service. This is the PKI equivalent of a DHCP server. Just as with a DHCP server, a user can still manually configure alternatives if they so desire or if the server is unavailable. The server may provide the services directly, for example as part of an ISP's overall service package or through an in-house CA, or it may merely act as a front-end or forwarder to another service provider such as a traditional public CA. For example it's quite possible that a large service provider such as AOL or MSN may want to run a CA as a value-added service for users. Similarly, large organisations such as corporates, universities, and government departments may run their own CAs. Smaller organisations, on the other hand, would typically farm the service out to an external CA.
- We're not designing a system to handle nuclear weapons launch codes. In particular, the system need only be as secure as an equivalent non-PKI alternative: "Cumbersome technology will be deployed and operated incorrectly and insecurely, or perhaps not at all" [19]. For example, typical online operations for which a certificate might be used (shopping, banking, vendor loyalty programs) all get by with a user name and password as the basic input to the authentication process. If it's good enough for existing applications, it's good enough here, although we use more appropriate cryptographic support (MACs and digital signatures) than the usual passwords-over-SSL.

The requirements that need to be met by the enrolment process are currently the subject of an extensive public consultation process being performed by the New Zealand State Services Commission [20], who have been soliciting input from various sources including government departments, businesses, and end users, as well as special-interest groups such as the New Zealand Privacy Commissioner. The work also takes into consideration similar work being carried out in Australia, Canada, Ireland, Singapore, and the UK, and will be the subject of a future paper.

The consultation process has among its policy goals producing an enrolment system which is acceptable to potential users (taking into account the different needs of different users), providing appropriate privacy protection, and being fit for its purpose, avoiding unnecessary over-engineering. The implementation goals include providing a user-focused system which is as convenient, easy to use and non-intrusive as possible, being flexible enough to accommodate change, being affordable and reliable for the public and for government agencies, and very importantly complying with relevant law, including privacy and human rights law, and functioning in a way that provides legal certainty. A final implementation goal is that it provide functional equivalence to existing systems, so that the authentication requirements are similar to those that apply to existing transactions except in cases where the online nature of the transaction significantly changes the level of risk. Although the State Services Commission is not considering a PKI-based enrolment system for individuals, the principles that underpin their work can be adopted to develop a Plug and Play PKI.

Now that the initial assumptions have been laid out, we can examine the various design problems that we need to address.

2.2 Locating the Certificate Service

The user was required to manually locate the certification service (typically a Certification Authority or CA) that they were using. In contrast a protocol such as DHCP automatically locates the services required for Internet access without the user even being aware that this is happening.

2.3 Entering User Data

Once the user had located the CA service, they were required to enter a considerable amount of data before they could go any further. In particular, despite the fact that the certificate being requested would contain no more than an (unverified) name and an email address (the rest is fixed data set by the CA), the user was required to fill in pages and pages of information unrelated to the certificate. After providing authentication data, the user was emailed back multiple authenticators that had to be cut and pasted into web pages and then had to answer several more pages of questions before a certificate was issued. Imagine the loss of productivity faced by any organisation that has its employees enrol for certificates if each employee is forced to run this gauntlet!

In contrast, signing up for an ISP account or connecting to a corporate network requires a single user name (to identify the user) and a password (to authenticate them). Even processes such as online banking (with far more at stake then a zero-value email certificate) can get by with a simple name and authenticator.

2.4 Obtaining the Certificate

The process of obtaining the certificate once it was issued was problematic and difficult for the user to comprehend, both conceptually (the user was told that a certificate had been issued, but all that meant was that a certificate existed somewhere, not that the user actually had it), and practically (it didn't work properly). In contrast, the success or failure of the ISP signup process is immediately obvious, and requires no further action by the user (barring the usual networking snafus).

2.4.1 PKI-enabled Toasters

A slightly different situation occurs when the certificate user isn't a human being but a device, particularly a device with a very limited user interface that can't function as, or emulate, the behaviour of a human being: the Internet-enabled toaster. Designing mechanisms to make working with these devices possible is the goal of zero-configuration networking [21][22][23], inspired by the host-requirements RFC observation that a self-configuring networking system would permit "the whole suite to be implemented in ROM or cast into silicon [...] it would be an immense boon to harried LAN administrators as well as system vendors".

The ability to initialise a PKI-enabled toaster is an additional plug-and-play PKI service that has no equivalent in the ISP world.

2.5 Bootstrapping the PKI

A final problem, which was hidden by the use of the web browser, is obtaining the initial CA certificates that are used to verify other certificates. Web browsers address this problem by including a large (well over 100 certificates for current browsers) collection of certificates hardcoded into them. These hardcoded certificates include ones from totally unknown CAs (including ones whose private keys have been on-sold to various third parties [24]), CAs with moribund web sites, 512-bit keys, policies disclaiming all liability for any use of the certificate or reliance on information contained in it, and similar worrying signs [25]. Furthermore, because the browser trusts all CA certificates in its collection equally, the overall system is only as trustworthy as the least trustworthy CA in the collection. In other words, Honest Joe's Used Cars and Certificates is assigned the same status as the Verisign Class 1 Public Primary Certification Authority, and can usurp its certificates if it so desires.

This bootstrap process, referred to in this paper as PKIBoot, is a PKI-specific issue that has no equivalent in the ISP world.

2.6 Problem Summary

In summary, we need a system with the following features:

- Transparent discovery of PKI services
- Bootstrap functionality that doesn't assume pre-existing trusted certificates
- Simple (user name + password) enrolment process
- Automated acquisition and renewal of certificates

Section 3 examines possible mechanisms that may be used to provide these services, and sections 3.2.1 and 5 go on to cover the actual implementation, and implementation experiences.

3 Approach

In order to meet the requirements listed in section 2.6, we need three distinct mechanisms:

- A service location mechanism
- A bootstrap mechanism
- A certificate acquisition/renewal mechanism

The abstract flow of operations for this process is shown in the sequence diagram in Figure 1. Initially, the user needs to determine where to go to obtain certificate services. This is handled by the service location step. Employing security mechanisms at this very preliminary level would be rather problematic since the actual service that handles security hasn't been engaged yet. There is in fact no need for this service to be secure, since the interaction with the certificate service that follows is authenticated, in the same way that SSL (in theory) doesn't require a secure DNS service since it authenticates the client and server. In practice though, SSL authenticates the server using a certificate from one of over a hundred CAs hardcoded into the client software that the user has no choice but to trust, and omits user authentication entirely because it's just too hard to manage, providing a good example of the exact problem that the plug-and-play PKI services are intended to solve.

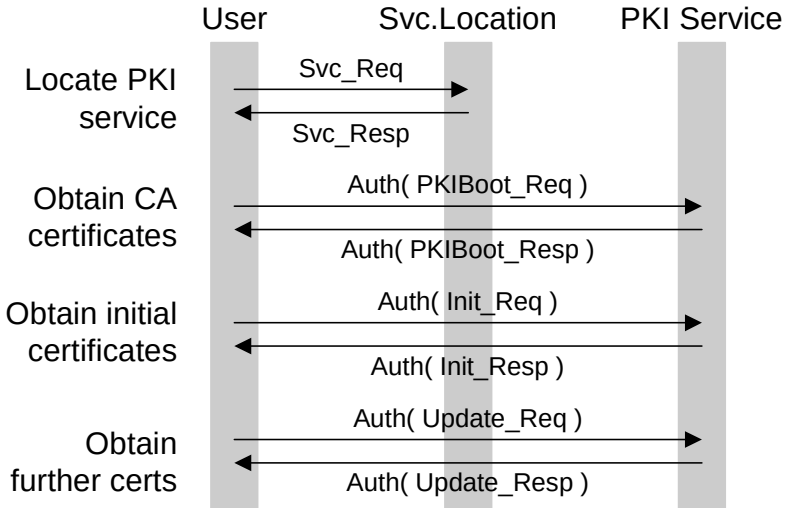

Figure 1: Plug-and-play PKI setup process

Once the user has located the certificate service provider, they cryptographically authenticate themselves to the server via the mechanism described in section 4.3 and request an initial set of certificates such as trusted CA certificates. For example if the certification service were being run by the user's employer, the server would return the employer's CA certificates (and any other certificates that the employer considers trustworthy, for example those of trading partners) to the user, authenticated using the same mechanism as the one used by the user. This is

equivalent to the certificate collection normally hardcoded into web browsers, except that the user is accepting a small set of certificates from the trusted service provider rather than a large collection of arbitrary certificates from sources completely unknown to them.

Now that the user has the necessary CA certificates, they can obtain their first certificate, identified by their user name and authenticated by their password (again, used as input to the appropriate cryptographic mechanism). In the case of the Internet-enabled toaster where there's no password available, the authentication is provided by a unique ID such as the toaster's serial number (which doesn't have to be private, merely unique) and the use of a private network segment for initialisation (in other words the security controls at this initial level are physical rather than electronic, exact details of the process are given in section 3.2.1).

Finally, the user can request further certificates using either their user name and password as before, or (more conveniently and logically) using the initial certificate to authenticate further requests. In the case of the Internet-enabled toaster, further operations can now be carried out off the private network, with communications authenticated using the certificates exchanged during the PKIBoot stage. Further certificates can be now obtained, and existing ones updated at any time.

Rather than designing yet another new PKI protocol (the IETF PKIX working group currently has 21 RFCs and 30 RFC drafts in the works, with several being small books over 100 pages long, for an incredible 1,600 pages of X.509 PKI standards), our intent is to make use of existing protocols and mechanisms wherever possible (although no standard exists that provides details on the basic act of initialising a PKI via a PKIBoot-type mechanism, there do exist RFCs covering issues such as how to add animations and theme music to an X.509 certificate [26]). An additional goal, not stated explicitly above, is to aim for a solution which is practical and workable (a single paper suffices to cover it), rather than the ultimate perfect certificate management mechanism (260 pages of RFCs in 4 parts with 6 additional RFC drafts in progress and a 200-message thread arguing over what colour to paint it when it's finished).

3.1 PKI Service Location

The first requirement is for a protocol to locate PKI services. In the comparison with signing up for an ISP account or connecting to a corporate network that was used in section 2, mention was made of the use of DHCP for locating IP-related services, so the obvious solution would be DHCP. However, there are also a number of other service-location mechanisms that should be considered, of which the major ones are Jini, UPnP, the Service Location Protocol (SLP), and perhaps a few lesser-known ones such as Salutation.

3.1.1 DHCP

The Dynamic Host Configuration Protocol [27] provides a framework for machines on a network to obtain information such as their IP address, subnet mask, and default gateway. Apart from having computer-literate relatives, DHCP is the primary mechanism used to make IP automagically work for users.

Use of DHCP for PKI service location has several problems. Many networks already contain DHCP servers, and don't take kindly to the appearance of a second server: "The diversity of hardware and protocol implementations in the Internet would preclude reliable operation if random hosts were allowed to respond to DHCP requests" [27]. Overloading existing servers for use for PKI service location would represent a nightmare for sysadmins and home users, who would be faced with reconfiguring an arbitrary variety and number of DHCP servers, many of which don't allow user configuration of the kind required for our purposes. Finally, DHCP functions at a very low level, while the PKI services can assume that an IP networking infrastructure is already up and running, doing away with the need for the low-level DHCP approach. All of these issues make DHCP unsuited for our purposes, apart from providing the bare plug-and-play conceptual model for our design.

3.1.2 Jini

Jini is used to locate and interact with Java-based services [28][29]. It has the disadvantage that it is tied to a particular programming language and philosophy, and requires a large amount of Java-specific mechanisms such as Java object serialisation, RMI (Remote Method Invocation), and code downloading (the ability to move Java objects between JVMs) in order to function. In addition it provides a large range of client/server communications services

that are unnecessary for our purposes, since our intent is to use a standard PKI protocol if possible rather than implementing our own one using Java. This makes Jini unsuited for our purposes.

3.1.3 UPnP

UPnP is XML's (or perhaps Microsoft's) answer to Jini [30][31]. Like Jini, it provides a mechanism for locating and interacting with services over a network. Also like Jini, the protocol is very complex, employing XML (SOAP) over HTTP [32], and provides a large range of capabilities such as the ability to query devices down to the level of manufacturer details, serial numbers, and UPC codes, an HTML GUI interface, URLs for assorted control interfaces, and so on and so on. In addition the protocol is peer-to-peer and aimed mainly at home environments, allowing devices such as printers, scanners, and cameras to communicate with a computer and each other without requiring tedious user configuration.

There exists a service-discovery subset of UPnP, the Simple Service Discovery Protocol (SSDP), which operates on HTTP over UDP and restricts itself purely to service discovery. This protocol provides more or less the same services as SLP (see section 3.1.4) and bootstraps directly into SOAP/GENA (the UPnP General Event Notification Protocol), so it's not really useful as a standalone service location protocol, and in any case doesn't provide any real advantages over SLP. A more pragmatic consideration is that sites may block UPnP from leaving the LAN or disable it at the host for security reasons, in the same way that they currently block/disable NetBIOS. This would make services such as PKIBoot collateral damage even though it isn't really a UPnP service. All of these considerations make UPnP unsuited to our purposes.

3.1.4 SLP

The Service Location Protocol (SLP) is an IETF standards-track protocol designed to function in a language and technology-neutral manner [33][34][35][36]. SLP is purely a lookup service that provides a service location mechanism and then gets out of the way, which appears to make it an ideal choice for a PKI service location mechanism. However, there are some issues that make it slightly less than ideal. Firstly, the protocol is quite complex to implement, although this is mitigated to some extent by the existence of a freely-available reference implementation [37]. A more significant concern is that SLP is still waiting for its killer application (PKI is unlikely to be that killer application), so there is little deployment or even knowledge of its existence. In order to avoid requiring sysadmins to install, configure, and maintain yet another network service, we allow for the use of SLP where available, but also provide a simple alternative in section 3.1.6.

3.1.5 Others

There exist a variety of other, lesser-known service discovery protocols such as Salutation [38] and Apple's Rendezvous [39]. Although these have advantages for particular environments (for example Rendezvous brings AppleTalk-style ease of use to TCP/IP environments, which is of benefit to Macintosh users), they have no particular advantage over SLP for our purposes. In addition there are efforts under way to provide bridging for some of the more common service location protocols, such as having Salutation able to query SLP servers [40]. It appears that SLP (if it takes off) will be the common-denominator protocol.

Another possible alternative is the use of DNS SRV records [41]. Unfortunately the operating system used by the user group most in need of handholding when it comes to technical issues has no support for anything beyond the most basic DNS address lookups, making it impossible to use DNS SRV with anything but very recent Win2K and XP systems. To make things even more entertaining, several of the function names and some of the function parameters changed at various times during the Win2K phase of development, and the behaviour of portions of the Windows sockets API changed in undocumented ways to match. This leads to the unfortunate situation in which a Unix sysadmin can make use of DNS SRV to avoid having to deal with technical configuration issues, but a Windows'95 user can't. As a result, we can't usefully rely on SRV for our needs.

3.1.6 Faking it

In the absence of any formal service-location mechanism, it's possible to fall back to the use of a "well-known" location constructed from the service provider's domain name or the device's IP address [42]. If the PKI service isn't provided locally, HTTP type 3xx redirection or DNS-based redirection may be used to redirect clients to the actual PKI service provider. Use of this means of service location has the significant feature that it can be managed

through a single line of DNS or HTTP configuration data added to existing services/servers, making it by far the easiest option to deploy of any of the service location mechanisms — the addition of a single DNS entry for, for example, `pkiboot.aol.com`, would instantly provide plug-and-play PKI facilities for the entire AOL user base, with no further configuration necessary[2].

When used with full-scale DNS-enabled IP, the well-known location is constructed by prepending "pkiboot" to the domain name to obtain the URL to use for PKI services. For example if the current operating environment has a domain `myorg.org` then the PKI service would be accessed as `http://pkiboot.myorg.org`. When the certificate service is being provided in a more limited form such as with a web-enabled embedded device, the access location is given by appending a fixed path portion "pkiboot" to the device's IP address or location. For example if the embedded device providing the service is available at 192.0.0.1 then the PKI service on the device would be accessed as `http://192.0.0.1/pkiboot/`.

In practice we can use an extra level of indirection to locate PKI services, since PKIBoot isn't the only service we require, and the services may be distributed across multiple servers for performance, maintainability, or security reasons. For example the server that publishes certificate revocation lists (a small number of very large files, often several MB in size) has very different operational characteristics than the one that makes certificates available (a large number of very small files), which in turn has very different security characteristics than the one that provides certificate status information by directly querying the CA certificate store.

Since this is the part of the system that is the most likely to be maintained manually by an overworked sysadmin, we make it as simple as possible: A set of entries matching the required PKI service to the server URL that provides it, one service per line. This is in effect a crude form of DNS SRV functionality that can be made to work with versions of Windows earlier than XP. Note that no authentication is necessary at this preliminary pre-PKI level — the one thing that a PKI facility doesn't lack is endless capability for authentication and authorisation at every stage of the process.

A possible alternative exists in the form of the authorityInfoAccess (AIA) and subjectInfoAccess (SIA) certificate extension, which can be used to indicate the location of various certificate services. Since URLs can be fairly ephemeral while certificates tend to persist forever, experience has shown that these URLs invariably point to servers that no longer exist (at one point it was suggested that certificate standards be reworded to require that certificates contain only invalid URLs, to match existing practice [43]), not helped by the fact that the entity that issues the certificates usually has different goals from the one that runs the servers that provide various associated services. Since no CA will reissue all of its certificates simply because a server URL has changed, we can't depend on the AIA/SIA extension, although we can opportunistically use it if it's present and actually valid.

3.2 PKIBoot

There currently exists no protocol or mechanism for this fundamental PKI operation. Since one of our design goals is to use existing message formats if at all possible rather than inventing yet another new PKI protocol (see section 3), we need to locate an existing mechanism that can be adapted to our needs. This task is made easier (or perhaps more complex) by the fact that every new PKI protocol seems to invent its own message format incompatible with that of all other protocols, providing us with over a dozen different ones to choose from. All of these implement simple request/response mechanisms that provide more or less the same functions, so in theory any would fit the bill. In order to make things easier, we focus only on the two that are specifically designated as general-purpose certificate management protocols, with a discussion of their relative merits provided in the next section, section 3.3.

In order to support the PKIBoot operation, we use the GenMsg (General Message) facility of the Certificate Management Protocol (CMP, discussed further in section 4.3) to provide the certificate setup operation shown as the second stage of Figure 1 (CMP also includes a kitchen-sink field in the message header, but this appears to be dedicated to workarounds for problems in the initial version of the CMP protocol, so we use a GenMsg instead. In addition, the CMP RFC at one point appears to make a reference to a vaguely PKIBoot-type of operation, but never defines any messages or data types to support it, so this can't be implemented as a standard part of the protocol). A PKIBoot request consists of a CMP GenMsg containing a request for the initial set of trusted certificates, authenticated with a MAC (Message Authentication Code, a cryptographic checksum derived from the user's enrolment password) for new users or a digital signature for existing users who already have a certificate.

[2] The question of whether exposing AOL users to certificates is a sound idea is beyond the scope of this paper.

The CA responds to the PKIBoot request with the initial trusted certificate set, also protected with a MAC or a signature. In this way the user will accept only a restricted set of known-good certificates from a cryptographically authenticated authority, rather than an arbitrary collection of certificates included by whoever supplied their web browser or PC. This authentication process also eliminates the need for cryptographically protecting the service discovery stage that was covered in section 3.1, since only the genuine CA will be able to generate the MAC/signature necessary to authenticate the initial certificate set (the CMP protocol includes additional features such as nonces to prevent replay attacks, transaction IDs to link messages, and so on. Interested readers are referred to the CMP specification for more information).

3.2.1 PKIBoot for PKI-enabled Toasters

The design presented here works just as well for Internet-enabled toasters as it does for devices with human users through the use of the baby-duck security model [44][45]. In this model, a newly-initialised device (either one fresh out of the box or one reset to its ground state) imprints upon the first device it sees in the same way that a newly-hatched duckling imprints on the first moving object it sees as its mother. In our case the device trusts the first entity it meets merely to issue it a certificate, not unconditionally as in the original baby-duck model. The process is shown in Figure 2.

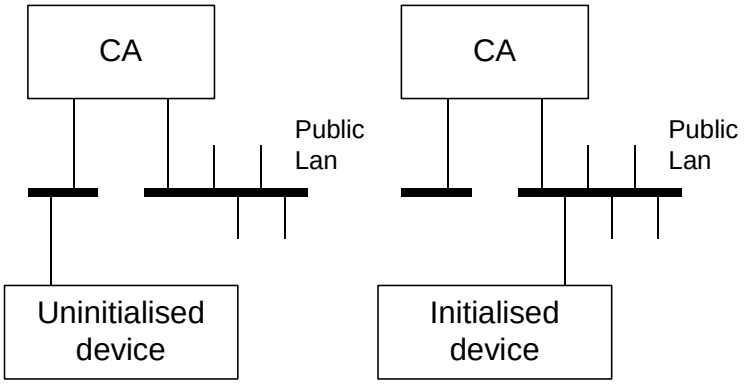

Figure 2: PnP PKI setup for an embedded device, with PKIBoot (left) and further setup (right)

Initially, the device is in the uninitialised (or reset-to-zero) state. Upon being connected to a network for the first time, it goes through the PKIBoot process to obtain its CA certificates and an initial certificate for itself, as shown in the left half of the diagram. This initialisation is carried out on a private LAN dedicated to device initialisation. Once the device has imprinted on the certificate server, it is moved to the public LAN (or WAN) and carries out any further operations there, as shown in the right half of the diagram. Note that we don't touch on the issue of who the device should trust (this is a making-PKI-workable paper, not an embedded device security paper). Readers are referred to the original work on the baby-duck security model for more coverage of this area [44][45].

Although Figure 2 shows a single server providing service for both sides, this is merely a convenience; the private network segment used for initialisation may have a server dedicated purely to device initialisation. Consider for example the case of an engine management system (EMS) of a car using the baby-duck PKI model. The initialisation stage is carried out at the manufacturer before the car is shipped to dealers. Once in the field, the EMS can use the initial certificate set to verify that upgrades are only performed by authorised service agents, and use its certificate to authenticate itself to the service agent[3]. Similarly, a technician installing PKI-enabled devices in the field could carry with them a laptop dedicated to PKIBoot device initialisation. After the device is first installed, it is initialised through a direct connection to the laptop. Once the initialisation phase has been completed, the device is connected to the general network where all further operations take place.

[3] As a side-benefit of these measures, an army of hot-chip enthusiasts will provide the system with considerable penetration testing at no cost to the vendor.

3.3 Obtaining Certificates

There exist two different protocols for certificate management, Certificate Management Protocol or CMP [46] and Certificate Management Messages over CMS or CMC [47]. Both are quite complex, and their main difference is that CMP has the curious design goal of being deliberately incompatible with any standard message format (really!) [48] while CMC was designed to use existing, well-established formats [49]. Some of the consequences of the CMP design decision are examined in more detail in sections 4.3 and 5.2. At the moment, CMP seems to be winning (in the sense that CMP is rarely used while CMC appears to be unused), so we employ a carefully-profiled subset of CMP here.

4 Implementation

This section looks at the implementation details of the various protocols and mechanisms chosen in section 3. The section that follows examines specific issues that cropped up during the process, and analyses the results of the implementation.

4.1 PKI Service Location

Implementation of the service location step was simple and straightforward, both for SLP and for the DNS/HTTP-based alternative. SLP required building the OpenSLP server and client and configuring the server with a simple template for PKI services, accessible via the SLP URL `service:pki.test:http://certificates.-myorg.org`. This URL can be built automatically using the rules from section 3.1.6. The alternative option, using DNS and/or HTTP to provide an equivalent service, does more or less the same thing, but without requiring SLP as an intermediary.

This illustrates the major problem arising from the lack of SLP deployment mentioned in section 3.1.4: SLP is a type of directory service which assumes that users will contact a well-known, central directory service (a Service Agent or Directory Agent in SLP terminology) to handle requests for particular services. This works well when SLP is in general use for locating all manner of services, but when the only service in use is the PKI service, having to find the SLP service in order to find the PKI service is an unnecessary complication. Although there is a facility for locating SLP services via DHCP [50], this has the same problems as using DHCP to directly locate the PKI service that were discussed in section 3.1.1 (SLP can also use UDP multicast for discovery, but this isn't a general-purpose solution since its range usually extends only as far as the local network segment).

In addition, SLP is a powerful, general-purpose service location protocol capable of responding to very specific requests such as "Where do I get Slowaris drivers for a WalletBuster 5000 colour printer?", when all we really need is "Where's the certificates?". Because of all of these issues, in practice it's easier to perform the PKI service location via the DNS/HTTP mechanism than via SLP, although we retain the ability to use SLP if it becomes generally adopted (this is no doubt the same excuse being used by every other not-quite user of SLP).

4.2 PKIBoot

The PKIBoot implementation was quite straightforward, and required only a relatively minor modification of the CMP implementation in the cryptlib toolkit [7] to handle the new GenMsg subtype. When in use, the certificate server is initialised with a set of trusted certificates that, at a minimum, includes its own certificate(s) and/or the certificates of the CA on whose behalf it operates. This operation is handled in a fairly straightforward manner as part of cryptlib's role-based access control system.

cryptlib recognises multiple user roles that are assigned different capabilities, one of which is a CA role authorised to issue and revoke certificates. Each user role has a variety of parameters associated with it, one of which is a collection of certificates used/trusted by the user. When the server receives a PKIBoot request, it queries the CA user for its trusted certificates, and sends these as the response to the PKIBoot query. The server verifies an incoming request using the authentication information it has stored for the user making the request, and responds with the initial trusted certificate set, exactly as described in section 3.2.

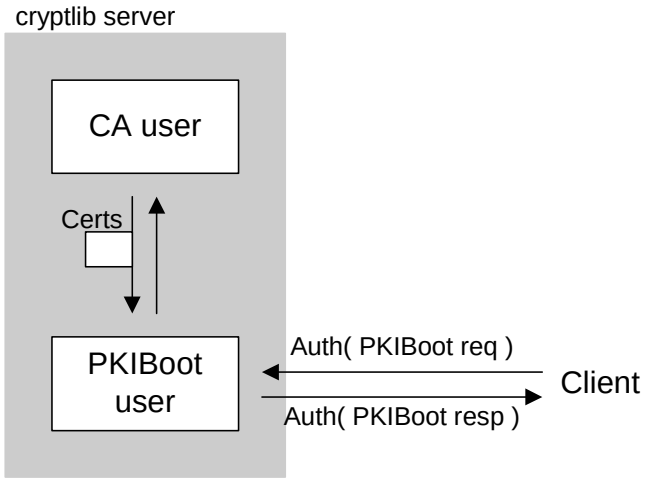

Figure 3: Role-based actions during the PKIBoot process

Because of the automated role-based handling of certificates (the CA will, at an absolute minimum, have its own certificate(s) trusted), there is no need to perform any explicit actions to manage trusted certificates. In effect, the PKIBoot process allows a client to perform a remote query of trusted certificates from the CA user, implemented using standard PKI mechanisms. The process is illustrated in Figure 3. In this case the cryptlib PKIBoot user is acting as a proxy for the cryptlib CA user, however it can also act as a proxy for another cryptlib PKIBoot user, for example when a departmental server relays certificates from an organisation-wide server.

The list of trusted certificates is communicated in the standard CMS format accepted by any PKCS #7/CMS/-SMIME implementation [49], and by extension virtually every certificate-aware application. Although the transport protocol we're using is CMP, it can be in interpreted in a manner that allows the same bits-on-the-wire format as CMS (see section 5.3 for details on how this is possible), so that the certificate payload can be used by anything that understands CMS certificate chains, which in practice includes pretty much everything that uses X.509 certificates. This means that any certificate-using application can be PKIBoot-enabled through the use of a CMP front-end to handle the messaging.

There is a possible alternative format, Microsoft's Certificate Trust List (CTL) [51], but this only contains a hash of the certificate rather than the certificate itself (it's assumed that the certificates are distributed via some unspecified mechanism), contains additional unnecessary information such as policy identifiers alongside the hash, and most importantly contains a number of undocumented details and is apparently covered by a Microsoft patent. CMS predates CTLs by many years and is well-defined, widely-supported, and unencumbered by patents.

The entire process beyond defining the contents of the trusted certificate set is quite transparent to both the server administrator and the user, being handled automatically by the (modified) CMP server and cryptlib's role-based access control system. From the users' point of view, the PKIBoot process is simply a part of the initial certificate exchange and doesn't require any explicit actions or effort, being protected by the security mechanisms used in the initial certificate exchange (see the next section).

4.3 CMP

The final stage of the process consists of obtaining the user's certificate (or subsequent certificates). At this point the software (or hardware, if a crypto token such as a smart card is being used) generates a public/private key pair for the user and uses it to obtain the initial certificate. This is the first point at which the user needs to perform any explicit actions, namely entering their user name and password, just as they'd be required to enter this information when signing on to an ISP account or connecting to a corporate network. The same information was also used to authenticate the PKIBoot exchange, but as mentioned in the previous section this occurs as a transparent addition to the CMP process, so the user isn't required to provide additional information for the PKIBoot process or even know that it's occurring, in much the same way that DHCP sets things up for IP operation. There may also be other supplementary actions required at this point, for example if the CA charges for certificates issued then the user

should be required to acknowledge that the operation that follows will result in some form of monetary charge being applied.

Although the discussion so far has referred to user names and passwords, in practice these values are typically text strings identical in format to those used in software product activation keys, a shared-secret form that most (Windows) users will be familiar with. These contain around 90 bits of entropy, which in combination with the use of a salted iterated hash for the MAC makes offline guessing attacks infeasible. Even if a standard password is used, the enrolment step is a one-time operation (further steps are authenticated using the certificate obtained during the enrolment phase), making a password obtained through an attack on observed MAC values in the initial exchange of little value to an attacker.

The use of a one-time password during the enrolment phase provides a further measure of protection in that if an attacker can somehow obtained the password before the exchange takes place rather than afterwards (for example by stealing it at the source), the user will be warned of the compromise when their attempt to obtain a certificate fails with an indication that the password has already been used.

The enrolment process, while in theory probably the simplest part of the plug-and-play PKI operation, is in fact the most awkward, being considerably hampered by the extraordinary complexity and ambiguity of the CMP protocol specification. The problems encountered are covered in more detail in the discussion of implementation issues in section 5.2, with resolutions provided in section 5.3.

5 Discussion

The section examines some of the major issues that were encountered in implementing the operations covered in section 3.2.1, and concludes with a real-world acid test of the resulting system that examines whether it can live up to the claim of being a PKI your mother can use.

5.1 Service Location

In practice the use of SLP, while no doubt the correct thing to do, is somewhat impractical. An informal survey of sysadmins indicated that they would be reluctant to have yet another network service running on their servers, not helped by the fact that there were easier alternatives available. In the end, the use of HTTP-based mechanisms prevailed.

5.2 Problems with CMP

CMP is an extremely complex protocol with a large number of ambiguous, redundant, or even incomprehensible features (there were some protocol features that were removed at one stage when even the RFC authors couldn't explain their purpose). In many cases the function of various protocol elements is unknown, and so implementors have to guess at their purpose. Needless to say, everyone guesses differently. For example, there are three different fields that function as nonces, with no-one certain as to their exact purpose or use (one interop report states that "no realistic replay attack was identified, it is not clear if nonces are required in [message types]" [52]). As a result, some implementations implement them, some don't, and some ignore them if they find them.

There is another nonce field described as a transaction ID that appears to be the most obvious place for a nonce, however some implementations use this field to identify the sender of the message. Others omit it (it's optional) and use the sender key ID field (also optional, and omitted by some implementations) to identify the sender. Still others use the sender X.500 Distinguished Name (DN) field for identification. This is the only mandatory ID field, but when obtaining their first certificate the user doesn't know their DN yet (in fact most users don't know their DN at any point, even when they have a certificate), so the field must be filled with a dummy value, which is rejected by some servers that (somehow) require the user to specify a DN in the initial request. Other servers accept any arbitrary DN, but return a certificate with a totally different DN of their own choosing, leaving the reason for needing to supply a DN in the first place open to speculation. To add to the confusion, there is a second DN present in the request itself that may or may not match the sender DN, and the sender DN may itself be overridden by the sender key ID, if present.

If pure DN-based identification is used, it's not possible to uniquely identify the certificate to be used to authenticate a CMP transaction, since a DN can only identify the overall owner of a group of certificates, but not an individual certificate. This problem was demonstrated by one CMP CA who updated their CA certificate, leading to the

existence of multiple certificates identified by the same DN, so that all signatures created by the CA mysteriously failed to verify, with no easily discernable reason (this is purely a fault of the CMP design, existing data formats such as CMS/PKCS #7 don't have any of these problems).

Further complications abound. Many fields are marked as mandatory, but never used, so special values have to be invented to fill them, with no other meaning than to indicate that their presence has no meaning. Other fields are marked as optional but are in fact mandatory, causing the protocol to break if they are omitted (because of the confusion over the purposes of many of the fields, it's not easy to determine this in advance). There are portions of CMP that serve no identifiable purpose (for example "what is the purpose of [part of protocol]? When should it be checked? What attack(s) does it protect against? Should it always be there or is it used in lieu of other mechanisms?" [52]), with the result that implementations omit them in order to reduce complexity, leading to lively debates over standards-compliance when a new implementation which does support the feature appears (during one interop session, interoperability among implementations was actually *worse* than it had been at the previous year's interop). Some protocol features were in fact moved from mandatory-to-support to optional when it was found that no-one had bothered to implement them.

In summary, trying to work with CMP was without doubt the most frustrating part of implementing the plug-n-play PKI design. Although it's easy enough to manage in a homogeneous environment, trying to achieve interoperability among multiple different implementations is at best difficult and at worst impossible, for example when one implementation requires that an X.500 DN be provided before the user knows what it is. The only mitigating factor here is that any CMP implementation configured to use the PKIBoot process will presumably be able to handle the necessary exchange of messages, leading to a de facto PKIBoot-capable common interpretation of the CMP specification.

5.3 Fixing CMP

In order to increase the chances of having two CMP implementations able to interoperate, we use an interpretation of the protocol designed to minimise problems. To work around the confusion involving identifiers, we add an extension containing an ESSCertID (a universal, unambiguous certificate identifier containing a hash of the certificate in question) [53] to all messages. This is possible because CMP uses a peculiar defunct form of ASN.1 which is imprecise enough that we can embrace and extend the protocol to use this type of identifier. This also allows us to use CMP to communicate standard PKCS #7/CMS certificate chains usable by any other certificate-using software as described in section 4.2, even if it doesn't talk CMP.

The use of the ESSCertID does away with the need to work with the unreliable CMP identification mechanisms. If there's no ESSCertID present, we take guesses at the various identifiers mentioned in section 5.2, trying them as key identifiers, user identifiers, and X.509 subject key identifiers (an X.509 certificate extension type). The X.500 DNs, which may be present, present but set to a special value to indicate that they're not present, or arbitrarily changed by the other side, are ignored since they're more trouble than they're worth.

The optional fields labelled as nonces are copied across into replies in case the other side pays attention to them, but otherwise ignored in order to interoperate with implementations that don't use them. The main nonce, labelled the transaction ID (although some implementations overload it as a user ID) is treated as the principal nonce, unless it's absent, in which case we fall back to the other nonces if they are present (as this description implies, the processing logic for CMP messages tends to be quite complex). Because of the uncertainty about nonces, the cryptlib CMP implementation is designed to function without them if necessary, employing a database with full transactional capabilities to record every operation so that an attempt to replay (for example) a certificate issue request will be detected by the presence of a duplicate entry in the database [54].

In practice then we use the ESSCertID to identify the certificate or key used for authentication of messages and the transaction ID (with a failsafe in the transaction processing system) as the nonce, falling back through a variety of alternatives if one or both are missing. If a certain minimum standard isn't reached (for example all of the various nonce fields are absent, or none of the various identifier fields appears to identify a certificate or key), we abort the processing. As with the assumption about PKIBoot capabilities in section 5.2, it's safe to assume that anything capable of handling PKIBoot will also handle identifiers and nonces in a sensible manner.

5.4 Testing

The final step in the process was to test the implementation to determine whether this really was "a PKI your mother can use" (the author resisted the temptation to cheat slightly and run the client in baby-duck mode, which would have allowed him to claim that he had a PKI his potted plants could use). The test system consisted of a Windows '98 home PC, with an HTTP redirect leading to a PKIBoot/CMP server located on the other side of the world, partially to prove that remote certification services were just as feasible as local ones, and partially just because it was convenient.

Taking a recently-calibrated reference mother, the system was stress-tested to determine usability by non-technical users. The whole process functioned exactly as expected, with the end result being a freshly-minted certificate stored in the system without the user being aware that it had happened. The implementation actually obtains two certificates, first a long-term signature certificate authenticated with a MAC derived from the user name and password, and then a short-term encryption certificate authenticated with the just-obtained signature certificate, providing the necessary separation of encryption and signature keys which is so often sacrificed for ease-of-use when obtaining certificates using conventional mechanisms. This process is totally transparent to the user, as is the fact (or even the need to know) that two different keys are in use. A similar process is used in OpenPGP [55], which maintains Elgamal encryption keys authenticated with DSA signing keys.

A second test was used to verify functionality when run in baby-duck mode, with an old headless Linux box acting as a stand-in for the Internet-enabled toaster. This test was pretty much a tautology: the system went through the PKIBoot and certificate-acquisition process, and subsequently initiated an SSL connection authenticated from the server side with a trusted certificate exchanged during the PKIBoot process and from the client side with its newly-obtained certificate. In the real world, this would presumably be followed by the downloading of control software or configuration data over the SSL-secured link. In contrast in a world without plug-and-play PKI services, SSL server authentication is performed using a hardcoded arbitrary collection of mostly-unknown CA certificates, and client authentication is omitted because it's just too hard to manage.

In attempting to design a security protocol with usability as a major goal, it's not possible (without access to the usability testing labs of a large software company) to immediately determine whether this goal has indeed been met. The intent of this paper was to create a design that makes the process as simple as possible. A more general test than the previous two, in the form of ongoing use in cryptlib by general users, is currently under way. To some extent the design has been vindicated by the fact that at least one beta-tester wanted to move the initial experimental implementation into production immediately, but in the long run only time will tell how successful the overall design is.

In conclusion, the system passed the initial tests, providing a fully automated means of acquiring (and later updating/replacing) certificates usable by even the most inexperienced user, or as part of an automated process requiring no user intervention. In addition, the system enforced best-practice key usage without the user having to be aware of this — they weren't even aware that they were using certificates, and were quite amazed when informed that this had achieved significantly more than what was achieved by the tortuous process covered in section 1.

6 Conclusion

This paper has examined the design process involved in making the various steps of establishing and using a PKI simple and automated enough that anyone can use it, including embedded devices that can't rely on manual intervention by a human to provide them with the necessary configuration. What appears at first glance to be a relatively simple concept is complicated by the lack of established standard(s) to help accomplish the task, and the high level of difficulty encountered in working with the standards created to allow certificate management operations. In contrast, the initial part of the PKI setup operation suffers from an embarrassment of riches, with the eventual choice of mechanism being driven by ease-of-use and ease-of-deployment considerations even if it may not be the most appropriate on purely technical grounds. Despite these difficulties, it proved possible to use (and in some cases slightly abuse) existing standards to accomplish the task at hand, without requiring the creation of yet another set of PKI standards to perform the task, with important portions of the protocol output (for example PKCS #7/CMS certificate chains sent over CMP) directly usable by existing applications.

7 References

[1] "In Search of Usable Security: Five Lessons from the Field", Dirk Balfanz, Glen Durfee, D.K.Smetters, and Rebecca Grinter, *IEEE Security and Privacy*, **Vol.2**, **No.5** (September/October 2004), p.19.

[2] "Peer to Peer: Collaboration and Sharing over the Internet", Bo Leuf, Addison-Wesley, 2002.

[3] "Network Security with OpenSSL", John Viega, Matt Messier, and Pravir Chandra, O'Reilly and Associates, June 2002.

[4] "OpenSSL: The Open Source toolkit for SSL/TLS", `http://www.openssl.org`.

[5] "Re: Purpose of PEM string", Doug Porter, posting to pem-dev mailing list, message-ID `93Aug16.003350pdt.13997-2@well.sf.ca.us`, 16 August 1993

[6] "The Design of a Cryptographic Security Architecture", Peter Gutmann, *Proceedings of the 8th Usenix Security Symposium*, August 1999, p.153.

[7] "cryptlib Encryption Toolkit", `http://www.cs.auckland.ac.nz/~pgut001/cryptlib/index.html`.

[8] Lucky Green, private communications.

[9] David Harris, private communications.

[10] "Gateway Guardians", Fred Avolio, *Information Security*, **Vol.6**, **No.2** (February 2003), p.51.

[11] "1st Annual PKI Research Workshop", `http://www.cs.dartmouth.edu/~pki02/`.

[12] "Amusing note on real-world PKI deployment", Peter Gutmann, posting to the cryptography@wasabisystems.com mailing list, message-ID `200201251007.XAA283003@ruru.cs.auckland.ac.nz`, 25 January 2002.

[13] "Santa Barbara Health Exchange: User-Driven Security Prevails", Greg Goth, *IEEE Security and Privacy*, **Vol.2**, **No.5** (September/October 2004), p.8

[14] "Computer Communications Security: Principles, Standard Protocols and Techniques", Warwick Ford, Prentice-Hall, 1994.

[15] "Network Security: Private Communication in a Public World (2nd ed)", Charlie Kaufman, Radia Perlman, and Mike Speciner, Prentice-Hall, 2002.

[16] "Security in Computing (3rd ed)", Charles Pfleeger, Shari Lawrence Pfleeger, and Willis Ware, Prentice-Hall, 2002.

[17] "Lessons Learned in Implementing and Deploying Crypto Software", Peter Gutmann, *Proceedings of the 11th Usenix Security Symposium*, August 2002, p.315.

[18] "The Crypto Gardening Guide and Planting Tips", Peter Gutmann, January 2003, `http://www.cs.auckland.ac.nz/~pgut001/pubs/crypto_guide.txt`.

[19] "Good-Enough Security: Toward a Pragmatic Business-Driver Discipline", Ravi Sandhu, *IEEE Internet Computing*, **Vol.7**, **No.1** (January/February 2003), p.66.

[20] "Online Authentication", `http://www.e-government.govt.nz/authentication/index.asp`.

[21] "Zero-configuration Networking", Eric Guttman, *Proceedings of INET 2000*, July 2000.

[22] "Autoconfiguration for IP Networking: Enabling Local Communication", Eric Guttman, *IEEE Internet Computing*, **Vol.5**, **No.3** (May/June 2001), p.81.

[23] "Zero Configuration Networking (zeroconf)", `http://www.ietf.org/html.charters/zeroconf-charter.html`.

[24] "RE: IP: SSL Certificate "Monopoly" Bears Financial Fruit", Lucky Green, posting to the cryptography@wasabisystems.com mailing list, message-ID `002901c228b4$14522fb0$6501a8c0@LUCKYVAI0`, 11 July 2002.

[25] "A rant about SSL, oder: die grosse Sicherheitsillusion", Matthias Bruestle, presentation at the KNF-Kongress 2002.

[26] "Internet X.509 Public Key Infrastructure: Logotypes in X.509 certificates", Stefan Santesson, Russell Housley, and Trevor Freeman, IETF draft, March 2003.

[27] "Dynamic Host Configuration Protocol", RFC 2131, Ralph Droms, March 1997.

[28] "Core Jini (2nd ed)", W.Keith Edwards, Prentice-Hall, 2000.

[29] "The Jini Specification (2nd ed)", Ken Arnold (ed), Addison-Wesley, 2000.

[30] "Universal Plug and Play Specifications", http://www.upnp.org/resources/specifications.asp.

[31] "UPnP Design by Example: A Software Developer's Guide to Universal Plug and Play", Michael Jeronimo and Jack Weast, Intel Press, 2003.

[32] "Simple Object Access Protocol (SOAP) 1.1", W3C Note, 8 May 2000, http://www.w3.org/TR/SOAP/.

[33] "Service Location Protocol, version 2", RFC 2608, Erik Guttman, Charles Perkins, John Veizades, and Michael Day, June 1999.

[34] "Service Templates and Schemes", RFC 2609, Erik Guttman, Charles Perkins, and James Kempf, June 1999.

[35] "An API for Service Location", RFC 2614, James Kempf and Erik Guttman, June 1999.

[36] "Service Location Protocol: Automatic Discovery of IP Network Services", Erik Guttman, *IEEE Internet Computing*, **Vol.3**, **No.4** (July-August 1999), p.91.

[37] "OpenSLP", http://www.openslp.org.

[38] "Salutation Consortium", http://www.salutation.org.

[39] "Rendezvous", http://developer.apple.com/macosx/rendezvous/.

[40] "Salutation and SLP", Pete St. Pierre and Tohru Mori, http://www.salutation.org/techtalk/slp.htm.

[41] "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, Arnt Gulbrandsen, Paul Vixie, and Levon Esibov, February 2000.

[42] "Internet X.509 Public Key Infrastructure Operational Protocols: Certificate Store Access via HTTP", Peter Gutmann, IETF draft, March 2003.

[43] "Re: Computation of issuerKeyHash in OCSP", Peter Gutmann, posting to the ietf-pkix@imc.org mailing list, 13 March 2001.

[44] "The Resurrecting Duckling: Security Issues in Ad-Hoc Wireless Networking", Frank Stajano and Ross Anderson, *Proceedings of the 7th International Workshop on Security Protocols*, Springer-Verlag Lecture Notes in Computer Science No.1796, April 2000, p.172.

[45] "The Resurrecting Duckling — What Next?", Frank Stajano, *Proceedings of the 8th International Workshop on Security Protocols*, Springer-Verlag Lecture Notes in Computer Science No.2133, April 2000, p.204.

[46] "Internet X.509 Public Key Infrastructure: Certificate Management Protocols", RFC 2510, Carlisle Adams and Stephen Farrell, March 1999.

[47] "Certificate Management Messages over CMS", RFC 2797, Michael Myers, Xiaoyi Liu, Jim Schaad, and Jeff Weinstein, April 2000.

[48] "Planning for PKI: Best Practices Guide for Deploying Public Key Infrastructure", Russ Housley and Tim Polk, John Wiley and Sons, 2001.

[49] "Cryptographic Message Syntax (CMS)", RFC 3369, Russell Housley, August 2002.

[50] "DHCP Options for Service Location Protocol", RFC 2610, Charles Perkins and Erik Guttman, June 1999.

[51] "Certificate Trust Lists: What Are They? Why Are They Useful?", Trevor Freeman, presentation at the NIST PKI Working Group meeting, 12 November 1998.

[52] "CMP Interoperability Testing: Results and Agreements", Robert Moskowitz, IETF draft, June 1999.

[53] "Enhanced Security Services for S/MIME", RFC 2634, Paul Hoffman, June 1999.

[54] "A Reliable, Scalable General-Purpose Certificate Store", Peter Gutmann, *Proceedings of the 16th Annual Computer Security Applications Conference (ACSAC'00)*, December 2000, p.278.

[55] "OpenPGP Message Format", RFC 2440, Jon Callas, Lutz Donnerhacke, Hal Finney, and Rodney Thayer, November 1998.