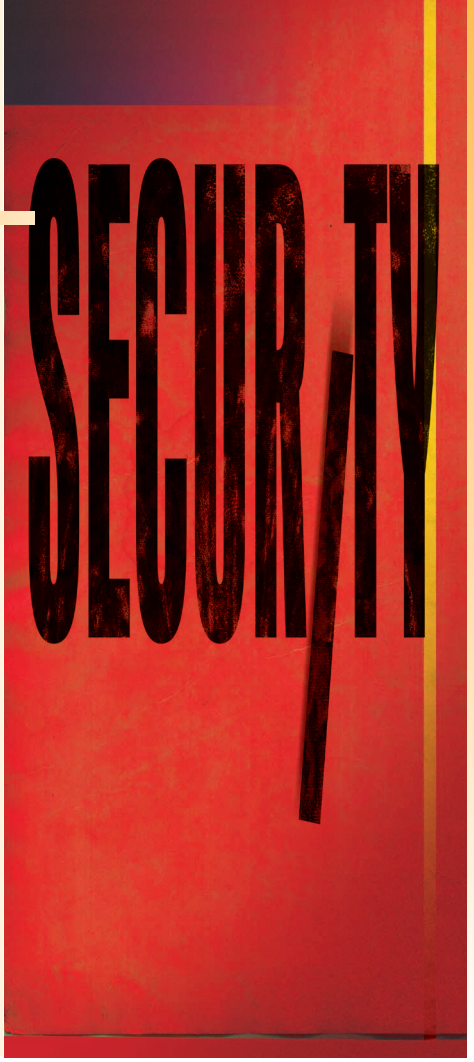
PKI: It's Not Dead, Just Resting



Despite an original design that failed to address the marketplace's needs, the use of innovative public key infrastructure models can make the technology meet today's requirements.

Peter Gutmann
University of Auckland

After some false starts, X.509 has slowly evolved into an extremely flexible public key infrastructure model, especially when X.509 proponents are describing its capabilities. However, like other flexible objects, PKI sacrifices some utility in trying to be all things to all people: Mainly, its generic, all-purpose identity certificates—issued by third-party certificate authorities—are not generally what the marketplace demands. Consequently, vendors continue to develop more economically efficient, useful, and imaginative business models.

X.509, originally designed to solve the problem of controlling access to an X.500 directory, does not address the problems that need solving *today*. This creates a severe mismatch because real-world business demands must be shoe-horned into the X.509 model to work with certificates.

The X.509 model's ties to X.500/LDAP directories, hierarchical structures, offline revocation, and other design decisions that stem from its X.500 origins further complicate the situation. Ideally, the model would instead use today's standard business tools and methods, such as relational databases, a nonhierarchical organization, and online validity and authorization checking.

The solution to these problems is to adapt the PKI design to the real world rather than trying to constrain the real world to match PKI. A variety of alternative approaches, ranging from simple workarounds to designing the application to sidestep the problem entirely, can help solve the inherent problems in the standard X.509 model.

PKI'S EVOLUTION

PKI's history dates back to Whitfield Diffie and Martin Hellman's seminal 1976 paper on public-key cryptography,[1] which proposed a Public File key directory that users could consult to find other users' public keys. Today, the Public File, which protected all communications by signing them, would be called a trusted directory.

Realizing some of this approach's shortcomings—including that it is a potential performance bottleneck, that it presents a tempting target for attackers, and that disabling access to the directory also disables users' ability to communicate securely—Loren Kohnfelder proposed the concept of certificates in 1978.[2] Certificates separate the signing and lookup functions by allowing a certificate authority to bind a name to a key through a digital signature and then store the resulting certificate in a repository. Since the repository no longer needs to be trusted and can be replicated, made fault tolerant, and given various other desirable properties, the CA approach removes many of the problems associated with a trusted directory.

0018-9162/02/$17.00 © 2002 IEEE

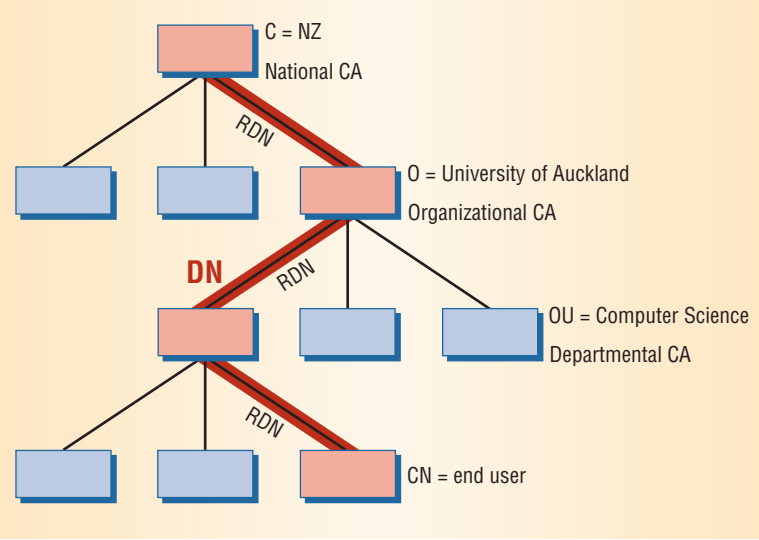


Figure 1. X.500 directory and certificate model. A different certificate authority (CA) is attached to each part of the directory to manage access control, while relative distinguished names (RDNs) define the path through the directory and together form a distinguished name (DN).

X.509

Some years after Kohnfelder published his thesis, developers incorporated certificate use into X.500, a global directory of named entities administered by monopoly telecommunications companies. The X.500 directory proposed a hierarchical database model, with the path through the directory being defined by a series of relative distinguished name (RDN) components that together form a distinguished name (DN). To protect access to the directory, its designers proposed various access-control mechanisms, ranging from simple password-based measures to the then relatively novel approach of using digital signatures. Each portion of the directory had CAs attached to it that issued access-control certificates, as Figure 1 shows.

The original X.509v1 certificate structure had an issuer DN and a subject DN to place the certificate in the directory, a validity period, and a public key. However, it lacked any indication of

- whether the certificate belonged to a CA or an end entity since this information is implicit in the directory;
- what the key in the certificate could be used for—there was only one use, directory authentication;
- what policy the certificate was issued under—again, there was only one policy, for authentication; or
- any of the other information without which no current certificate can be complete.

Although no real directories of this type were ever deployed, PKI designers and users have been forced to live with the legacy of this approach.

The main conceptual problem with X.509 certificates is that they turn simple public keys into *capabilities*, tickets that control access rights and that an end entity can use to demonstrate access to an object. Unfortunately, capabilities make access review—deciding who has access to what, since a capability can be easily passed on to others—and revocation extremely tricky. X.500 tried to address this problem through certificate revocation lists. CRLs are a digital analog of 1970s-era credit-card blacklists, which were in turn modeled after even earlier check blacklists. The standard, vague on how these checks should work, left all the details to the CA, the directory, or both. Other options included simply replacing the revoked certificate with the new one or notifying the certificate owner "by some offline procedure."[3]

Figure 2 shows the final form of the X.509 certificate usage model, in this case for general digital signature verification rather than directory authentication. The relying party, wishing to verify a signature, fetches a certificate from a repository, fetches the CRL from the same or another repository, checks the certificate against the CRL, and finally checks the signature against the certificate. Originally, developers intended the repository to be the X.500 directory. Today, in practice, it can be anything from flat files, a relational database, Berkeley DB, or the Windows registry, to a hard-coded local certificate or a certificate included with the data it authenticates. Only occasionally is the repository actually a directory.

Identity certificate problems

The abstract model, while simple, hides many problems. The phrase "fetches a certificate from a repository" reflects the biggest one. Because a global distributed directory never appeared, there is no clear idea where the system fetches a certificate or its CRL from.

Developers solved this well-known PKI issue, called the "Which directory?" problem, by including any certificates that might be needed wherever they might be needed. For example, an S/MIME signature usually includes all the certificates needed to verify it, and a secure socket layer server's communication to the client usually includes the certificates needed to protect that communication. A relying party can obtain a new certificate either by out-of-band means such as mailing a user and asking for the certificate or by a lazy update mechanism in which an application keeps copies of any certificates it may come across in case they're useful in the future. This approach solves the certificate distribution problem at the expense of transferring the load to an even harder problem: certificate revocation.

Even if the user knows which directory to look in, there's no way to determine which DN to use to

find a certificate, or which of several identical names being searched on belongs to the person whose key the user is seeking.

This raises another standard PKI issue, the "Which John Smith?" problem. As a result, the current version of X.509 turns a key distribution problem into an equally intractable name distribution problem. Although it's possible to disambiguate names through ad hoc measures such as adding the last four digits of a user's Social Security number to the DN, doing so results in a unique DN that is useless for name lookups because no third party will know how to construct it.

Other certificate models such as the Pretty Good Privacy and Simple Public Key Infrastructure/Simple Distributed Security Infrastructure (SPKI/SDSI) ones have their own approaches to the naming problem. SDSI recognized that globally unique names were only necessary in a few special cases. Instead, servers usually require only a name meaningful within a limited community. For example, while the name "John Smith" is essentially meaningless within the "USA" community, it is meaningful when we restrict the community to "people allowed to use this server." SPKI then paired SDSI names with the concept of using the public key as an identifier to provide global uniqueness.[4]

PGP solves the problem less rigorously but equally effectively by letting users choose any kind of identifier they want for certificates. These generally consist of an e-mail address and accompanying user name. Because the e-mail address is unique and PGP is used mostly for e-mail communications, this approach works reasonably well.

In effect, PGP and SPKI employ the same conceptual model, using a locally meaningful identifier within a specific domain. PGP implicitly sets the domain to "e-mail addresses," whereas SPKI implies the domain by defining the restricted community in which the certificate is used—to authenticate to a particular server, for example. Credit card and bank account numbers and Social Security and tax identifiers are other examples of such schemes. Although easily confused when presented without a disambiguating context, these identifiers are meaningful in an application's particular domain.

X.500 DNs, on the other hand, did not fit a real-world domain, and most users didn't understand them. As a result, except for a few carefully managed, centrally controlled schemes, users employed de facto local naming schemes, cramming anything into the DN until it became a mostly meaningless blob whose sole purpose was to uniquely identify a public key.

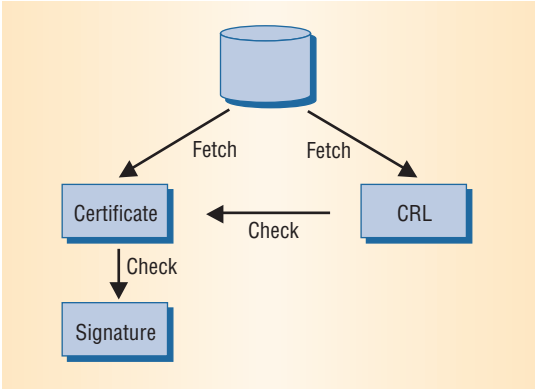


Figure 2. Final X.509 certificate usage model, used in this case for digital signature verification.

All three of these approaches, two by design and one by coincidence, eventually found the same solution of using a locally unique identifier such as an e-mail address and a user name within a specific domain. This solution mostly resolved the identification issue, which led to the next problem: revocation.

REVOCATION

Certificate revocation doesn't really work, particularly when implemented as envisaged in X.509.[5] The problem with CRLs is that they violate the cardinal rule of data-driven programming: Once you have emitted data, you can't take it back. Viewing the certificate issue–revocation cycle as a proper transaction-processing transaction, the certificate issue becomes a PREPARE command and the revocation becomes a COMMIT command. This means, however, that nothing can be done with the certificate in between the two commands because any action would destroy the transaction's atomicity and consistency properties. Allowing other operations with the certificate before the transaction has been committed results in nondeterministic behavior, which provides little value for relying parties.

CRL problems

Several problems make CRLs difficult to work with and unreliable as a certificate-status-propagation mechanism. Critical applications require prompt revocation—or, more accurately, real-time certificate-status information—and CRLs don't really solve this core problem, for several reasons.

Developers based the CRL concept on the credit-card blacklists used in the 1970s. Credit card companies periodically printed booklets of canceled card numbers and distributed them to merchants, expecting them to check any cards they handled against the blacklist before accepting them. The same problems that affected credit-card blacklists then affect CRLs today:

- the CRLs aren't issued frequently enough to be effective against an attacker who has compromised a card or private key,

In one widely used application, certificate-revocation-list checking caused every certificate operation to stall for a minute, after which it timed out and processing continued as before.

- distributing them is expensive,
- checking them is time-consuming, and
- a denial-of-service attack easily renders them ineffective.

When a CA issues a CRL, it bundles up a blacklist of revoked certificates along with an issue date and a second date indicating when the next blacklist will become available. The CA expects a relying party to fetch the current CRL and use it to check the validity of the certificate. In practice, this rarely occurs because users and applications don't know where to find a CRL, they put off using it until things grind to a halt, or they disable the function because fetching the CRL and checking the list takes too long. In one widely used application, CRL checking caused every operation that used a certificate to stall for a minute while the application groped around for a CRL, after which it timed out and processing continued as before.

CRLs suffer from several other practical problems. To guarantee timely status updates, the server must issue CRLs as frequently as possible. Yet issuing the CRL increases the load on the server and the network that transmits it and, to a lesser extent, on the client that fetches it. Issuing a CRL once a minute provides moderately timely revocation at the expense of massive overhead as each relying party downloads a new CRL. On the other hand, delaying the issuance to once per hour or day does not provide timely revocation.

CRLs also lack mechanisms for charging relying parties for checking revocation. When a CA issues a certificate, it charges the user a fee. The amount the CA charges is typically tied to how much checking it does before issuing the certificate. On the other hand, users expect CAs to create and issue CRLs for free. Neither the user nor the CA can say definitively who will validate the certificate, how often it will be validated, or what degree of latency will be acceptable. This situation serves as an active disincentive for CAs to pay much attention to CRLs because creating and distributing them requires processing time, one or more servers, and significant amounts of network bandwidth.

Who pays for general PKI operations is also a concern. For example, in the Secure Electronic Transaction (SET) PKI, the issuing bank carries the cost and associated risk of handling certificate enrollment and issue, while the acquiring bank obtains all the benefits. Although sharing the cost between both banks could mitigate this flaw, it provided one of the many nails that sealed SET's coffin.

Another approach charges a per-certificate-use transaction fee to cover the cost of running the PKI. In a US General Services Administration project, the cost ranged from $.40 to $1.20 for each transaction,[6] a considerable disincentive to certificate use. A more general CA practice is to bury the per-transaction costs elsewhere, although how well this will work when the PKI project leaves the pilot stage remains to be seen.

Proposed CRL workarounds

CRLs with built-in lifetimes all expire at the same time, forcing every relying party to fetch a new CRL simultaneously, which leads to huge peak loads whenever the current CRL expires. In effect, CRLs contain their own built-in distributed denial-of-service attack. Many solutions to this problem have been proposed, including

- staggering CRL expiry times for different certificate classes so that they don't expire simultaneously,
- overissuing CRLs so that multiple overlapping CRLs exist at one time,
- segmenting CRLs based on the revocation information's perceived urgency so that a CRL with a "key compromise" reason code would be issued more frequently than one with an "affiliation changed" reason code, and
- issuing delta CRLs that augment the main CRL.

There is little real-world experience with any of these mechanisms, although discussions on PKI mailing lists indicate that attempts to implement them will prove interesting.

SET uses one possible solution to the problem, taking advantage of certificates' ties to credit cards to avoid using CRLs altogether. SET cardholder certificates—which are expected to be invalidated relatively frequently—are revoked by revoking the card to which they're tied. Merchant certificates—which are invalidated less frequently—are revoked by removing them from the acquiring bank's database. Acquirer payment gateway certificates—which are seldom invalidated—are short-term certificates that can be quickly replaced.

This process takes advantage of existing mechanisms for invalidating certificates or designs around the problem so that PKI-like revocations are not needed. Account Authority Digital Signatures (ANSI X9.59) uses a similar scheme to design

around the problem. AADS uses a simple extension to existing account-based business infrastructures. The extension stores public keys on a server that handles revocation by removing the key. The most widely used key-management system for the secure shell works along similar lines, tying keys to Unix user accounts.

SPKI takes a slightly different approach by implicitly making validation part of the certificate-processing operation. SPKI prefers revalidation, which represents a positive statement about a certificate's validity, to CRLs, which represent a negative statement. This approach works because positive assertions are much more tractable than negative ones—consider, for example, the relative difficulties of proving that aliens exist versus proving that they don't.

The time interval for SPKI certificate revalidation is shorter than the traditional year granted to X.509 certificates and is based instead on a risk analysis of potential losses from excessively long certificate-validity periods. To avoid clock skew problems, SPKI also allows one-time revalidations that guarantee the certificate's validity for a single transaction.

Another alternative takes an application-specific approach to avoiding revocation. Consider the case in which authority-to-individual communications, such as those for tax filing purposes, must be secured. The obvious solution involves using S/MIME or PGP-secured e-mail. A simpler solution uses an SSL Web server with appropriate access-control measures. The server handles revocation by disabling access for the user, which is

- instantaneous, because there's no CRL propagation delay;
- consistently applied because we don't have to worry about whether the client software will check for revocation or not; and
- effectively administered from the server containing the data, not an external CA.

Other issues such as the "Which directory?" and "Which John Smith?" problems also disappear because everyone knows who the tax department is, and the tax department knows who its users are.

SSL itself provides an example of handling revocation in an application-specific manner. Using the X.509 CRL reason codes as usage cases, SSL handles

- *cessation of operation* by shutting down the server;
- *affiliation changed* and *superseded* by obtaining a new certificate for the changed server URL—changing the subject's name is a standard, if clunky, approach to revoking a capability; and
- *key compromise* by hoping it doesn't occur, since it's unlikely to be useful unless attackers inform the server administrator that they've stolen the key.

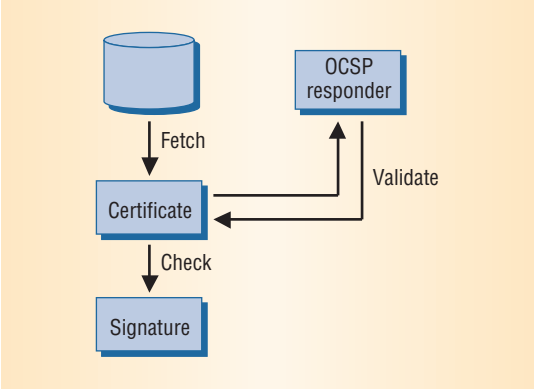


Figure 3. Certificate usage model with the Online Certificate Status Protocol (OCSP) responder, which functions as a special-purpose CRL-creation mechanism and solves many problems inherent in monolithic CRLs.

Issues such as these prompt some analysts to refer to the SSL certificate management process as "certificate manufacturing" rather than PKI because its only real infrastructure component is the one that, once a year, exchanges the client's credit card number for a collection of bits.

Online revocation authorities

The Online Certificate Status Protocol, a recently proposed solution for the revocation checking problem,[7] provides a responder that can be queried online, as Figure 3 shows. In effect, the OCSP responder functions as a special-purpose CRL-creation mechanism. This approach solves many problems inherent in monolithic CRLs by creating a one-off, fresh, single-entry CRL in response to a query. In contrast, as Figure 2 shows, the CRL-based model requires relying parties to repeatedly fetch a huge number of irrelevant entries to obtain status information for the one certificate they care about.

OCSP comes with a cost, however. Instead of preparing a CRL as a background, offline operation, the CA must now perform a certificate lookup and pseudo-CRL-creation operation for each query. To make OCSP economically feasible, the CA must charge for each revocation check. OCSP handles this by signing requests to identify the sender for billing purposes.

Despite their shortcomings, CRLs are useful when a revocation check is, quite literally, worthless. For example, consider a signed executable such as ActiveX controls. A vendor can buy a relatively inexpensive code-signing certificate from a commercial CA and use it to sign software that it distributes over many machines. With an installed base that spans millions of Windows machines, all

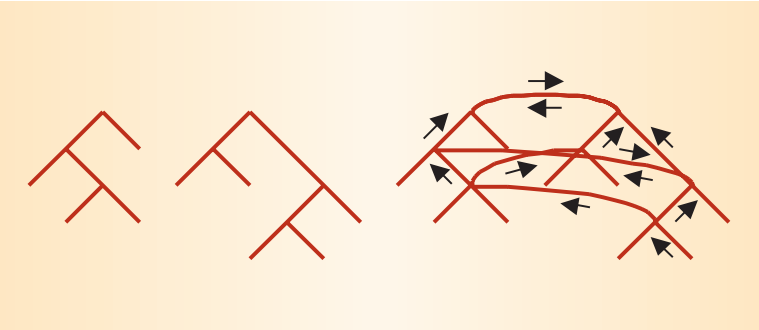


Figure 4. Certificate hierarchy (left), with cross-certification (right), which turns the hierarchy of trust into the spaghetti of doubt.

containing hundreds of ActiveX controls, the costs involved in providing a useful online revocation checking service for code-signing certificates would be astronomical.

As a result, there is a strong financial incentive for CAs to do as little revocation handling as possible for these certificates, beyond paying lip service in the form of an infrequently issued CRL located at a semidocumented location. CRLs are perfect for scenarios such as this. CRLs are also useful when a statutory or contractual obligation to use them exists, so that a relying party must demonstrate CRL use for due-diligence purposes or to avoid liability in case of a dispute.

OCSP problems

OCSP's major shortcoming is that, instead of providing a simple yes-or-no response to a validity query, it uses multiple, nonorthogonal certificate status values because it can't provide a truly definitive answer. This vagueness stems at least in part from the original CRL-based certificate status mechanism. Because a CRL can provide only a negative result, the fact that a certificate is not present in a CRL doesn't mean it was ever issued or is still valid. Some OCSP implementations may report the "I couldn't find a CRL" response as "not revoked/good" or relying parties will interpret the response as "good" because it's not the same as "revoked," which is assumed to be "not good." Opinions on the exact semantics of the various responses vary somewhat among implementers.

A fundamental problem of blacklist-based approaches such as CRLs and OCSP is that they ask entirely the wrong question. Instead of asking, "Is this currently valid?" they ask, "Has this been revoked?" because that's the only question a blacklist can answer. This is not a problem with SPKI because it uses a certificate revalidation process.

A related problem affects CRLs more than OCSP itself. As with their 1970s credit-card-blacklist cousins, CRLs represent an inherently offline operation in an almost completely online world. Credit card vendors realized this when they began using full online verification of transactions in the 1980s. As with online credit-card checks, a response to a query about a certificate only needs to return a simple Boolean value, either "The certificate is valid right now" or "The certificate is not valid right now." Unfortunately, OCSP cannot do this.

CERTIFICATE CHAINS

The initial problem with multiple certificates is constructing a path from a leaf certificate to a trusted top-level root CA and validating the certificate chains once the path has been built. Cross-certification, in which CAs in disjoint hierarchies cross-certify each other, can make this problem almost intractable. When multiple certificate paths lead from a given leaf certificate, all with different semantics, the certificate paths can contain loops, and, in extreme cases, the semantics of a certificate can change across different iterations of the loop. Cross-certification turns the hierarchy of trust into the spaghetti of doubt, with multiple certificate paths possible from leaf to roots, as shown in Figure 4. An alternative, bridge CAs, avoids this problem to some degree by adding a single superroot that bridges two or more root CAs.

To help address the problem of verifying all certificates in a chain, OCSP uses an access concentrator or gateway that farms out the revocation checking to one or more OCSP responders, CRL-based implementations, or both, as Figure 5 shows. The Identrus PKI uses gateways, called transaction coordinators, to provide real-time certificate status information to its members. This makes billing easier, a required feature because each Identrus transaction comes with certain guarantees absent from general CAs.

Working with certificate chains also leads to an extreme case of the "Which directory?" problem. This approach requires locating not just a single directory but multiple directories for the different CA certificates and CRLs. Cross-certificates complicate the problem because the server must now locate all certificates on all possible paths. This can become an intractable problem because it is impossible to determine whether further paths exist based on certificates in as-yet-undiscovered repositories.

One proposed solution uses path construction servers, a type of smart repository that offloads the chain-building process from the end user. While effective in theory, this approach simply offloads all the problems to the PCS, while adding the problem of communicating certificate selection criteria from the client to the server. The path validation server concept extends the approach by offloading the validation process as well as the path construction process, which requires communicating even more constraint information to the server. Both of these approaches, which effectively imple-

ment PKI-crawler analogs to standard Web crawlers, also suffer from the usual "Which directory?" problem.

CLOSING THE CIRCLE

Gordon Bell once observed that a system's most reliable components are those that aren't there. Based on this principle, removing the need to perform revocation checking in the X.509 sense would solve a significant portion of the PKI problem. Working with a PKI community of interest—a restricted group of participants that agrees to play by certain rules—is one approach to achieving this goal. A COI can quantify the risk reliably enough to make meaningful warranties to relying parties, either by requiring that all participants follow certain rules, or by executive fiat or government decree. In contrast, an open environment, in which a certificate represents a general-purpose ID, exposes the issuing CA to virtually unlimited liability—unless it specifically disavows liability for its certificates, as many public CAs indeed do. Disavowing responsibility for identity in ID certificates seems somewhat ironic, however.

Since the entity that accepts the risk can dictate the technology used, these closed communities can use PKI models that differ radically from the traditional X.509 design. The communities will likely be small and tied together by a common interest or policy requirements. The automated clearinghouse network, in which the participants are tied together by both a very stringent set of operating requirements and the ACH system's operating rules, exemplifies such a community. Members agree to comply with the community's rules, and they are then contractually bound to stand behind signatures made with their private keys. These communities manage risk by admitting only members who can afford to carry it and who have the means to manage it.

An unwritten benefit of operating within a closed community is that supporting a fixed, hard-coded set of rules avoids the ongoing feature creep inherent in PKI standards and technology. At some point, the feature set can be frozen, everyone agrees to work within the given framework, and the PKI can be realized.

Bypassing revocation checking

A community can address the revocation problem by collapsing the certificate-fetch-and-validation process even further than OCSP provides for. Both CRLs and OCSP fetch a certificate first, then immediately fetch revocation or validity information for that certificate. An alternative is to combine the two into a single fetch of a known-good certificate from a server known by the community. Although this approach requires a trusted server, OCSP also requires a trusted server to perform the same function, but in a more roundabout manner. This combined fetch technique derives from the original 1970s concept of public-key distribution in which keys were held in public directories or key distribution centers that handed out only known-good keys in response to queries.

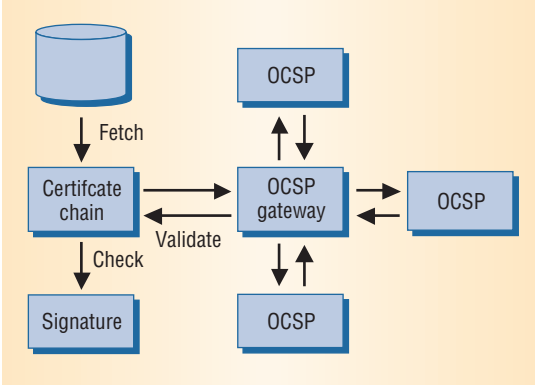


Figure 5. OCSP certificate verification. OCSP offloads the revocation-checking process to a gateway that farms out the checking for a certificate chain.

This approach relies on using established mechanisms such as security policies and auditing that the members of the community have agreed upon or that have been determined sufficient to carry evidentiary weight in court. For example, significant legal precedent is attached to the US's "business records exception," which allows computer-generated records "kept in the course of a regularly conducted business activity" to be treated as evidence rather than as hearsay.[8] It's probably preferable to rely on this type of mechanism than to become the test case for PKI.

Bypassing certificates

In practice, because we are really only interested in the public key, we can go beyond simply collapsing two queries into one, requesting only a copy of the appropriate key needed to perform an operation such as verifying a signature. This returns us to Diffie and Hellman's original Public File approach.[1] Most certificate-based applications already use this technique, which submits a request for a public key to a certificate store on the local machine, for example the Windows registry, and obtains a key in response that, as far as the system knows, is associated with the given entity. Making the key lookup a remote rather than local query simply transfers the administrative burden for key management to a centralized location rather than having end users perform it ad hoc or not at all.

A final step can collapse the query-then-validate process into a single stage. A server trusted to provide a known good key can just as easily validate a signature directly. This is analogous to the online credit-card-processing model in which the relying party can perform the entire transaction online.

PKI Design Recommendations

The various public key infrastructure issues can be condensed into a set of recommendations for working with certificates.

Identity

Choose a locally meaningful identifier such as a user name, e-mail address, account or employee number, or similar value. Attempts to do anything meaningful with distinguished names are probably doomed to failure. "Locally meaningful" doesn't necessarily mean meaningful to humans. For example, an authorization mechanism keyed off an account number is the logical choice for use as a local identifier. On the other hand, objects that are pure tickets—capabilities—don't require any identity information.

Revocation

If possible, design the PKI so that it does not require certificate revocation. The best way to handle revocation is to avoid it entirely. The Secure Electronic Transaction system, Account Authority Digital Signatures (AADS), and the secure shell and secure sockets layer (SSL) protocols exemplify this approach.

If it isn't possible to avoid revocation by designing around it, consider using a PKI mechanism that allows certificate-freshness guarantees, thereby avoiding the need for explicit certificate revocation. A repository that returns only known-good certificates exemplifies this approach.

If you can't avoid revocation, use an online status query mechanism. The best mechanism gives a direct indication of whether a certificate is valid or not, a slightly less useful one provides a certification revocation list (CRL) response. The online certificate status protocol exemplifies this approach.

For cases in which revocation information is of little or no value, use CRLs. Revocation of code-signing certificates exemplifies this approach.

Application-specific PKIs

Certificates and PKIs specifically designed to address a particular problem are much easier to work with than a one-size-(mis)fits-all PKI design. For example, simple public key infrastructure (SPKI) certificates bind a public key to an authorization to perform an action. X.509, on the other hand, binds a key to an often meaningless identity that must then be mapped, via some unspecified means, to an authorization. SPKI is therefore ideal if the goal is to authorize a particular action or grant a capability. Similarly, Pretty Good Privacy handles secure e-mail communication and employs a laissez-faire key management model that imposes few restrictions on users.

In many situations, no PKI is necessary, vendor claims to the contrary. This holds particularly true when two or more parties have an established relationship. For example, the secure shell protocol avoids dependence on a PKI by having the user copy the required public keys to where they're needed, an approach feasible for its application domain. Likewise, AADS takes advantage of existing business relationships to tie public keys to accounts.

In some cases, even PKI-less public-key encryption may be unnecessary. If asking a bank to confirm that a particular certificate is still valid is not faster or easier than asking the bank to directly authorize a transaction, it makes sense to perform the transaction directly. If external constraints specifically require using X.509, nothing requires using it as anything more than a somewhat complex bit-bagging scheme. If you have a means of distributing and managing certificates that isn't covered in a formal standard but that fulfills its intended function, go ahead and use it. This provides the benefits of broad X.509 toolkit and crypto token support from vendors while allowing you to choose a PKI model that works.

An early certificate model proposed by Donald Davies and Wynn Price in the late 1970s provided a precedent for this approach.[9] In this model, arbitrators and key registries—a CA's predecessors—provided a dispute resolution mechanism to relying parties by issuing an interactive certificate attesting to a key's validity in the context of a particular transaction. SPKI's one-time revalidations provide another example of this idea. There are already trends back to this type of model for use with banking and other settlement-oriented transactions.

The Security Assertion Markup Language embodies a related concept. SAML provides an XML-based mechanism for describing authorization mechanisms and authentication events that, however, still rely on an unspecified external PKI. SPKI finally completes the circle by combining the authorization specification system with a built-in, special-purpose PKI. This PKI is designed to avoid the problems of the traditional X.509 PKI while providing a direct authorization management system, rather than stopping short at identification and leaving the mapping from identity to authorization as an exercise for the user.

The proposed approaches for adapting the PKI design to serve the market's current needs I've described, and my advice for implementing them contained in the "PKI Design Recommendations" sidebar, provide a starting point for freeing the technology from the legacy of its X.500/OSI origins. By taking its strong points—broad vendor support in the form of software toolkits and crypto tokens—and adapting it to current standard practices—online processing and validation, Web-based storage and distribution, and real-world IDs such as e-mail addresses or account

numbers—PKI can be turned into a tool capable of meeting current requirements. It remains to be seen whether X.509 can meet this challenge, or whether the way forward will be led by alternatives such as SPKI and XML-based certificates, which already use these techniques. ■

References

1. W. Diffie and M. Hellman, "New Directions in Cryptography," *IEEE Trans. Information Theory*, vol. 22, no. 6, 1976, pp. 644-654.
2. L. Kohnfelder, "Toward a Practical Public-Key Cryptosystem," bachelor's thesis, Dept. Electrical Engineering, MIT, Cambridge, Mass., 1978; http://theses.mit.edu/Dienst/UI/2.0/Composite/0018.mit.theses/1978-29/1.
3. "Information Technology? Open Systems Interconnection? The Directory: Authentication Framework," ISO/IEC 9594-8, 1993, also ITU-T Recommendation X.509, v2.
4. C. Ellison, "SPKI Requirements," RFC 2692, Sept. 1999; http://www.ietf.org/rfc/rfc2692.txt.
5. P. Hope, "Certificate Revocation: Why You Should Do It and Why You Don't," *;login*, Dec. 2001, pp. 36-40; http://www.usenix.org/publications/login/index.html.
6. US General Accounting Office, "Advances and Remaining Challenges to Adoption of Public Key Infrastructure Technology," GAO-01-277, 2001; http://www.cio.gov/fpkisc/documents/gao-01-277pkireport.pdf.
7. M. Myers et al., "Online Certificate Status Protocol—OCSP," RFC 2560, June 1999; http://www.faqs.org/rfcs/rfc2560.html.
8. Federal Rule of Evidence 803(6), "Hearsay Exceptions; Availability of Declarant Immaterial"; http://www.law.umich.edu/thayer/prop803.htm.
9. D. Davies and W. Price, *Security for Computer Networks: An Introduction to Data Security in Teleprocessing and Electronic Funds Transfer*, John Wiley & Sons, New York, 1984.

Peter Gutmann *is a researcher at the University of Auckland. His research interests include security engineering, secure cryptographic hardware, and PKI. Gutmann received a PhD in computer science from the University of Auckland. Contact him at pgut001@cs.auckland.ac.nz.*