Delegated Cryptography, Online Trusted Third Parties, and PKI

Trevor Perrin, Logan Bruns, Jahan Moreh, Terry Olkin
{tperrin,logan,jmoreh,tolkin}@sigaba.com, Sigaba Corporation

Abstract

We propose that enterprise PKI users should delegate asymmetric cryptography operations to an online trusted third party maintained by their enterprise, thus freeing themselves from the burdens of owning key pairs or interfacing with PKI. Users would authenticate to this third party (which we'll call a delegate server) and then request it to sign and decrypt data on their behalf with its own private key and encrypt and verify data with the public keys of other users or other delegate servers. A delegate server would thus be like a CA in that it represents a group of users but like an end-entity in that it signs and decrypts using its own private key and encrypts and verifies using public keys which it has calculated certificate paths to. To bind encryptions and signatures performed with delegate server keys to particular users we suggest two approaches, one using XML security standards, and one using what we call signature operation certificates which are signed by a delegate server and bind a hash value to a signing user, and encryption operation certificates which are encrypted to a delegate server and bind a symmetric key to an intended decrypting user. These operation certificates have several benefits, and so we propose that conventional PKI end-entities as well as delegate servers could use them to encapsulate signatures and encryptions, and that current PKI protocols could be modified to support them. If this was done, enterprises could individually choose whether to utilize delegate servers or conventional PKI. In many situations a delegate server infrastructure would be easier to deploy, easier to use, and easier to integrate with applications, and would offer advantages in security, extensibility, and efficiency.

1 Introduction

The deployment of cryptography on modern computer networks is proceeding on two fronts. The first is the use of cryptography to achieve authentication. Users log-on to workstations, networks, and networked applications using credentials such as passwords, one-time password devices[1,2], smartcards containing private keys, or biometrics[3]. Cryptography either provides encryption to protect the transmission of the user's credentials (e.g. SSL[4] for website passwords), or provides credentials-presentment protocols with various intrinsic degrees of security (e.g. SRP[5,6] for passwords, or SSL with client authentication for private keys).

These authentication and session-establishment protocols are easy to use for both the credentials-presenting and the credentials-verifying parties, and thus are widely deployed. Such methods are not, however, sufficient to fulfill the promise of cryptography. Ideally a cryptographic infrastructure would be capable of providing confidentiality and authentication to both interactive and noninteractive communications amongst large groups of users[1]. Compared to this, authentication methods (with the exception of asymmetric key pairs) are limited in both scale and scope: scale in that a single credential should only be used between a single pair of users, and scope in that these methods can only secure interactive traffic, and thus cannot be used to encrypt or sign data such as emails or files. Attempts to solve the scalability problem involve clients authenticating once to a "single sign-on" service with their primary credentials and then receiving secondary credentials that they can use to access other services. This approach represents the current cutting-edge of authentication techniques, particularly on the web where it is being pursued by Microsoft Passport[7], the Liberty Alliance Project[8], and the SAML XML specification[9].

The second front on which cryptography is being deployed is known as public key infrastructure (PKI). This technology assumes that every client should possess a long-lived asymmetric key pair[10]. The public key can be shared with different parties, which can then authenticate the private key owner, verify the owner's signature on a piece of data, or encrypt data that only the owner can decrypt. A single key pair thus allows point-to-many instead of just point-to-point security, and can be used to secure both interactive and noninteractive communications. These characteristics enable certificates[11]: a user Alice could sign Bob's public key along with Bob's name, thus producing a certificate which could be published in a public direc-

tory or carried around by Bob, and which would convince anyone who trusts Alice that Bob's public key really belongs to him. Bob can issue certificates to other users as well; the certificates issued amongst a group of users comprise a directed graph, and if one user can compute a path from himself to another then he can determine that other user's public key and use it to secure communications between them. Often a specialized entity known as a certificate authority (CA) will assume responsibility for issuing, revoking, and publishing the certificates for some group of users. A system of cooperating CAs, directories, and other supporting services are what we collectively refer to as PKI.

PKI appears to meet our requirements for cryptographic infrastructure, yet attempts to deploy it over the last decade have met with strikingly little success. Even within a single enterprise, PKI rollouts are often expensive and time-consuming, and result in stovepipe systems unable to interoperate with each other, integrate with new applications, or evolve to meet new demands and incorporate new technologies[12,13,14,15]. Some feel these are growing pains that will disappear as the technology matures, but we will argue that they instead reflect several systemic flaws in the PKI vision:[2]

First, private keys are difficult for people to use and easy for attackers to abuse. Private keys are not memorable like passwords, derivable from the person like biometrics, or enterable from any keyboard like one-time passwords. They are instead typically stored in a file on the user's computer, stored on smartcards, or stored at a server that delivers them to the user on request[16]. These approaches have various deficiencies in terms of portability, universality, and security. Security concerns are aggravated because private keys can be stolen and abused offline (i.e. without generating an audit trail).

Second, trust relationships are difficult for people to manage. If a global PKI had materialized, with a single CA at the root of a certificate hierarchy, then this would be a non-issue: each user would simply configure his software to fully trust the CA's public key. Instead many different CAs exist, some public and some private, and in some systems (such as PGP[17,18]) users can issue certificates to each other as well. Faced with such a fragmented trust environment, users must configure their software's trust list with the "trust anchors" from which to begin computing certificate paths, by first importing and verifying these anchors' public keys and then indicating to what extent the user trusts each anchor or over which names the user considers the anchor authoritative. These procedures are complicated yet security-critical, and users rarely understand or perform them well[19].

Third, the interface between end-user software and PKI systems is complex and difficult to standardize. End-user software must interact with the PKI to perform management operations such as obtaining, revoking, renewing, archiving, and recovering the user's certificate and key pair, and also to construct and validate certificate paths to other users' public keys. These operations require knowledge of the PKI's management protocols, directory architecture, trust topology, certificate formats and profiles, certification policies, and revocation/validity-checking methods, among other things. Each of these provides an axis along which PKIs can and do vary. As a result, PKI end-user software tends either to provide lowest-common-denominator support for PKI or to be tightly coupled to a particular vendor’s products or even a particular deployment, yielding systems that are either underfunctional or overly rigid and brittle.

Fourth, end-to-end path construction is inefficient: every user’s certificates and revocation data must be made accessible to every other user in a timely fashion, and every user must compute the paths between himself and every other user with whom he wants to communicate. The first requirement necessitates a high-performance and high-availability distributed directory that will be difficult to scale to large communities of geographically dispersed users. The second requirement results in redundant computations of potentially lengthy and complex paths.

A few observations about the enterprise environment will suggest a way to remedy these flaws. PKIs are generally deployed to support users who communicate under the aegis of enterprises (meaning businesses or similar organizations). In these enterprises there are authentication methods already deployed to control access to networks and workstations; there is trust between members and their enterprise, and between enterprises and each other; there are administrators capable of configuring and maintaining networked services for enterprise members; and there are private networks offering reasonably high performance and reliability, and some measure of protection against outsider attacks.

In such environments, we believe the authentication and PKI uses of cryptography should be hybridized in the form of a networked service, hosted by an enterprise for its members, which users could authenticate to and request to perform asymmetric cryptography on their behalf. More precisely: users within an enterprise would authenticate to a locally-provided delegate server (DS) which would possess a key pair and would interface with a PKI system whose end-entities would include both individuals and other DSs representing other enterprises. To produce a signature on a document Alice would send her DS a cryptographic hash of the document, and the server would sign this hash along with an attached statement that says "this was presented

by Alice", and return this signed message to Alice who would embed it in the document. To encrypt a document to Bob, Alice would send her DS a symmetric document encryption key and the name Bob, and the DS would encrypt the symmetric key along with an attached statement "this is intended for Bob" using Bob's public key or the public key corresponding to Bob's DS. When Bob received these secured documents from Alice he would extract the messages that came from Alice's DS and either process them using his own private key or forward them to his DS. If the latter, his DS would verify or decrypt these messages and then examine the attached statements, and either confirm that it was Alice on whose behalf the signature was produced, or release the document encryption key after verifying that it was indeed intended for Bob.

This approach addresses the first flaw in PKI by using convenient authentication methods, instead of private keys, to access a server which can monitor all events for intrusion detection and response purposes. It addresses the second and third flaws by centralizing trust relationships and PKI software at the organizational instead of individual level, where they can be managed at a single point by qualified staff. It addresses the fourth flaw by associating certificates and key pairs with enterprises instead of individual users, thus reducing the volume of certificates and revocation data that must be distributed and the length of paths that must be computed. It also centralizes path computation at servers where its cost can be amortized across large groups of users.

One drawback of this approach is its reliance on communication between users and DSs. This raises issues of performance and availability, and also raises the spectre of denial-of-service and traffic analysis attacks[20], but the characteristics of enterprise networks we mentioned above should mitigate these concerns. Another drawback is the potential security risk and performance bottleneck of performing all asymmetric operations for an enterprise at a single point. We believe that adequate security can be achieved by choosing an appropriate lifetime and strength for DS key pairs and by confining sensitive data at the DS to a secure coprocessor[21,22], and that adequate performance can be achieved with appropriate hardware or techniques such as caching Diffie-Hellman key agreement values. A third drawback is that having DSs involved in all cryptographic operations may raise privacy concerns, but it provides compensating benefits such as centralized auditing, fine-grained access control, and instantaneous user revocation. A final drawback is that current cryptographic protocols and data formats were not designed with DSs in mind, but we believe that an elegant extension to the notion of certificates will make it easy to retrofit DS support into current PKI systems.

In what follows we will expand on these points to argue that DSs make large-scale cryptographic infrastructure feasible. In the next section we will take a step back and develop a more abstract understanding of the problems and methods of cryptographic infrastructure. In section three we will apply this understanding to the real world to show why delegated cryptography makes sense. And in the final section we will consider how current cryptographic protocols and data formats could be retrofitted to support this technique.

2 Cryptographic Infrastructure

For our purposes, the basic situation of cryptographic infrastructure is this: there is a graph consisting of nodes linked by communication channels, where a node could be a machine or a person, and a channel could be such things as a trusted courier, a computer network, or even just a stretch of time. Some of these channels are physically secure. Others may be subject to passive attacks, where an adversary eavesdrops on the messages going back and forth; or active attacks, where an adversary alters, deletes, and adds messages. It is desired that certain communications between nodes be confidential and/or authenticated. By confidential we mean that an adversary cannot determine their contents. By authenticated we mean that an adversary cannot delude one party to a communication as to the other party's identity.

Physically secure channels are not subject to attacks and are thus both confidential and authenticated. Otherwise passive attacks can violate confidentiality and active attacks can violate authentication. To ensure these properties on channels subject to these attacks the nodes must code their communications using cryptographic algorithms. These algorithms can be used to protect either noninteractive messages (i.e. self-contained units of data sent from one node to another) or interactive sessions. Interactive sessions allow the use of cryptographic protocols which make certain security properties easier to obtain; in particular, Diffie-Hellman key exchange[10] can establish confidential sessions between any pair of nodes, thus making authentication the only difficulty in the interactive case.

2.1 Cryptographic Data

With the exception of Diffie-Hellman key exchange, the algorithms and protocols used by two nodes to provide confidentiality and authentication require that certain related cryptographic data be used as inputs by both sides. We can classify these data as credentials, symmetric keys, or asymmetric keys. Credentials involve a data source possessed by one node (such as a password, one-time password device, eyeball, etc.) and credentials-verifying data possessed by another node

(such as the password itself, a hash of the password, an SRP verifier[5,6] of the password, an iris code[23], etc.). Credentials either possess low entropy (passwords), imprecision (biometrics), or time-variance (one-time password devices), and thus can only be used to authenticate interactive sessions (i.e. they can't provide message security). Certain credentials, such as passwords, can be used in conjunction with zero-knowledge password protocols[5,6,24] that provide mutual authentication between nodes. Otherwise, the credentials need to be presented from one node to the other, which can only be done securely if the credentials-presenting node has already authenticated the credentials-verifier.

Symmetric keys possess high entropy, and thus a pair of nodes sharing a symmetric key can exchange confidential and/or authenticated messages and establish confidential and/or authenticated sessions. Asymmetric key pairs consist of both a private and public key. The private key should be kept secret by its owner, but the public key could be shared with many other nodes, which makes this type of cryptographic data intrinsically different from both credentials and symmetric keys, which can only provide security between a single pair of nodes (if these data are shared with more than two nodes, the excluded parties to any communication relying on these data could launch passive or active attacks on the communication). Source authentication can be provided to messages travelling from the private key owner to a public key possessor, and confidentiality can be provided to messages travelling in the opposite direction. The private key owner can also authenticate himself interactively to a node possessing the public key.

Pulling this together, nodes can generate cryptographic data and then exchange them with other nodes over secure channels, then leverage these data to add security to vulnerable channels. Credentials or symmetric keys must be exchanged over confidential and authenticated channels; otherwise the adversary could intercept or forge the data, and later on launch attacks. Public keys can be exchanged over authenticated but nonconfidential channels, since secure use of asymmetric cryptography does not depend on the secrecy of public keys. The value of all this is that a secure channel which may be too transient, performance-limited, or costly to use for regular communications can be used to bootstrap security on a more convenient but vulnerable channel.

2.2 Trust

Now it may happen that nodes desire a secure channel, but do not have any physically secure channel with which to bootstrap a cryptographically secure one. In this case, they may have to trust a third party. For example, if Alice wants to send a secure message to Charlie but can't do so directly, she may have to entrust this message to Bob. But just how far do Alice and Charlie trust Bob? If Alice sends the message directly through Bob, and Charlie is willing to believe Bob when Bob says "Alice sent this", then Bob is clearly in a position to read, alter, and forge messages from Alice to Charlie. Alternatively, Alice could give Bob a symmetric key to give to Charlie, then communicate with Charlie herself on a separate channel, but since Bob knows the symmetric key, he could still launch passive or active attacks on this channel, so we won't consider this method as requiring significantly less trust in him. A third alternative is for Alice and Charlie to exchange public keys through Bob. Bob could perform a man-in-the-middle attack here, giving Alice and Charlie false public keys so that he could read and forge their messages, but if he doesn't want to be found out when they try to communicate he will have to launch an active attack where he makes it appear as if the messages were actually processed using the appropriate public keys. If Bob's diligence in these attacks flags, or if Alice and Charlie acquire a communication channel that Bob can't attack, then it is likely that they will discover his perfidy.

So we can roughly say that Bob is either completely trusted (meaning Alice and Charlie accept that he can read or alter their communications without detection), or partially trusted (meaning Alice and Charlie accept that he can only read or alter their communications if he falsifies the key exchange, and that he can only keep this from being detected by launching continuous active attacks on their communications). To clarify the overall picture, we can imagine that each node maintains a table of trust relationships, each of which consists of some cryptographic data and a list of names[3] with associated trust values. For example, Alice might have a trust relationship indicating that Bob's symmetric key is completely trusted for Bob but only partially trusted for Charlie. The trust table contains both primary trust relationships, which the node axiomatically trusts, and derived relationships, which are a consequence of communications that were secured under some other relationship.

For example, when Bob sends Charlie's public key to Alice, Alice will add it to her trust table as a new trust relationship, but if Alice laters discovers that Bob's key was compromised, or decides that she no longer trusts Bob to vouch for Charlie, then the relationship containing Charlie's public key and any relationships derived from it will be invalidated. As another example, asymmetric cryptography is expensive. When Alice establishes a secure session with Charlie, she will likely not encrypt all her messages to him using his public key but might instead verify his signature on a Diffie-Hellman public value as a prelude to establish-

ing a session key, which she will consider as a trust relationship with Charlie for the duration of the session.

2.3 Trust Derivation

The point of the above is that a cryptographic infrastructure can be analyzed in terms of both its static structure of primary trust relationships and the procedures it uses to build and utilize a dynamic structure of derived trust relationships. To make this more concrete we'll examine PKI:

A classic, X.509-style PKI[25,26,27] consists of end-entity nodes and CA nodes. End-entities and CAs have key pairs, and end-entities have trust relationships with certain CAs wherein they partially trust the CAs to vouch for certain other end-entities. CAs express their trust relationships with each other and with end-entities in the form of certificates, which they make available in a directory (which is equivalent to broadcasting them to all end-entities across an unsecure channel). A certificate is an authenticated message from a CA which asserts a trust relationship. This trust relationship contains a public key and expresses complete trust in an end-entity name if it is an end-entity certificate, or expresses partial trust in a set of end-entity names if it is a CA certificate. When an end-entity examines a certificate which comes from a CA whom the end-entity trusts, and references some names which the end-entity trusts the CA to vouch for, the end-entity will add a new, derived trust relationship to its table consisting of the certificate public key and those names. When an end-entity wants to communicate with another, it will search in its trust table for a completely trusted relationship with the other end-entity and use the corresponding public key.

Now since CAs don't participate directly in end-entity communications, all derivation of trust relationships is performed by end-entities themselves. Furthermore, since complete trust is only granted to end-entity public keys whose corresponding private key is assumed to be held by the end-entity himself, Alice has only one choice when she wants to send a confidential message to Bob: she must derive trust in his public key and encrypt the message to it. One theme of this paper is that there are alternative infrastructure choices worth exploring. In particular, imagine replacing the CAs on the trust path between Alice and Bob with online nodes and changing the trust relationships along this path to involve complete rather than partial trust. Alice could still derive a trust relationship with Bob and encrypt to him if these nodes published certificates, but she could just as well encrypt and send the message to the next node along the trust path, with an attached statement that says "please forward this to Bob". If each node did the same the message would get to Bob securely without any derivation of trust relationships at all.

More abstractly: given any network of trust relationships, nodes could always securely communicate with other nodes by relying on the involvement of intermediaries. Derived trust relationships are a way for endnodes to improve this process by relying on intermediaries to securely communicate a trust relationship once which the endnodes can thereafter use directly. This enables partial instead of complete trust in intermediaries when the derived relationship involves public keys, and more generally improves the performance, security, and availability characteristics of an infrastructure since the overhead of constantly contacting the intermediaries is removed, and they are also (at least to some extent) removed as potential points of failure or attack.

From this perspective, the PKI decision to minimize interaction with intermediate nodes seems well-founded. But there are costs associated with end-to-end derivation of trust relationships as well: for one, if multiple end-to-end paths share an intermediate path segment, having each endnode calculate a trust relationship across that segment is redundant. For another, end-to-end derivation requires each endnode to be capable of communicating with every other intermediary in the infrastructure; this level of compatibility might be difficult to achieve in a heterogenous infrastructure, and the amount of communication required could be expensive in a large or geographically dispersed system. Finally, an intermediary node might desire to control an endnode's ability to communicate securely, so as to be able to monitor the endnode's cryptographic activities, and instantly revoke the endnode's access if necessary. This is not possible if trust paths can be derived between the endnode and other nodes that exclude the intermediary. Clearly, weighing all these factors to select and locate trust derivation mechanisms is a challenging job, and much of our argument centers on the claim that conventional PKI has done this poorly.

2.4 Design Methodology

At this point we can suggest a methodology for designing cryptographic infrastructures. First, nodes and channels should be identified, and channels should be classified as either secure enough to use for establishing primary trust relationships or as vulnerable and in need of cryptographic protection. Nodes should then be assigned cryptographic data sufficient to their capabilities: asymmetric key pairs if they are capable of managing these and shouldering the computational burden, credentials or symmetric keys otherwise. We should assume nodes will distribute these data to each other and assign trust in them, thus establishing their primary trust relationships. Finally, mechanisms should be deployed to support the derivation of trust relationships. These should be placed so as to maximize the

gains acquired by removing the involvement of intermediaries while trying to avoid requiring derivations be performed from nodes where this would be difficult or expensive, or where it would remove nodes that we would prefer to remain involved. We will apply this methodology in the next section to analyze the application of PKI to enterprise scenarios and to develop our suggestion for a more effective approach.

3 Delegate Servers

We are using the term 'enterprise' for any organization that has the following characteristics: the organization has a number of members and an administrative entity (or perhaps multiple entities, arranged in a hierarchy); the members would like to have secure communications with each other and with people outside the organization across vulnerable computer networks; the members trust the administrative entity to vouch for the identity of everyone with whom they communicate, and they also trust that it will not read, modify, or forge their communications illicitly; each member has some sort of secure channel with the administrative entity, even if this is only the ability to walk into the administrator's office and talk to him; and the administrative entity has the resources to operate a networked service for the benefit of its members.

In the previous section we presented a methodology for designing cryptographic infrastructures. In the first section we presented a number of criticisms of PKI, and presented an alternative approach using what we called delegate servers. We will now apply our methodology to the enterprise scenario, showing step-by-step the construction of a DS-based infrastructure, and highlighting how it differs from conventional PKI.

3.1 Credentials vs. Private Keys

First, we must identify nodes and channels. Clearly each member of an enterprise is a node, and we will make each enterprise's administrative entity a node as well (or perhaps a hierarchy of nodes, if this reflects administrative relationships more adequately). Computer networks will provide channels between all nodes which are subject to both active and passive attacks, but we will assume that there are low-performance but physically secure channels between enterprise members and administrative nodes, and that there are low-performance channels protected against at least active attacks between certain administrative nodes, whether belonging to the same or different enterprises.

The next step is to assign cryptographic data to nodes. The PKI approach is to give each administrative node a key pair and call it a CA, and give each person a key pair and call him or her an end-entity. Since key pairs are the most powerful form of cryptographic data, and since administrative entities can presumably install their private key once on a high-performance system and be done with it, we concur in giving key pairs to administrative nodes. We will call those administrative nodes that have direct trust relationships with enterprise members DSs, and those which only have trust relationships with other administrative nodes (in a bit of foreshadowing) CAs. As for enterprise members, we reject PKI's assertion that key pairs are the best form of cryptographic data for them. Instead, we feel that authentication credentials such as reusable passwords, one-time passwords, and biometrics are generally easier for the enterprise to deploy and easier for people to use. Moreover, authentication credentials are widely deployed: password authentication systems such as Kerberos[28,29], RADIUS[30], LDAP[31,32], or various other Unix and Windows logon systems exist on most corporate networks, and a vendor recently shipped more than ten million of their one-time password devices[33]. The DS infrastructure will thus assume only that people have some way of authenticating themselves to a service; we will let each enterprise, or perhaps even each person, decide precisely which authentication method they prefer. Since authentication credentials can only provide point-to-point secure sessions instead of point-to-many secure sessions or messages like asymmetric key pairs can, this decision will have substantial ramifications. Before moving on, let's examine it carefully.

To compare credentials against private keys we need to consider both ease of use and security characteristics, since these tend to trade off against each other. For example, a general difference between authentication credentials and private keys is that the former can only be used online and the latter can be used offline as well. Offline support is an occasional convenience for the user, particularly when travelling, but it's an even greater convenience for attackers, since they can steal a private key and use it without generating an audit trail or monitorable events. Since there are ways the DS approach could allow a limited degree of offline operation, we consider the offline exploitability of private keys a serious deficiency. To compare credentials and private keys more closely we will examine three dimensions: portability, universality, and vulnerability. By portability we mean the ease with which a person can carry the data with him. By universality we mean whether or not the data can be used with any computer or whether the computer requires special hardware. By vulnerability we mean how easily the data can be stolen, taking into particular account active attacks on authentication protocols and local attacks by software or hardware on the user's machine. We will consider three authentication methods versus three private key storage methods: memorized passwords, one-time password devices, and biometrics, versus private

keys stored in files, smart cards, and servers.

Memorized passwords are highly portable, universal since their entry only requires a keyboard, relatively invulnerable to active attacks since they can be used with zero-knowledge password protocols[5,6,24] but might be vulnerable to online guessing, and vulnerable to local attacks since the password must be entered and stored in memory somewhere.

One-time password devices[1,2] are reasonably portable, universal since entry of the password only requires a keyboard, invulnerable to active attacks since they can be used with zero-knowledge password protocols and have enough entropy to resist online guessing, and invulnerable to local attacks since the device secret is not exposed to the local computer.

Biometrics[3] are extremely portable, not universal since their entry requires special hardware, vulnerable to active attacks since they cannot be used with zero-knowledge password protocols, and vulnerable to local attacks since the biometric value is typically exposed to the local computer; also, attacks are unusually damaging since once compromised a biometric cannot be changed.

Private keys stored in files are difficult to transport, universal since the file could be copied to any computer, and highly vulnerable to local attack since the private key file can be stolen at any time, not just when the user is performing an operation.

Private keys stored in smart cards are reasonably portable, not universal since not all computers have smart card readers, and reasonably invulnerable to local attack since the private key is contained within the card and probably only vulnerable to hardware attacks such as timing or power analysis[34,35], or glitching[36]).

Private keys stored on servers and delivered to authenticated users are as portable and universal as the underlying authentication technique, and are vulnerable to local attacks.

In sum: storing private keys on the file system is inferior in our metrics to downloading private keys from a server, assuming a universal authentication technique is used. So with private keys, we essentially must choose between a nonuniversal solution reasonably secure against local attack (smart cards) and a universal solution vulnerable to it (server delivery). With authentication techniques, we can choose a solution that is both universal and secure against local attack (one-time password devices). Since there is no single metric or combination of our metrics in which private keys are superior to authentication credentials, since private keys can be exploited offline, and since authentication credentials are already widely deployed, we believe that enterprise cryptographic infrastructures should be designed for people with authentication credentials (including private keys) instead of solely for people who have private keys.

3.2 Delegated vs. End-to-End Derivation

Now that we've assigned cryptographic data we can assume nodes distribute these data to each other along secure channels. In the PKI case that means all nodes share their public keys with those whom they want to be trusted by and then construct primary trust relationships: CAs will trust end-entity keys completely for the end-entity's name, and all nodes will trust CA keys partially for the set of end-entity names over which they consider each CA authoritative. In the DS case the only differences are that people share their credentials-verifying data with their DS instead of sharing their public keys; that if the authentication protocol for a given credential provides mutual authentication, then the person constructs their trust relationship with the DS in terms of that credential instead of the DS's public key; and that all trust extended to DSs is complete instead of partial.

Now that we have a structure of primary trust relationships, we must specify the mechanisms whereby this structure is used to secure communications. In a PKI system CAs publish their primary trust relationships as certificates, then do nothing further. End-entities retrieve these certificates and use them to derive trust relationships with other end-entities' public keys which are then used as inputs to cryptographic algorithms and protocols. We earlier criticized this approach because it exposes end-entity software to the specifics of all the PKI systems and data formats in a potentially large and heterogenous infrastructure, and because it requires the distribution of significant amounts of certificates and revocation data, and redundant path computations upon these; below we elaborate on these points:

To address the first point: a network of primary trust relationships is similar to a routing network, in that each link (i.e. cryptographic datum) is labelled with the addresses (i.e. names) which can be reached through it. The requirement that end-user software must construct a complete trust path is analogous to a requirement that each host network card not only compute the entire route for its packets, but understand each link-layer protocol along that route. This violates modularity by making every point on the edge of a network dependent upon every link in the network. The result is either going to be that new PKIs are prevented from joining the system and changes are not made in an effort to keep the system homogenous so that end-user software will continue to function, or alternatively every modification will force painful software upgrades on the user population; we earlier called this situation one of rigidity and brittleness. We will enumerate the dependencies that cause this brittleness below:

To construct a certificate path requires retrieving certificates from one or various directories, com-

prehending the syntax and semantics of each certificate, applying a search algorithm appropriate to the PKI's trust topology to these certificates to determine candidate or partial paths, applying a revocation or validity-checking method to each certificate along such a path to determine its validity, and iterating the above processes until a complete and valid path has been built. This process is thus dependent upon knowledge of directories and directory protocols (such as X.500[37], LDAPv2[31], LDAPv3[32], or various proprietary protocols); certificate formats (such as X.509[25], OpenPGP[18], or SPKI[38]) and their semantics (how they express identities, attributes, and authorizations) and their many different versions, profiles, policies, extensions, unique identifiers, signature algorithms, encoding quirks, etc.[39]; trust topologies (such as bidirectional hierarchies, top-down hierarchies, hybrids such as bridge structures, meshes, etc.); and revocation/validity-checking methods (such as CRLs[25,26], segmented CRLs[25,26], delta-CRLs[40], sliding-window delta CRLs[41], OCSP[42], DPV/DPD[43], etc.).

And these are only the complications that pertain to path construction; if we consider the management protocols between an end-entity and CA we find a similar raft of competing, complex, and variably-implemented protocols[44] such as PKCS #10[45] with SSL[4], PKCS #10 with PKCS #7[46], CMP[47], CMC[48], and SCEP[49]. In some of the above cases we can expect convergence and stability as winners are chosen and interoperability is pursued, but in other cases, particularly those involving certificate semantics and revocation strategies, there are substantial unresolved theoretical issues. In an environment of such turmoil and complexity we can't expect end-entity software that supports anything but the simplest or most homogenous PKI deployments to exist for some time.

Even disregarding this complaint, end-to-end path construction suffers from being inefficient. Many users, particularly within an enterprise, will share the same trust anchor points and validation policies, and thus the certificate paths they construct will be identical. Path construction is expensive because it requires retrieving certificates and revocation information from a potentially wide variety of sources and performing expensive signature verifications and revocation look-ups (or online status checks) upon these. Performing revocation checking on end-user certificates is particularly taxing[50]: end-users generally have transient relationships with their certificate issuer (people get hired, fired, demoted, etc.), and their private keys are highly vulnerable to compromise or loss (people lose their smart cards, forget their PINs, store their private keys on unprotected computers, etc.). A large volume of revocation information will thus be generated, and since the costs of relying on a compromised private key are high, end-entity software must frequently download the latest revocation lists. In the case of a complex trust topology, these costs are magnified: path construction is not deterministic but requires sophisticated graph exploration algorithms that can ignore dead-ends and detect cycles; even if these algorithms work flawlessly they may have to retrieve and consider a large number of certificates before determining a path.

So for reasons of overcoupling and inefficiency, we feel that PKI's reliance on end-user software which processes messages from each CA to derive end-to-end trust relationships is a poor design choice. A better approach would be to have administrative nodes derive trust relationships. For one thing, an administrative node could derive these relationships once and then reuse them. For another thing, administrative nodes are assumed to be under the control of a competent staff who can configure them and upgrade their software quickly, thus managing the complex coupling with the infrastructure that trust derivation requires. Also, there will be many fewer administrative nodes in an infrastructure than end-user nodes, so having to configure and maintain them is less of a burden. The administrative nodes that end-users have direct access to we have called DSs. Since end-users must have access to these trust-derivations to make use of them, it follows that DSs are where trust derivations should be performed, or at least made accessible.

Now it should be noted that end-to-end derivation of trust relationships without online interaction with intermediaries is impossible in a DS infrastructure, because of the structure of primary trust relationships: end-users do not have public keys which can be communicated widely, instead they have authentication methods which can only be used to provide secure sessions with their DS. Thus, trust derivation will not only begin at DSs, it must terminate at them as well, since it is impossible to construct a path all the way to an enterprise member.

So we have located trust derivation; now the question is how to perform it. Obviously we should use PKI! Our criticisms of PKI are that it requires private keys, that it requires a tight coupling between end-entities and infrastructure, and that it is inefficient when deployed to a large community of users with a high revocation rate. But these criticisms don't apply to DSs: for one, DSs already have key pairs, and they are maintained by a staff which can manage the challenges involved in software that is tightly coupled to an infrastructure. Also, there are many fewer DSs then there are end-users, and DSs are much less likely to be compromised than end-users, or to have a volatile relationship with the enterprise, so revocation rates will be lower. Path construction will thus occur in a much smaller and more stable environment, and in this environment the advantages of PKI come to the fore:

namely, communication with and trust in intermediaries is minimized. Given that cross-enterprise trust relationships will span the public Internet, where traffic analyzers and denial of service attackers may be lurking and where occasional traffic outages and performance lags occur; and given that cross-enterprise trust paths might involve any number of intermediaries (governments, industry consortiums, public CA maintainers, etc.), there is clearly value in reducing communication with these nodes and trust exposure to them.

Within the enterprise, it's a completely different story. Private keys are burdensome for people, desktop software is so widely deployed that complex configurations and upgrades are difficult, enterprise networks are reliable, high speed, and sheltered from grosser forms of abuse, and though DS involvement adds a new point of attack and failure to the infrastructure, it also adds a point of monitoring for the first point of attack (the user's credentials). Under the reasonable assumptions that DSs are highly trusted, that credentials are highly vulnerable, and that monitoring substantially increases the ability to detect compromises, determine their extent, and trace their source, this results in a net increase in security.

3.3 Delegate Server Infrastructure

Pulling this all together: we are proposing that DSs should possess key pairs and function as end-entities in a PKI, and that enterprise members should authenticate to DSs to secure their communications. An efficient way to do this is for enterprise members to perform bulk cryptographic operations involving symmetric keys and hash values, then call out to DSs only to perform asymmetric cryptography upon these. We will discuss protocol details in the next section. For now, we should view the result as simply PKI applied at the granularity of enterprises instead of individuals: since functioning as a PKI end-entity is difficult for both the user and his software, and since PKIs with large numbers of volatile and vulnerable end-entities have performance problems, we are applying PKI only at a high level, and extending security down to the user level with simpler and more user-friendly authentication mechanisms. Alternatively, we can view DSs as simply an aggregation technique: a DS allows us to treat a group of user nodes as a higher-level enterprise node from the perspective of a higher-level cryptographic infrastructure. Though we've claimed this higher-level infrastructure should be PKI, this is by no

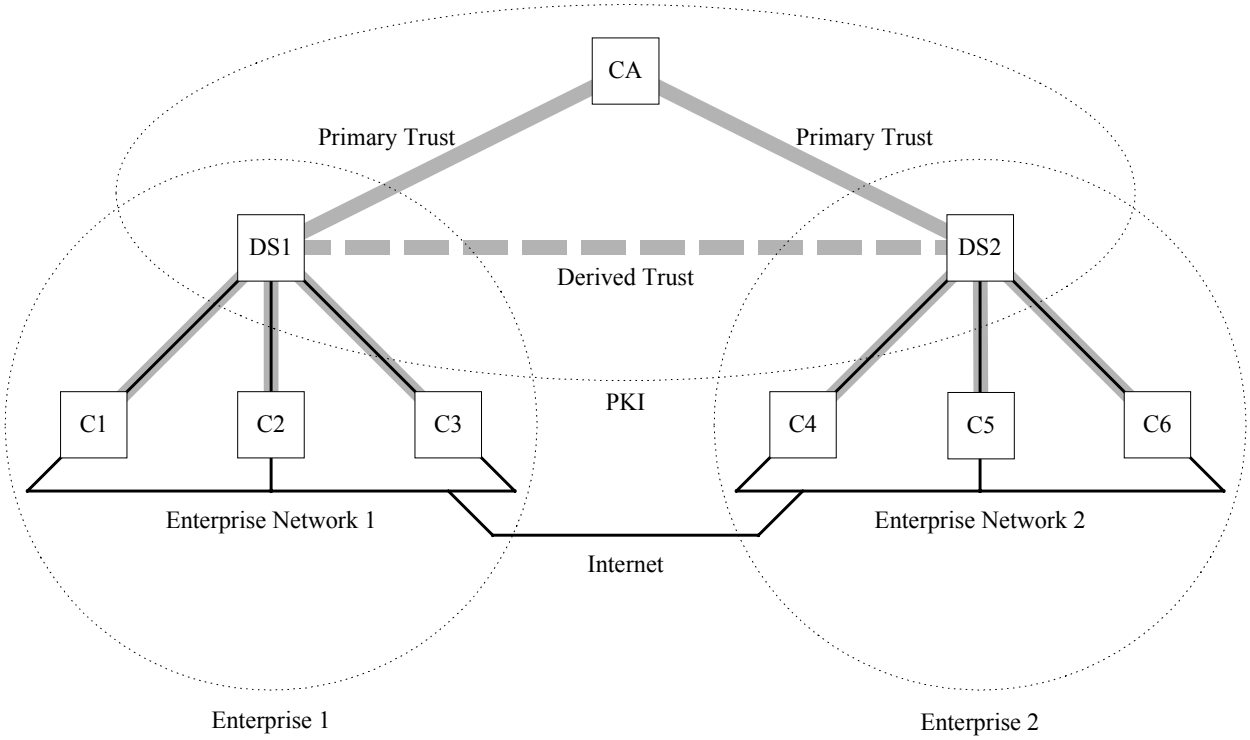

Diagram. An example infrastructure showing two enterprises, each with a single DS and three clients. The thick gray lines represent trust relationships: primary trust relationships link DSs and clients, and DSs and CAs. The DSs construct paths to each other to establish a derived trust relationship, shown as a broken gray line. The thin black lines represent network communication channels. Clients communicate with their DS and with other clients in the same enterprise using the enterprise network. Clients communicate across enterprise boundaries using the Internet.

means a foregone conclusion, and by no means the only way we could knit DSs together. If end-user software treats DS messages as opaque blobs of data, then we can retain some flexibility for DSs to support whichever algorithms, formats, and alternative infrastructure strategies they desire. For example, when asked to encrypt a symmetric key to another DS, a DS could instead add a unique identifier to the key and forward it over a secure channel to the other DS, then return this identifier to the client as if it was the encrypted key.

From a user's point of view, a DS infrastructure would be simple yet effective. Alice could sit down at any computer that was inside her enterprise's network (or perhaps even outside it), configure the computer with the address of her DS, and then, with nothing more than her password, achieve confidential and mutually authenticated communications using email, file transfers, instant messaging, remote login, web browsing, videoconferencing, etc.. From an administrator's perspective a DS would be reasonably easy to manage. The chief task would be to set up primary trust relationships with users, CAs, and other DSs, and to configure the DS with whatever knowledge of PKI systems was necessary to compute paths to other DSs and end-entities (perhaps knowledge of remote directories, certification policies, etc.). The chief ongoing maintenance would be the periodic replacement of DS key pairs and user credentials, and detecting and responding to credentials compromises.

3.4 Security Criticisms

Several criticisms could be directed against DSs. For one, authentication methods like passwords are generally considered much weaker than private keys[51,52], but DSs would use strong authentication protocols[5,6,24], forcing attackers to contact the DS to verify each guess. A DS could detect and rebuff guesses by slowdown or lockout mechanisms, and could deter them by attempting to trace the attacker and respond. Using randomly assigned passwords or enforcing a password-strength requirement should give adequate security for many scenarios. For high security, one-time password devices can be used. Since authentication credentials are easier to use than private keys and thus easier to deploy widely and since abuse can be detected and monitored, we believe that in most scenarios our approach is more secure than giving users private keys.

We have made much of the ability of DSs to detect and manage compromises. We will give an example of how this might work: Alice would receive a weekly usage statement from her DS. For each private key operation Alice had performed, the statement would list the date, time, IP address of the requestor, and perhaps a plaintext string generated by Alice's end-user software saying things like "Signing document ContractForBob" or "Decrypting message from Charlie with subject 'Meeting notes'". Alice would review this usage statement and compare it against a local log of her activities; if she noticed illicit uses she would contact the DS and repudiate them, then cancel her credential and begin the process of getting a new one. Alice's DS could proactively notify the DSs that were on the other end of any forged message authentication codes, faked sessions, or illicitly decrypted messages, or perhaps could publish a revocation list containing these which other DSs could check, and if one of them noticed that one of its users' communications was affected, then this DS would notify the relevant user by email or phone that certain of his communications had been compromised. A DS could also monitor requests in real time to detect suspicious patterns of activity such as users performing operations during nonbusiness hours or when on vacation, or the decryption of a tripwire message or authentication with a duress code; when suspicious activity was detected the DS could prompt the user for a backup password, lock him out, pretend to be unreachable, trigger an alarm, generate a warning message to the user's email address, etc..

This should be contrasted to how a PKI handles compromise or loss of a private key. If Alice's private key is stolen covertly, the attacker could decrypt all messages encrypted to her and could authenticate or forge her signature without her receiving any indication that this is occurring. If she does begin to suspect something, she will have no way of knowing when the attack occurred and how much data was compromised. To be safe, she might have to repudiate everything ever signed under the private key, and consider all messages encrypted to it as compromised. Since PKI users typically have long-lived key pairs, this compromise could affect a year or two's worth of data. In sum, compromise of a private key is catastrophic in PKI; when using DSs, compromise of a credential can be quickly detected, limited, and recovered from.

Another security criticism is that DSs provide a high-value target for attacks. This is true: DSs are completely trusted, not partially trusted like CAs. If a DS was compromised, its private key could be used to decrypt any messages encrypted to its users, forge any of their signatures, or fake any of their authentications. To reduce the risk of cryptanalytic attack, DSs should be given large key pairs[53], and should change them frequently. DSs should also destroy old private keys once these reach a certain age, so a compromise of the entire DS will only expose a limited amount of traffic; this policy could impact the availability to users of their own data; we will look at ways to mitigate this shortly. The impact of compromise would also be lessened if users availed themselves of protocols with perfect forward secrecy[54,55] when possible.

As a matter of prudence DSs should be kept in a physically secure location, and their private keys and credentials-verifying data should only be handled within a tamper-resistant secure coprocessor[21]. This may not keep them safe from someone willing to take the DS offline and who has the technical resources and time to dissect the module and defeat its safeguards, but it should prevent covert theft of cryptographic data by attackers with intermittent physical access. When all these steps are taken (large keys with short lifetimes that are deleted at a certain age, physical protection of DSs, hardware protection of cryptographic data), we believe the security advantages offered by DSs outweigh their disadvantages. We also believe that it is not possible to significantly reduce trust in a DS while preserving its advantages; some private key delivery servers store private keys encrypted by user passwords so as to hide these keys from the server itself, but the server could always launch an offline attack and probably recover any user's password and private key.

3.5 Performance and Privacy Criticisms

A third criticism is that DSs are a potential performance bottleneck because of the computational cost of asymmetric cryptography. Given that computers are fast and getting faster, and that accelerator cards costing less than two-thousand dollars currently measure their speed in thousands of 1024-bit operations per second[56], we think DSs could offer adequate performance for a reasonable price. Moreover, centralizing computation has the advantage that weak client devices such as PDAs, cellphones, chat devices, web appliances, etc., would not have to perform the calculations themselves; they could authenticate once to the DS using lower bit length or elliptic curve calculations[57,58], then request multiple different operations in a single DS session (decrypting dozens of messages, encrypting and signing others, etc.).

Regardless, if the cost of constantly performing asymmetric operations was deemed too high, a community of DSs could use Diffie-Hellman key pairs. These key pairs would allow each DS in every pair to perform a single asymmetric calculation to determine the shared symmetric key it has with its partner, which it could then cache and reuse. With the appropriate protocols and data formats these symmetric keys could be used for point-to-point security; they could not be used for signatures (i.e. point-to-many message authentication) or anonymous encryption, but in a community not requiring these operations the use of Diffie-Hellman key pairs could eliminate asymmetric operations except for periodic key agreements.

A fourth criticism is privacy. A DS can monitor all operations performed by its users, and thus can harvest data about whom users communicate with and when, what their work habits are, etc.. This is unavoidable; we can only hope the security benefits of monitoring and the ease of use benefits of DSs overwhelm this deficiency. Theoretically, users could contact DSs using pseudonyms and untraceable channels, and could distribute their operations under multiple different names and DSs, and perhaps even blind[59,60] the data they send to a DS so that it could not correlate user operations with intercepted communications, but users capable of all this probably don't need DSs in the first place. A more feasible approach to privacy might be for a secure coprocessor at the DS to encrypt audit trail records so as to keep them hidden even from the DS administrator, and only make the relevant records available to authenticated users[22].

3.6 Communications Criticisms

The most serious criticisms involve the online communications required between users and DSs. The channel between a user and DS may be subject to denial of service attacks or traffic analysis[20], and may suffer performance or availability problems. Inside an enterprise network we believe these are minor issues. On the Internet, DSs should not be used for important time-critical communications, since an attacker or transient network failure could render the DS unreachable for a period of time. Internet DSs should also not be used when extremely high performance is required, or when the mere fact that a communication is occurring needs to be hidden. Anyone designing DSs for Internet use should take extreme care that the protocols are not exploitable and that they offer a measure of resistance to denial of service attempts (by using stateless cookies, for example[61]). Anyone deploying DSs on the Internet should take measures such as deploying redundant DSs in separate locations with high-speed network access and ensuring a competent staff is on hand to ward off crises.

Another concern with online communications is that users cannot secure new communications or read encrypted old ones when they are in environments without network access. One partial solution to this, and to the latency DSs introduce into asymmetric operations, is a persistent cache of symmetric keys and hash values stored on the user's machine. Whenever a DS is contacted to perform a decryption or verification, the result would be stored in this cache, and from then on every time the user opened the same file or read the same email these values would be fetched locally. The cache would be encrypted, so that whenever a user logged onto his computer he would have to contact the DS to decrypt it. This would keep data in a stolen computer secure and help detect illicit access (for example, by a coworker who knew your password and read your emails while you were at lunch).

If you knew you were going to be offline for a period of time, you could leave the cache in a decrypted state so that you could continue to access secure emails and files without DS contact. Another benefit of this cache is that it would keep alive messages that were encrypted to private keys that the DS had expired. To ensure that the cache itself does not expire, it could be encrypted at the DS under a special long-term key, or alternatively, every time the user contacted the DS the cache could be re-encrypted under the DS's current private key. To reduce the dangers of cache compromise, users should be able to review and purge the contents of their cache. This cache would not allow users to perform new operations when offline, but since the user is offline and thus presumably not in a position to communicate anyways, we consider this acceptable.

3.7 Delegate Server Interoperability

Despite all our arguments, DSs are inappropriate for some environments. If a user does not completely trust anyone but himself, if offline operation is important or constant contact with a DS is too vulnerable, inefficient, or unreliable, or if no administrators are willing to maintain a DS for this user, then he will have to possess his own key pair and interact with PKI on his own. Interoperability between non-DS and DS users is assured because a non-DS user can be viewed as merely a special-case DS: a DS's key pair is used on behalf of many people; a non-DS user's key pair is used on behalf of only one (himself).

In some environments, users may wish to use DSs for some operations but not all. For example, a user may wish to manage his own key pair but use a DS for public key operations (i.e. encrypting and verifying), thus freeing himself from the burden of path computation. Alternatively, he might feel more secure using the DS's key pair, but prefer to establish trust relationships with others himself.

3.8 Alternatives

Our criticisms of PKI are not novel. Various proposals have attempted to address the vulnerability and inconvenience of private key transport and the difficulty and expense of path construction. One approach is not to use PKI at all, but to use an infrastructure like Kerberos[28,29], which is entirely based around symmetric key trust relationships. We feel asymmetric cryptography has significant advantages in minimizing the trust and availability requirements placed on infrastructure nodes. However, there is a proposal to use asymmetric cryptography for cross-realm authentication[62] in Kerberos which would realize these advantages but still allow users to authenticate to their local server using passwords. The resulting hybrid infrastructure is quite similar to what we are proposing. The difference is that Kerberos only supports the establishment of symmetric keys between clients, whereas DSs allow clients to perform asymmetric operations such as signatures or anonymous encryptions. Also, whereas Kerberos requires users to operate under an online server, DSs are optional, and DS clients could seamlessly interoperate with conventional PKI users.

Turning now to proposals for improving PKI, one approach to private key transport that we have already mentioned is the use of private key delivery servers[16], which make the private key more transportable but leave it vulnerable to theft and offline abuse. Another approach is to use proxy certificates[63], which are issued under a user's regular certificate or under another proxy certificate but are only valid for a limited period of time and for a restricted set of uses. The generation of proxy certificates could be performed by the user on his local machine or by an online service that issues proxy certificates to users under proxy certificates that users had previously issued to it[64]. The advantage of the proxy certificate approach is that long-lived end-entity private keys can be kept in a highly secure environment while the more exposed proxy certificate private keys are given limited validity periods and privileges so as to minimize the damage done by a compromise. Nonetheless, like any approach that gives users control of private keys, the security benefits of auditing and instant revocation are not available; in addition, the transient nature of proxy certificates makes them unsuitable for message confidentiality. Short-lived certificates issued by online CAs[65] have the same disadvantages, but eliminate the need for long-lived end-entity certificates, while requiring greater trust in the online service since it possesses a long-lived CA private key instead of short-lived proxy keys issued to it by various users.

An approach more like ours is the use of virtual smart card servers[66], where each user's private key is stored at a server which the user authenticates to and requests operations from. These servers provide the same portability, auditing, and message confidentiality benefits as DSs. As for path construction, protocols like DPV/DPD[43] and XKMS[67] have been proposed to allow clients to offload path construction to servers, and we will assume that these work adequately.

Now if virtual smart card servers in conjunction with path construction servers accomplish the same things as DSs but work with current PKI protocols and data formats, isn't that good enough? Why bother to add explicit support for DSs? For a few reasons: For one, storing end-entity private keys on a server abuses certificate semantics: someone verifying an end-entity signature will have no way of knowing that the corresponding private key was actually in the possession of a third party. This is a significant fact and should be

somehow represented. For another, a virtual smart card server requires a separate certificate for each user; exchanging and updating these is inefficient and will reveal much information about enterprise members, including their affiliation with the enterprise, their contact information, and the revocation status of their private keys (which an attacker can check to determine whether his compromise of a private key has been detected, for example). A DS would need only a single certificate to represent an entire enterprise, and wildcards within the name forms (such as DNS names, IP addresses, telephone numbers, etc.) would not reveal anything about the enterprise's internal structure.

Another problem with current PKI technologies is that you can only revoke keys from a particular date and time, you cannot revoke particular operations. A DS could allow the user to sift through audit trails after a compromise and revoke private key operations on a fine-grained basis. Another advantage of DSs is that if DS messages are treated as opaque by clients, then DSs acquire significant flexibility in terms of algorithms, certificate formats, etc., and clients are shielded from these details. For all these reasons (improved semantics, fewer certificates, fine-grained revocation, shielding client software from infrastructure), we believe that it is worthwhile to insert DS support into current PKI protocols and data formats. We will turn our attention to this in the next section.

4 Protocols and Data Formats

Below is an example protocol demonstrating DS-secured messages. We assume all communications between clients and DSs are mutually authenticated and confidential.

C1, C2: end-users
S1, S2: Delegate Servers for the respective end-users
D1, D2: Delegate Servers' private keys (RSA-like)
E1, E2: Delegate Servers' public keys (RSA-like)
k: symmetric encryption key
m: message
h(): hash function
k(): symmetric encryption function
D1(): asymmetric signature function
E2(): asymmetric encryption function

Signed Message
C1 → S1: h(m),C1
C1 ← S1: D1(h(m),C1)
C1 → C2: m,D1(h(m),C1)
C2 → S2: h(m),C1,D1(h(m),C1)
C2 ← S2: true|false

Encrypted Message
C1 → S1: k,C2
C1 ← S1: E2(k,C2)
C1 → C2: k(m),E2(k,C2)
C2 → S2: C2,E2(k,C2)
C2 ← S2: k

Signed and Encrypted Message
C1 → S1: h(m),C1,k,C2
C1 ← S1: D1(h(m),C1),E2(k,C2)
C1 → C2: k(m,D1(h(m),C1)),E2(k,C2)
C2 → S2: C2,E2(k,C2)
C2 ← S2: k
C2 → S2: h(m),C1,D1(h(m),C1)
C2 ← S2: true|false

If the sending and/or receiving clients were not using DSs, the messages sent between clients would be the same but the asymmetric keys D1 and E2 might refer to the sender's private key or the receiver's public key instead of to the corresponding DS keys, and clients could perform the processing themselves without engaging DSs. In fact, clients could always perform public key operations (i.e. encrypting and verifying) without engaging their DSs, so we should be very clear that only private key operations are rigorously auditable.

When DSs are employed, clients need only a minimal understanding of the DS data blocks. From the perspective of client software, the protocol looks like:

Signed and Encrypted Message (client perspective)
C1 → S1: h(m),C1,k,C2
C1 ← S1: X,Y
C1 → C2: k(m,X),Y
C2 → S2: C2,Y
C2 ← S2: k
C2 → S2: h(m),C1,X
C2 ← S2: true|false

This gives the protocol a pluggable structure, allowing the DS blocks to change without affecting client software (to incorporate a new asymmetric algorithm or data format, for example).

4.1 Operation Certificates

To emphasize the differences between the DS and non-DS approaches, consider what a non-DS signed and encrypted message would look like:

Signed and Encrypted Message (without DSs)
C1 → C2: k(m,D1(h(m)),E2(k)

First, since the asymmetric keys uniquely identify end-entities, there is no need to bind sender or receiver names into the data format. Second, there are obviously no sideband protocol exchanges with DSs. When retrofitting DS support, then, we must determine some standard representation of the DS signature and encryption blocks X and Y which will allow us to represent operations and the names they apply to in a packaged format whose processing can be delegated to DSs.

One way to address these tasks is to embed names into the data structures used to represent signed hashes or encrypted keys. A name that was bound to a signature would be a declarative statement from the signer about whom the signature was produced on behalf of. A name that was bound to an encryption would be an imperative statement to the encryptee about whom the encrypted data is destined for.

For example, to produce a delegated XML Signature[68], a client could authenticate to his DS and forward it a ds:SignedInfo containing the hash values the client would like signed. The DS would add a SAML[9] Authentication Assertion as a ds:SignatureProperty, then sign the resultant ds:SignedInfo and return a ds:Signature to the client. The Authentication Assertion would name the client, the authentication method he used, and advice or conditions relating to these. To validate an XML Signature using a DS, a client could validate each hash within the ds:SignedInfo himself, then forward the ds:Signature structure to his DS and receive back a boolean.

To produce an XML Encryption[69] targeted to a DS client, one would encrypt some data with a symmetric content encryption key, then generate an XACML[70] xacml:policyStatement that expresses to whom you would like the content encryption key delivered, and encrypt this xacml:policyStatement with a symmetric policy encrytion key. Then one would generate a symmetric key encryption key and encrypt both the content and policy encryption keys with it, and finally encrypt the key encryption key with the DS's public key. The client who received all these data would authenticate to his DS and forward them to it. The DS would recover the content encryption key and the xacml:policyStatement, then evaluate the policy statement against the client's Authentication Assertion, and return the content encryption key to the client if access is permitted. The use of an access control language like XACML would allow the sender to specify the intended recipients in sophisticated terms (i.e.: "give this key to Bob or Carol, but only if they have a Top Secret clearance, and only if they authenticate with a hardware token").

To verify a DS-produced signature, or encrypt to a DS client, one must be capable of determining the DS public key. We could incorporate a flag into a ds:KeyInfo to represent DS public keys, and perhaps even add an xacml:policyStatement into a ds:KeyInfo to express to whom and for what the key should be used for. Someone wanting to encrypt a message to Bob could perform an XKMS[67] query and receive back an explicitly flagged DS key, and would thus know to embed an xacml:policyStatement into the encryption to represent the intended final recipient.

But XKMS is only intended as an interface to PKI, so this raises the question of how we can represent DSs within PKI data formats. For example, we would need to modify the X.509 certificate format to support DS certificates as opposed to CA or end-entity certificates. DS certificates would be like CA certificates in that they are authoritative over some group of users (they should support the name constraints extension to express which group), but like end-entity certificates in that the corresponding private key can perform signatures or be encrypted to directly. We could add a boolean into the basic contraints extension to identify DS certificates (which might also be CA certificates, allowing a DS to not only perform operations for clients, but perhaps issue these clients short-lived certificates).

We might also wish to retrofit DS support into PKI protocols that don't use XML, such as SSL[4] or S/MIME[71]. As we recall, adding DS support requires a standard format for representing signature and encryption operations with names bound into them. In XML there are standard formats for representing signatures and encryptions which we could easily add names into. In the X.509 PKI world there are not; however, instead of binding names into operations, we can add operations into bound names. In other words, we can generalize the notion of certificates so that instead of only binding names to public keys, certificates can bind names to hashes as well, and thus represent delegated signatures, and we can also invert the notion of signed certificates to yield encrypted certificates, which are imperative requests that a binding should be made to exist in the future, instead of declarative assertions that a binding did exist in the past.

In more detail, consider an X.509 end-entity certificate. Typically such a certificate is said to bind a name to a key. In truth, it binds not only a name, but also a serial number so the certificate can be referred to later and possibly revoked, a validity interval which delimits the binding in time, and a policy which clarifies the binding's semantics. And when we say that these things are bound to a key, we really mean that they are bound to the particular operations performed by this key: that is, that they are attributes of the signatures which it verifies and the encryptions it can be used to produce. In other words, an X.509 certificate is a mechanism for binding (within the limits of a validity period and policy) an end-user name and a serial number to operations as expressed through the indirection of

a public key.

It seems logical, then, to use certificates to bind these same attributes directly to particular operations. For example, consider an end-entity who wants to sign a document with his private key. He could hash the document and then collect this hash along with a serial number, a validity interval, and a policy, and then use his private key to sign these, producing a signature operation certificate (OC). The serial number would allow him to later revoke this particular signature by including its number in a revocation list. The validity interval would allow him to represent the time period over which he is asserting this binding. The policy would allow him to express the particular semantics of his signature on this document. Someone verifying this signature should validate the entire certificate chain, including first the CA certificates, then the end-entity certificate, and finally the OC, before extracting and checking the hash value inside the OC.

An encryption OC would be similar to a signature OC but would contain a symmetric encryption key instead of a hash value, and would be encrypted to the target's public key, instead of signed by the issuer's private key. The policy identifier would identify a request instead of a statement: that is, instead of a statement from the signer saying "I authenticated Alice to degree X and assume liability Y for the assertion that this data is associated with her", it would say "Please authenticate Bob to degree X and only deliver this data to him if you are willing to assume liability Y". One difference between signature and encryption OCs is that signature OCs represent past occurrences, whereas encryption OCs represent conditions on future occurences (mirroring the distinction between SAML assertions and XACML policies). Thus while signature OCs would be similar to end-entity certificates in that they bind a particular name, encryption OCs would be like CA certificates in that they might bind a range of names (using the name constraints extension), representing all the users who would be allowed to decrypt this data.

Looking back at our protocol diagrams, the D1(h(m),C1) blocks represent signature OCs, and the E2(k,C2) blocks represent encryption OCs. The chief problem with OCs is that they don't yet exist: current cryptographic protocols and data formats such as CMS[72] (used by the S/MIME email security standard) or TLS[73] (derived from SSL) would need surgery to support them. Below we will consider exactly what this entails.

A signature OC will be mostly identical to an end-entity OC except that the issuer field will refer to the end-entity certificate or DS certificate that issued it, and the subjectPublicKeyInfo field will be replaced by digestAlgorithm and digestValue fields. An encryption OC will be a little different; in particular, its top-level structure will be something like this:

```
EncryptionOperationCertificate ::= Sequence{
  encryptedCertificate  EncryptedCertificate
  encryptionAlgorithm AlgorithmIdentifier
  encryptedKey          BIT STRING
  target                TargetIdentifier }
```

Instead of this:

```
Certificate ::= Sequence{
  tbsCertificate        TBSCertificate
  signatureAlgorithm    AlgorithmIdentifier
  signatureValue        BIT STRING}
```

It may be desirable to support signature and encryption OCs that have both an issuer and a target, so that hash values and encryption keys could be transmitted securely using key agreement algorithms, and this could be done with straightforward extensions to the EncryptionOperationCertificate.

Clients could create and process OCs on their own or by authenticating to DSs and engaging in a request/response protocol. We could use TLS for confidential and authenticated session establishment, and modify it to support SRP for mutual authentication[74] between the client and server. The request/response protocol would allow clients to request signature OCs along with the certificate chains leading up to them by sending hash values and to-be-signed attributes to DSs, and to request encryption OCs by sending symmetric encryption keys and the names of intended recipients to DSs. We would also want to allow clients some input into the validity and policy fields of the OCs, and allow clients to retrieve the certificate chain up to their DS in a separate step from procuring an OC, for use in protocols where one party sends a certificate chain to a second who then encrypts something to the first's certificate (such as TLS). To process OCs (i.e. to verify signatures and extract symmetric encryption keys) would involve similar protocol exchanges.

OCs would work with revocation-checking mechanisms such as CRLs and OCSP. The issuer (whether an end-entity or DS) would be capable of revoking signature OCs, and the target (whether an end-entity or DS) would be capable of revoking encryption OCs. Reason codes should be added that are suitable for use by DSs and end-entities. For example, DSs should be able to specify that an operation was revoked because it was accessed using stolen credentials. Revocation-checking of OCs would not need to take place for online operations where timeliness was guaranteed (such as verifying a signature OC on a nonce). For operations where the overhead of retrieving and checking CRLs is too great, revocation-checking can be deferred and done periodically: for example, a DS might download all CRLs only at midnight every day and then

compare them against its audit logs to determine if any of its users were affected. For point-to-point operations (i.e. operations involving key agreement, or where signature and encryption OCs have been cryptographically linked in some way), the DSs can notify only the affected parties instead of having to make the revocation public.

4.2 Using Operation Certificates

Finally, we need to add OCs into application protocols and data formats. These formats already have ways of representing signed hashes and encrypted keys, and we will simply replace these older representations with the corresponding OCs. For example, a CMS SignerInfo could be changed from something like this:

```
SignerInfo ::= Sequence{
  version              CMSVersion
  sid                  SignerIdentifier
  digestAlgorithm      DigestAlgorithmIdentifier
  signedAttrs          SignedAttributes
  signatureAlgorithm   SignatureAlgorithmIdentifier
  signature            SignatureValue
  unsignedAttrs        UnsignedAttributes}
```

To this:

```
NewSignerInfo ::= Choice{
  oldSignerInfo        SignerInfo
  opSignerInfo         OperationSignerInfo}

OperationSignerInfo ::= Sequence {
  version              CMSVersion
  signOpCert           SignatureOperationCertificate
  unsignedAttrs        UnsignedAttributes}
```

The sid, digestAlgorithm, signatureAlgorithm, and signature fields would all be replaced by the signature OC, and the signed attributes could be incorporated into the OC as extensions. To add DS-based encryption to CMS, we could extend the RecipientInfo type with:

```
OperationRecipientInfo ::= Sequence{
  version              CMSVersion
  encryptOpCert        EncryptionOperationCertificate}
```

To add DS support to TLS we could similarly replace the Signature structure with a signature OC and replace the EncryptedPreMasterSecret with an encryption OC. On these lines, we believe any public key protocol or format (such as ssh[75], IPsec[76], OpenPGP[18], etc.) could be retrofitted to use OCs.

In sum, OCs are a powerful primitive even apart from DSs. OCs extend certificate validation to the level of particular operations, allowing policies and validity periods to be bound to operations, signed and encrypted attributes to be incorporated into them, and revocation-checking to occur upon them. By defining a standard structure that uses asymmetric keys to secure this information and bind it to hash values and symmetric keys, protocol designers are given a higher-level building block that makes their job easier. With DSs, OCs become even more valuable, since OCs allow the binding of names to particular operations and can be easily passed back and forth between clients and DSs and embedded in protocols.

It may be objected that we are abusing the notion of certificates, but we feel that we are generalizing it in a coherent way. A conventional certificate authenticates a binding between attributes such as names and a public key and qualifies this binding via policies, validity periods, etc.. This public key can then be used to produce authenticated or confidential bindings between these attributes (or some subset of them) and further data. In the case where an authenticated binding is produced between attributes and another public key, this is called a certificate.

In our opinion, this is a restrictive notion of certificates: the idea of a qualified binding between attributes and data is sufficiently important and general that the same data format and terminology should be used when binding attributes to data that are not public keys (i.e. OCs) and when producing confidential instead of authenticated bindings (i.e. encryption OCs versus signature OCs). By treating all such bindings consistently, the scope of concepts such as revocation-checking, policies, and validity intervals is increased, and the bindings are packaged into a standard format which makes it easy to reuse them in the context of different protocols and easy to delegate their processing to DSs. This approach seems promising, but it clearly needs a much more thorough analysis and explication then we have provided here.

5 Conclusion

Delegated cryptography splits the problem of end-to-end security into an intra-enterprise portion that can be addressed with authentication techniques and an inter-enterprise portion that can be addressed with PKI. This exploits the strengths and avoids the weaknesses of both technologies: Authentication techniques are easy to use and widely deployed, but can only secure interactive sessions between two parties. PKI can secure sessions or messages between a large number of parties, but imposes complex and difficult burdens on these parties. By using authentication techniques to access a PKI-enabled server we can confine the burdens of PKI to a single point within an enterprise while making its benefits available throughout.

There are proposals to improve authentication techniques by having one authentication stand in for several (single sign-on), and to improve PKI through piecemeal delegation of various functions (private key storage, path construction, etc.). We believe these proposals are in the right direction but don't go far enough. We think authentication should be used to access more than simply further authentications, and that delegation should be pushed to its logical extreme. Taken together, these points indicate an infrastructure that would be easy to use, easy to write software for, full-featured, highly secure, and efficient, and could be built on top of the data formats and protocols in use today. We encourage and hope to participate in further research in this direction.

Acknowledgements

We thank Sayan Chakraborty and the anonymous reviewers for their encouragement and helpful comments on the ideas and organization of this paper.

Notes

[1] Actually, we could expect much more from a cryptographic infrastructure, and from cryptography in general: we might want notary, timestamping, and nonrepudiation services, protocol support for things like voting, simultaneous contract signing, and digital cash, steganography and watermarking functionality, anonymous communications, etc.[77]. Here we focus on the more prosaic objectives of confidentiality and authentication, but it would be interesting to explore more exotic uses of DSs.

[2] Much of our argument against conventional PKI, and our proposed solution, was anticipated by Don Davis' paper "Compliance Defects in Public-Key Cryptography"[78]. In particular, after reviewing PKI's advantages in reducing trust, availability, performance, and reliability demands on the infrastructure, he points out that "these attractive features come at the cost of transferring corresponding burdens onto users". His suggestion, similar to ours, is a hybrid system: "We can combine both cryptosystems' administrative benefits, by restricting public-key deployment to servers, and by using symmetric-key protocols for desktop clients". This paper is highly worth reading, and provides further evidence for many of our arguments.

[3] Here as elsewhere we assume that principals possess global names and form their trust relationships in terms of these. This approach has been criticized: often the name of some party to a communication is less relevant than some attribute of this party (such as his organizational affiliation, security clearance, credit rating, etc.)[38]. If trust relationships are expressed and calculated in terms of names then some other mechanism (such as an access control list) must be used to map identities to these authorizations or attributes, which is both clumsy and a threat to privacy since user identities are exposed in situations where they are not strictly necessary. We agree with this criticism, but we believe the debate is orthogonal to our approach: DSs could wield attribute or authorization certificates just as easily as identity certificates. For simplicity of presentation we will continue to speak in terms of names but no loss of generality should be assumed.

References

[1] R. Owens, *One-Time Passwords: Functionality and Analysis*, October 2000
http://rr.sans.org/authentic/onetime2.php

[2] N. Haller, C. Mertz, P. Nesser, and M. Straw, *RFC 2289: A One-Time Password System*, February 1998
http://www.ietf.org/rfc/rfc2289.txt

[3] A.K. Jain, R. Bolle, and S. Pankanti, *Biometrics: Personal Identification in Networked Society*, Kluwer, 1991
http://www.wkap.nl/prod/b/0-7923-8345-1

[4] A.O. Freier, P. Karlton, and P.C. Kocher, *The SSL Protocol Version 3.0*, November 1996
http://www.netscape.com/eng/ssl3/draft302.txt

[5] T. Wu, *The Secure Remote Password Protocol*, Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium, March 1998
http://www-cs-students.stanford.edu/~tjw/srp/ndss.html

[6] T. Wu, *RFC 2495: The SRP Authentication and Key Exchange System*, September 2000
http://www.ietf.org/rfc/rfc2945.txt

[7] Microsoft .NET Passport
http://www.microsoft.com/myservices/passport/security.doc

[8] The Liberty Alliance Project
http://www.projectliberty.org/

[9] P.H. Baker, E. Maler, et. al, *Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML)*
http://www.oasis-open.org/committees/security/docs/

[10] W. Diffie and M.E. Hellman, *New Directions in Cryptography*, IEEE Transactions on Information The-

ory, 22, 1976
http://citeseer.nj.nec.com/diffie76new.html

[11] L.M. Kohnfelder, "Toward a Practical Public-Key Cryptosystem", B.Sc. thesis, MIT Department of Electrical Engineering, 1978

[12] C. Daniel, *Internet Security cannot be left to technologists alone*, Financial Times, September 2001
http://specials.ft.com/ftit/FT34WRFC6RC.html

[13] J. Lewis, *PKI Won't Hit The Mainstream Until Vendors Reduce Complexity*, InternetWeek, January 2001
http://www.internetweek.com/columns01/lewis010801.htm

[14] B.D. Reimers, *PKI's Are Still Tough To Deploy*, InternetWeek, April 2001
http://www.internetweek.com/security/secure040901-1.htm

[15] GAO, *Information Security: Advances and Remaining Challenges to Adoption of Public Key Infrastructure Technology*, Item No. 0546-D; SuDocs No. GA 1.13:GAO-01-277, February 2001
http://www.gao.gov/new.items/d01277.pdf

[16] A. Arsenault and S. Farrell, *RFC 3157: Securely Available Credentials – Requirements*, August 2001
http://www.ietf.org/rfc/rfc3157.txt

[17] S. Garfinkel, *PGP: Pretty Good Privacy*, O'Reilly & Associates, 1995
http://www.oreilly.com/catalog/pgp/

[18] J. Callas, L. Donnerhacke, H. Finney, and R. Thayer, *RFC 2440: OpenPGP Message Format*, November 1998
http://www.ietf.org/rfc/rfc2440.txt

[19] A. Whitten and J.D. Tygar, *Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0*, Proceedings of 8th USENIX Security Symposium, August 1999
http://www-2.cs.cmu.edu/~alma/johnny.pdf

[20] J. Raymon, *Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems*, Workshop on Design Issues in Anonymity and Unobservability, 2000
http://citeseer.nj.nec.com/454354.html

[21] S.W. Smith and S.H. Weingart, *Building a High-Performance, Programmable Secure Coprocessor*, Computer Networks (Special Issue on Computer Network Security), 31, pp. 831-860, April 1999
http://www.research.ibm.com/secure_systems/papers/arch.pdf

[22] S.W. Smith and D. Safford, *Practical Private Information Retrieval with Secure Coprocessors*, IBM Research Report RC-21806, July 2000
http://www.research.ibm.com/secure_systems/papers/rc21806.pdf

[23] J. Daugman, *High Confidence Visual Recognition of Persons by a Test of Statistical Independence*, IEEE Transactions on Pattern Analysis and Machine Intelligence, v. 15 no. 11, pp. 1148-1161, November 1993

[24] S.M. Bellovin and M. Merritt, *Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks*, Proceedings of the IEEE Symposium on Research in Security and Privacy, May 1992
http://www.research.att.com/~smb/papers/neke.ps

[25] ITU-T Rec. X.509, *The Directory: Public-key and attribute certificate frameworks*, March 2000
http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-X.509

[26] R. Housley, W. Ford, W. Polk, and D. Solo, *Internet X.509 Public Key Infrastructure Certificate and CRL Profile*, January 1999
http://www.ietf.org/rfc/rfc2459.txt

[27] R. Housley and Tim Polk, *Planning for PKI*, John Wiley & Sons, Inc., 2001
http://www.wiley.com/cda/product/0,,0471397024,00.html

[28] J.G. Steiner, B.C. Neuman, and J.I. Schiller, *Kerberos: An Authentication Service for Open Network Systems*, Proceedings of the Winter 1988 Usenix Conference, pp. 191-202, February 1988
ftp://athena-dist-mit.edu/pub/kerberos/doc/usenix.ps

[29] J. Kohl and B.C. Neuman, *RFC 1510: The Kerberos Network Authentication Service (V5)*, September 1993
http://www.ietf.org/rfc/rfc1510.txt

[30] C. Rigney et. al, *RFC 2865: Remote Authentication Dial In User Service*, June 2000
http://www.ietf.org/rfc/rfc2865.txt

[31] W. Yeong, T. Howes, and S. Kille, *RFC 1777: Lightweight Directory Access Protocol*, March 1995
http://www.ietf.org/rfc/rfc1777.txt

[32] M. Wahl, T. Howes, and S. Kille, *RFC 2251: Lightweight Directory Access Protocol (v3)*, December 1997
http://www.ietf.org/rfc/rfc2251.txt

[33] RSA Security Inc., *Delivery of Ten Millionth RSA SecurID Authenticator*, Press Release, December 2001
http://www.rsasecurity.com/news/pr/011212.html

[34] P. Kocher, *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems*, Advances in Cryptology–CRYPTO '96, Springer LNCS

1109, pp. 104-113, 1996
http://www.cryptography.com/resources/whitepapers/TimingAttacks.pdf

[35] P. Kocher, *Differential Power Analysis,* Advances in Cryptology-CRYPTO '99, Springer LNCS 1666, pp. 388-397, 1999
http://www.cryptography.com/resources/whitepapers/DPA.pdf

[36] R.J. Anderson and M.G. Kuhn, *Tamper Resistance – A Cautionary Note*, Proceedings of the Second Usenix Workshop on Electronic Commerce, pp. 1-11, November 1996
http://www.cl.cam.ac.uk/~mgk25/tamper.html

[37] ITU-T Rec. X.500, *The Directory: Overview of concepts, models and services*, February 2001
http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-X.500

[38] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen, *RFC 2693: SPKI Certificate Theory*, September 1999
http://www.ietf.org/rfc/rfc2693.txt

[39] P. Gutmann, *X.509 Style Guide*, October 2000
http://www.cs.auckland.ac.nz/~pgut001/pubs/x509guide.txt

[40] P.C. Van Oorschot, W.S. Ford, S.W. Hillier, and J. Otway, *Method for efficient management of certificate revocation lists and update information*, U.S. Patent 5,699,431, December 1997
http://www.uspto.gov/

[41] D.A. Cooper, *A More Efficient Use of Delta-CRLs*, Proceedings of the 2000 IEEE Symposium on Security and Privacy, pp. 190-202, May 2000
http://csrc.nist.gov/pki/documents/sliding_window.pdf

[42] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams, *RFC 2560: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP*, June 1999
http://www.ietf.org/rfc/rfc2560.txt

[43] D. Pinkas, *Internet Draft: Delegated Path Validation and Delegated Path Discovery Protocols*, work in progress, July 2001
http://www.ietf.org/internet-drafts/draft-ietf-pkix-dpv-dpd-00.txt

[44] R. Housley and T. Polk, *Planning for PKI*, John Wiley & Sons, Inc., chapter 11, 2001
http://www.wiley.com/cda/product/0,,0471397024,00.html

[45] B. Kaliski, *RFC 2314: PKCS #10: Certificate Request Syntax Version 1.5*, March 1998
http://www.ietf.org/rfc/rfc2314.txt

[46] B. Kaliski, *RFC 2315: PKCS #7: Cryptographic Message Syntax Version 1.5*, March 1998
http://www.ietf.org/rfc/rfc2315.txt

[47] C. Adams and S. Farrell, *RFC 2510: Internet X.509 Public Key Infrastructure Certificate Management Protocols*, March 1999
http://www.ietf.org/rfc/rfc2510.txt

[48] M. Myers, X. Liu, J. Schaad, and J. Weinstein, *RFC 2797: Certificate Management Messages over CMS*, April 2000
http://www.ietf.org/rfc/rfc2797.txt

[49] X. Liu, C. Madson, D. McGrew, and A. Nourse, *Cisco System's Simple Certificate Enrollment Profile*, 2000
http://www.cisco.com/warp/public/cc/pd/sqsw/tech/scep_wp.htm

[50] S. Berkovits, S. Chokhani, J.A. Furlong, J.A. Geiter, and J.C. Guild, *Public Key Infrastructure study: Final Report*, MITRE Corporation, April 1994
http://csrc.nist.gov/pki/documents/mitre.ps

[51] D.C. Feldmeier and P.R. Karn, *Unix Password security – ten years later*, CRYPTO Proceedings, 1989
http://www.ja.net/CERT/JANET-CERT/../Feldmeier_and_Karn/crypto_89.ps

[52] T. Wu, *A Real-World Analysis of Kerberos Password Security*, Proceedings of the 1999 Network and Distributed System Security Symposium, 1999
http://www.isoc.org/isoc/conferences/ndss/99/proceedings/papers/wu.pdf

[53] A.K. Lenstra and E.R. Verheul, *Selecting Cryptographic Key Sizes*, to appear in The Journal of Cryptology, Springer-Verlag
http://www.cryptosavvy.com/Joc.pdf

[54] C.G. Günther, *An identity-based key-exchange protocol*, Advances in Cryptology-EUROCRYPT '89, Springer LNCS 434, pp. 29-37, 1990

[55] R. Shirey, *RFC 2828: Internet Security Glossary*, May 2000
http://www.ietf.org/rfc/rfc2828.txt

[56] Cryptographic Appliances, *Cryptographic Appliances Releases Two PCI Accelerators*, Press Release, August 2001
http://www.cryptoapps.com/press08072001.html

[57] V. Miller, *Uses of elliptic curves in cryptography*, Advances in Cryptology: proceedings of Crypto '85, LNCS 218, pp. 417-426, 1986

[58] N. Koblitz, *Elliptic curve cryptosystems*, Mathe-

matics of Computation, 48, pp. 203-209, 1981

[59] D. Chaum, *Blind Signatures for Untraceable Payments*, Advances in Cryptology: Proceedings of Crypto 82, Plenum Press, pp. 199-203, 1983

[60] D. Chaum, *Security without Identification: Transaction Systems to Make Big Brother Obsolete*, Communications of the ACM, v. 28, n. 10, pp. 1030-1044, October 1985
http://www.chaum.com/articles/Security_Wthout_Identification.htm

[61] P. Karn and W. Simpson, *RFC 2522: Photuris: Session-Key Management Protocol*, March 1999
http://www.ietf.org/rfc/rfc2522.txt

[62] M. Hur, B. Tung, T. Ryutov, C. Neuman, A. Medvinsky, G. Tsudik, B. Sommerfeld, *Internet Draft: Public Key Cryptography for Cross-Realm Authentication in Kerberos*, work in progress, November 2001
http://www.ietf.org/internet-drafts/draft-ietf-cat-kerberos-pk-cross-08.txt

[63] S. Tuecke, D. Engert, I. Foster, V. Welch, M. Thompson, L. Pearlman, and C. Kesselman, *Internet Draft: Internet X.509 Public Key Infrastructure Proxy Certificate Profile*, work in progress, February 2002
http://www.ietf.org/internet-drafts/draft-ietf-pkix-proxy-02.txt

[64] J. Novotny, S. Tuecke, and V. Welch, *An Online Credentials Repository for the Grid: MyProxy*, Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, August 20001
http://www.globus.org/research/papers/myproxy.pdf

[65] Y. Hsu and S.P. Seymour, *An Intranet Security Framework Based on Short-Lived Certificates*, Proceedings of the 6th workshop on Enabling Technologies Infrastructure for Collaborative Enterprises, 1997
http://www.computer.org/internet/ic1998/w2073abs.htm

[66] Secure Computing Corporation, *Virtual Smart Card Server Solution*, July 2000
http://www.securecomputing.com/pdf/safeword_plus_wp_vscs.pdf

[67] W. Ford, P.H. Baker, B. Fox, B. Dillaway, B. LaMacchia, J. Epstein, and J. Lapp, *XML Key Management Specification (XKMS)*, March 2001
http://www.w3.org/TR/xkms/

[68] D. Eastlake, J. Reagle, and D. Solo, *RFC 3275: (Extensible Markup Language) XML-Signature Syntax and Processing*, March 2002
http://www.ietf.org/rfc/rfc3275.txt

[69] D. Eastlake, J. Reagle, T. Imamura, B. Dillaway, and E. Simon, *XML Encryption Syntax and Processing, W3C Candidate Recommendation*, March 2002
http://www.w3.org/TR/xmlenc-core/

[70] S. Godik and T. Moses, *OASIS eXtensible Access Control Markup Language, Committee Draft*, April 2002
http://www.oasis-open.org/committees/xacml/docs/

[71] B. Ramsdell, *RFC 2633: S/MIME Version 3 Message Specification*, June 1999
http://www.ietf.org/rfc/rfc2633.txt

[72] R. Housley, *RFC 2630: Cryptographic Message Syntax*, June 1999
http://www.ietf.org/rfc/rfc2630.txt

[73] T. Dierks and C. Allen, *RFC 2246:The TLS Protocol Version 1.0*, January 1999
http://www.ietf.org/rfc/rfc2246.txt

[74] D. Taylor, *Internet Draft: Using SRP for TLS Authentication*, work in progress, June 2001
http://www.ietf.org/internet-drafts/draft-ietf-tls-srp-01.txt

[75] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen, *Internet Draft: SSH Protocol Architecture*, work in progress, January 2002
http://www.ietf.org/internet-drafts/draft-ietf-secsh-architecture-12.txt

[76] R. Thayer, N. Doraswamy, and R. Glenn, *RFC 2411: IP Security Document Roadmap*, November 1998
http://www.ietf.org/rfc/rfc2411.txt

[77] B. Schneier, *Applied Cryptography, Second Edition*, John Wiley & Sons, chapters 2-6, 1996
http://www.counterpane.com/applied.html

[78] D. Davis, *Compliance Defects in Public-Key Cryptography*, Proceedings of the 6th USENIX UNIX Security Symposium, July 1996
http://world.std.com/~dtd/compliance/compliance.ps