Open Software Foundation **W. Tuvell (OSF)**
Request For Comments: 98.0
December 1996

CHALLENGES CONCERNING PUBLIC-KEY IN DCE

1. INTRODUCTION

At the level of *theoretical cryptography*, public-key (PK) technology has a compelling elegance that makes it the current choice of academicians (not to mention marketers) everywhere. However, discussions with end-users make it clear that what has not yet received enough attention are the real-world system design challenges at the level of *practical engineering* that need to be answered before PK is ready for actual large-scale deployment alongside (augmenting or partially replacing) the existing (Kerberos-based) secret-key (SK) technology. The goal of this paper is to record some customer concerns on these practical matters, primarily in the context of DCE. The point of creating this record is that any PK solution provided by future DCE projects must adequately address these concerns in order to gain widespread customer acceptance.

In writing this paper, some unfashionable questions about the relationship between pure science (theoretical cryptography) and applied technology (computing systems) tend to creep in,[1] because they are on customers' minds, and they won't simply go away by themselves. Unfortunately, merely asking such questions runs the risk of being perceived as ‘‘PK bashing’’. But that is no more the intention of this paper than is its opposite, namely, blindly accepting the PK religion and proceeding to reinvent the constellation of security services with PK-only as a single-minded focus.

The use of PK in real systems is still manifestly quite immature (this really needs no proof, though some additional details are given below). However, PK has at least evolved to the point where its basic infrastructure is starting to gain some concensus. Namely, the Architecture for Public-Key Infrastructure paper ([APKI]) specifies the components (services, mechanisms) and interfaces (APIs, formats, protocols, profiles, negotiation) that make up ‘‘the’’ PKI (independently of DCE). But, consistent with its goals, the [APKI] specifications are somewhat value-free, i.e., [APKI] never questions the ‘‘system-level’’ implications of the PKI. This paper does raise those questions, in relation to DCE. The emphasis in this paper is on how the low-level PK infrastructure of [APKI] will actually be used in a full-featured, high-level distributed computing *system*, such as DCE. As will be seen, DCE's usage of the PKI is not a simple matter of ‘‘throwing the switch’’ from SK to PK technology: a large number of design decision issues must be addressed along the way.

It is beyond the scope of this paper to state *user requirements* (see [RFC 8] for that), or to write detailed *functional specifications* (that is the responsibility of organizations ultimately producing technology for DCE) for future releases of DCE. Instead, it is hoped that future releases of DCE will respond not only to the customer requirements but also to the customer concerns raised in this paper, so that customers will be confident that their voices have been completely heard.

1.1. Terminology

The terminologies ‘‘SK’’ and ‘‘PK’’ are well-defined at the level of small-scale cryptographic algorithms, but ill-defined at the level of large-scale systems of services.[2] So, rather than try to give succinct definitions here, we will sometimes use these terms a bit imprecisely and ambiguously, just as is done in practice. For example, the notion

---

1. This science vs. technology dichotomy is illuminated by a classical anecdote involving a similar dichotomy between legal theory and practical legislation. Solon (whose name survives as our word ‘‘solon’’, meaning ‘‘wise and skillful statesman’’) became known as the ‘‘lawgiver of Athens’’ after he reformed the previous draconian code of laws (written by Draco), making them altogether more humane. As related by Plutarch, when Solon was afterwards asked if he had left the Athenians the best laws that could be given, he replied: ‘‘The best they could receive.’’

2. Much the same is true of the terminology ‘‘object-oriented’’.

of key escrow makes sense for both SK and PK, but most key escrow proposals presume a PK environment, so key escrow will usually be be lumped in with the ‘‘PK(-related) technologies’’ in this paper.

The *SK technology* (or *SK infrastructure*) we start from is the existing set of security services in the DCE 1.0 and 1.1 releases: Kerberos-based authentication, PAC/ACL-based authorization (including EPACs and delegation), protected RPC (for both integrity and confidentiality), and auditing. These are augmented in DCE 1.2 by some initial PK support (reviewed in context below).

The *PK technologies* we consider are those whose security depends on either integer factorization (*IF*) or discrete logarithms (*DL*) in concrete groups (we are interested only in straightforward applications of PK, nothing advanced or esoteric). These PK technologies depend on attaching two different, but related, keys to every principal:

(a) A *public key*, which is publicly known and represents or ‘‘speaks for’’ the principal (used for verifying the principal’s signature, and for encrypting messages targeted to the principal). In a very tangible sense, the public key is the ‘‘identity’’ (or public persona) of the principal.

(b) A *private key*, secretly held by the principal and uniquely associated to the public key, but not derivable from it (used for creating the principal’s digital signature, and for decrypting messages targeted to the principal). It is knowledge of the private key that makes the associated public key usable.

The combination of a public key and its related private key is called a PK *key-pair* (however, the shorthand and ambiguous phrase ‘‘PK key’’ often also means the same thing). In fact, *two* (logically different, and preferably physically different) key-pairs should be attached to every principal — one for signing/verifying, and one for encryption/decryption (or actually, key-exchange, see the Functionality section below). And indeed, more than two key-pairs will often be attached to users (some of the reasons are discussed in the Multi-Crypto section below).

However, the whole area is fraught with subtleties and technicalities which are beyond the scope of this paper to review in detail. One especially striking difference between PK and SK is that individual PK algorithms all have their own peculiarities, while SK algorithms are pretty much all the same, at least with respect to their suitability for many basic purposes. And this is even more pronounced for the protocols based on those algorithms. For example, PK encryption and decryption is really only suitable for key-exchange (and digital signatures), not directly for bulk data protection (see the Functionality section below). As another example, some PK algorithms can be used for key exchange but not for signing, or *vice versa*. As yet another example, some algorithms which are (inappropriately) labeled ‘‘PK’’ have the property that both keys must even be kept secret (because it is possible to derive them from one another). The reader is referred to the literature for appropriate background (start with [Schneier]).

Occasionally a bias towards the more popular algorithms, either SK or PK (e.g., DES, RSA, Diffie-Hellman, El Gamal, etc.) may seem to creep into this paper, but that is often just a manner of speaking — there is no intention to exclude other algorithms. Similarly, a bias towards the RPC mode of communication may creep in, but again this is just a manner of speaking — it is expected that the security capabilities we discuss will be made available for use on non-RPC transports as well, via APIs such as GSS-API.

1.2. Structure of this Document

There are three main sections to this paper:

(a) General issues related to PK, SK, and their hybridization.

(b) Specific issues related to PK, SK, and DCE. This includes a quick review of security features supported by DCE (just some lists, to set the stage for the rest of the paper).

(c) Strawman proposal for integrating PK into DCE. The rationale for this proposal draws on the issues in the preceding two sections (and that’s the reason for structuring the paper this way).

Within the first two sections there are subsections that fall into various informal, rough, rambling and overlapping categories. But the overall thrust is clear: given that the existing security infrastructure of DCE (and the industry as a whole) is based on SK, what are the goals we can expect for a PK-based infrastructure, and how do we achieve those goals? Such questions have been raised casually from time to time in the literature, but recent events (such as electronic commerce on the Internet) have lent them with an urgency they haven’t had previously.

Appropriate to the preliminary character of this paper, all opinions are conditional, all proposals (as few as they are) tentative, and all theories suspect — pending the further debate this paper is intended to foster.

2. GENERAL ISSUES RELATED TO PK, SK, AND HYBRIDIZATION

Any general attempt to ''compare'' PK and SK, from an ideological viewpoint, is a religious issue of no interest to this paper. Instead, the meaningful engineering question for us is how PK and SK can be combined to achieve desired services. Ultimately, it is expected that any acceptable solution will entail an intelligent hybrid, using the best parts of both SK and PK, at levels sufficient to satisfy requirements. The necessity for such hybridization is already apparent at the cryptographic primitive level, namely, data (in the sense of verifiable plaintext) is not normally encrypted by PK, instead PK is used to key-exchange a SK and then the data is encrypted by the SK (for the reasons, see the Performance and Functionality sections below). Hybridization has already been successfully employed for certain tasks in DCE 1.2. And there is every reason to expect that hybridization will be useful at higher levels, too. In the following subsections we note some of the elements of the general discussion (i.e., independently of DCE) about SK, PK, and their hybridization.

2.1. Maturity

PK is more far more immature than SK. This can be seen from the fruitful research still being done on PK algorithms and protocols, and from the ongoing activities standardization committees such as IEEE P1363. Because of this immaturity, PK has not withstood the kind of peer review (cryptanalysis) that is appropriate for security technologies, so from a scientific viewpoint, PK must be considered a little more suspect than SK. In particular, in ''the doomsday scenario'', certain mathematical discoveries (fast factoring or fast discrete logarithm algorithms) could potentially destroy the security of PK. That is unlikely, though unforeseen advances are likely to continue for some time.

A particular problem is that while PK *algorithms* (a.k.a. *mechanisms*) are starting to be relatively well-understood, PK *protocols* are still poorly-understood. This fact is proven by the continuing flow of academic papers on this subject, a fair summary of which is given in [AndNeed]:

> Curiously enough, although public key algorithms are based more on mathematics than secret key algorithms, and have much more compact mathematical descriptions, public key protocols have proved much harder to deal with by formal methods. ... With symmetric algorithms it is often possible to treat the algorithms as a black box, as symmetric block ciphers have a certain amount of muddle in them. ... However, the known asymmetric algorithms are much more structured, ... [t]hus they are much more likely to interact with other things in the protocols they are used with, and we have to be that much more careful.

2.2. Complexity

PK is much more complex than SK, partly because PK algorithms and protocols are more mathematically sophisticated than corresponding SK ones, and partly because certain security services (such as certificate revocation and key escrow) are more difficult to achieve in the PK world than in the SK world. Complexity is antithetical to security, because of the assurance problem: security based on simple technology can be understood, implemented, and validated to be correct; complex technologies are then to be layered upon simple ones, in incremental steps. This dictum has been held to be a truism ever since the notion of a ''security kernel'' (or ''reference monitor'', i.e., a small, simple module through which all security-relevant events are required to flow) was invented.[3]

On the plane of ordinary human understanding (as opposed to technical code-level assurance problem, above), the complexity of both SK and PK makes them hard for non-mathematicians and non-security-specialists (including most system implementers and everyday users) to accept, and hence to have confidence in. This is not considered an insuperable technical barrier (again, as compared to the assurance problem above), but it is a sociological factor that must be dealt with. For example, how can ordinary citizens be assured that backdoors haven't been surreptitiously designed into hard-to-understand algorithms and systems?[4]

---

3. To quote [AndRoe], ''[C]omplexity is likely to lead to the subtle and unexpected implementation bugs which are the cause of most real world security failures.''.

4. The ''naturalness'' (i.e., non-artificiality) of PK is a distinct advantage here. Since PK algorithms derive from mathematical theorems and formulas related to integer factorization and discrete logarithms that occur ''in nature'' and have been around for centuries, they cannot have had backdoors designed into them. Contrast this with DES, where the backdoor question was first raised because its design criteria are only

2.3. Chain of Trust: Trustlessness vs. TTP

From a strict security viewpoint, PK is theoretically more secure than SK, because in PK secret keys are shared with nobody else, while in SK technology the keys are shared with a trusted third party (TTP) — and every trusted entity is just one more potentially weak link in the security chain (in fact, the TTP is a single point of failure for the population that trusts it unconditionally). Said another way, pure PK is considered more ''trustless'' (i.e., fewer trusted entities, and only a few secrets must be trusted ''in depth'') than SK, because PK depends only on two-party relationships while SK depends on a TTP.

The problem with this trustlessness argument is that it is only theoretical, in the sense that it examines only the basic algorithms, not the entire system. The analysis of [Davis], for example, concludes that PK is actually no more trustless than SK — PK just silently transfers to *users* certain elements of trust that may more properly belong with the security infrastructure and with administrators.

Another example of basic algorithms vs. whole systems is the way PK key-pairs are generated, in particular who gets to know the private key. If the key-pair generator is not strictly confined to the user's control, then a *de facto* TTP is introduced. E.g., if a CA or a smartcard generates the key-pairs, then it becomes a TTP (or even a key escrow agent) whether or not it is ''supposed to be''.[5]

Concerning the TTP itself, one needs to ask exactly what it is trusted to do, because there are different degrees of trust. For example, a TTP that escrows long-term private keys must be more trustworthy than one that escrows only session keys. To ameliorate this problem, many key escrow proposals include provision for secret-sharing (also known as threshold schemes or key-splitting).

2.4. Functionality: Signing & Key Exchange

Only SK, not PK, is suitable for confidentiality-protecting bulk data or verifiable plaintext, primarily for performance reasons.[6] This problem is commonly ameliorated by using PK to encrypt only small amounts of unverifiable (random) data, such as hashes or SK keys.

On the other hand, PK does support two very desirable capabilities that give it an advantage over SK:

(a) *Signing*. This is the *sine qua non* of PK. (Using RSA, this is done by encrypting a hash, such as MD5, with the RSA private key.) It has at least the following three guises (one and a half of which require a TTP):

   (i) *Persistent integrity* protection (of data), offline (as well as online). This is essential for store-and-forward and multicast communications (such as email or other queueing services). This doesn't require a TTP.

   (ii) *Non-repudiation*. The restricted meaning of this is undeniable origin authentication of data, verifiable by third parties not directly involved in the communications stream. However, the broader meaning includes all forms of denial of accountability (not just data-origination), for example origination of resource-consuming actions such as processing power. PK handles the restricted meaning non-repudiation well, but has no advantage over SK for the broader meaning — only the recording of events as they occur by a TTP is effective against non-repudiation in the broad sense. [Which raises the question: is it really necessary to use multiple mechanisms (PK signing vs. audit logging)

---

incompletely known. Compare this with the MD[2|4|5] hash algorithms, which (partially) address this problem by using naturally-occurring phenomena like pi, the square root of two, and the sine function.

5. Some system designers and enterprises do not seem to like the idea of letting users generate their own key-pairs. Reasons given range from export requirements (some countries have laws against keys being generated at the client), to key escrow reasons (commercial or government), to ''the CA can have a stronger random number generator'', to smart card administration issues (where private keys must be stored on a smart card as part of the key/certificate generation process).

6. PK may also be criticized for being somewhat vulnerable to ''chosen plaintext attack'' (by an attack that doesn't compromise the PK private key, but can potentially reveal the encrypted data). Namely, since the public key is publicly known, an attacker can repeatedly encrypt trial messages until the ciphertext is matched (this argument is made in [Schneier]). (Confounders can be helpful here. In the SK case, where no key is publicly know, confounders are commonly included, but for a different reason, namely so that multiple successive encryptions of the same data will look different to the eavesdropper.)

to deal with non-repudiation?]

(iii) *Notarization*. This means signing by a TTP, additionally stamped with a (trusted) timestamp. For example, when a CA creates a certificate with a validity timestamp in it, the CA is acting as a specialized notary service. A generalized notary service would simply bind a requester's identity and a timestamp to arbitrary data, on demand.

(b) *Key-exchange* (online or offline). This uses PK to exchange SK keys, then uses SK to encrypt bulk data. For example, Diffie-Hellman key exchange (with authentication added) avoids the cascaded-compromise problem (whereby one compromised key automatically leads to the compromise of others).

Corresponding to the above classification, PK key-pairs are usually classified into two kinds: ''signature key-pairs'' (a.k.a. *send keys*) and ''key-exchange key-pairs'' (a.k.a. *receive keys*). These two kinds of keys are sometimes treated alike, and sometimes differently. For example, for key escrow[7] purposes, the private key-exchange key is more important; for CRL purposes, the signature key is more important.

Note that support for signing in DCE RPC does not yet exist, so would have to be added (RPC currently keeps security state with the entities at the communicating endpoints, while signing binds the security state with the datastream). This would give developers a high-level interface for writing end-to-end-secure store-and-forward applications. A corresponding lower-level non-RPC API would also have to be supported (presumably that would be IDUP GSS-API). And, consistent with the view that no core services of DCE should be available via PK only, signing should be available to both SK and PK environments.

2.4.1. PK = SK + TTP

Actually, anything that PK can do can also be simulated (albeit less elegantly) with SK plus a TTP (which may be stateless, except for knowing its own secret master key) acting as a simulation server.[8] Namely, instead of using private keys to speak for principals, the TTP (which shares a long-term SK with every principal) speaks for them (the TTP also has a master SK it shares with no one else). One use of this fact is that PK(-like) *services* (such as signing) remain valid even in the unlikely event that PK *technology* itself ultimately turns out to be insecure (provided one is willing to accept a TTP). Such a TTP PK simulation server has been called various names in the literature. In [Schneier] it is called an *arbitrator*. In [DaSw], which introduces the notion of *private-key certificates* (and the user-to-user protocol implemented in DCE 1.2), it is called a *translator*. In [LABW], it is called a *relay*. It has also been called a *witness*. As an example of how such a TTP might be used, multicast (as opposed to broadcast) integrity and/or privacy can be economically implemented by the TTP maintaining the session key protected under an appropriate (e.g., group) ACL.

Such a simulation server may become a processing bottleneck; however, that is not certain. In particular, the services of the simulator may be required only infrequently, depending on the overall design, since most communications occur directly between clients and servers without the simulator's intervention, making it no worse than today's DCE, where the KDS is not a significant bottleneck. For example, if an escrow server has to be visited anyway to escrow PK session keys, the incremental overhead of an SK simulation server might be minimal. For another example, if a non-repudiation server has to be visited anyway to log operation initiations, the incremental overhead of visiting it for signature purposes (i.e., data-origin integrity and non-repudiation) might be minimal, especially considering that only a hash of the data (not the data itself) needs to be sent to the signature server. See the Scalability section for more discussion about bottlenecks.

---

7. Even though *key escrow* is perfectly adequate legal terminology for the concept in question, it has attained a bad connotation because of governmental initiatives (especially Clipper), so many circles now prefer such circumlocutions as *commercial key escrow*, *emergency key backup* or *encrypted information recovery* (or various combinations thereof). This paper accepts all such terminologies equally, ignoring questions of political correctness. (Similarly, the author's personal views on key escrow policies are irrelevant to this paper.)

8. The point of this extended discussion (that PK = SK + TTP) is not to say ''SK + TTP is the way to go'', but rather ''if you have to have a TTP anyway, then PK buys you no more security than SK''. I.e., in the words of [AndRoe], ''[W]hy not just use Kerberos? … [I]ntegration of a completely new suite of authentication and encryption software … will mean less secure systems.''

2.5. Key Escrow

If escrowing of private keys is imposed on a PK system, then PK loses its claims to trustlessness, as well as its claims to functionality benefits (signing, strong key-exchange) over SK (since SK + TTP can simulate any PK operation). Note that the requirement of key escrow may be imposed by national governments and the international community.[9]

There are different degrees of key escrow: escrow agents may hold all users' long-term private keys (whether for signing, confidentiality, or other purposes); or it may hold only session keys for confidentiality (see, for example, the escrow system described in [Yaksha]); or something in between (such as escrowing users' session-key-encryption-keys but not their signing keys).

This issue is so fundamental that if key escrow becomes a required component of the GII security infrastructure, it is difficult to see why a full-blown PKI (in the sense of [APKI]) needs to be deployed at all, at least for pure security reasons[10] — unless vendors intend to support different domestic and exportable versions, which seems unlikely (presumably the domestic version wouldn't require key escrow). Namely, if TTPs become required, then it may be more economical to re-use the existing SK infrastructure than to deploy an untested PKI that doesn't give any functionality beyond SK + TTP.

The complexity, and potential for compromise and/or abuse, makes escrow a daunting feature, depending on various design features. For example, a near-realtime policy may require every session key to be immediately escrowed in a safe central location, in a transactional manner, in which case an escrow server would have to be online and would need a large sophisticated database (because session keys, individually or in blocks, are generated continuously). At the other end of the spectrum, the escrow server could be offline and have simple database needs if it only escrowed long-term keys (assuming those long-term keys are themselves generated offline, and the session keys exchanged by the private long-term keys are bound to the data they encrypt).

Various level of key escrow can be imagined, to satisfy differing customer needs. The spectrum includes at least:

(a) Escrow no keys.

(b) Escrow only short-term session keys (some or all, on a per-session basis).

(c) Escrow only certain private keys. For example, those used in a business context but not in a private citizen context; or those used in an international context but not those used in a domestic context (depending on international and national laws).

(d) Escrow all long-term key-exchange private keys (not signing keys).

(e) Escrow all long-term private keys.

Note that users can generally defeat key escrow systems by superencrypting traffic (using non-escrowed encryption, provided it's legal to do so) at application-level and passing that encrypted data to the system. That's harder to do in DCE RPC, because the RPC IDL interface specifies the datatypes that are to be passed to the RPC operations, but it can still be done by using byte-strings in RPC (losing the advantages of strong typing), or by using GSS-API.

2.6. Performance

From a performance viewpoint, PK is slower, in two ways:

(a) On the level of cryptographic algorithms, PK is 100–1000 times slower than SK (for both software and hardware implementations).

---

9. International agreement on a TTP scheme may be forthcoming as early as Spring 1997, from the current round meetings of the Organization for Economic Cooperation and Development (OECD) being held in Paris. However, one prominent proposal, that of Royal Holloway ([Mitch]), has recently come under rather scathing critical scrutiny ([Laurie] and [AndRoe]).

10. Market reasons are different, and are likely to be the ruling ones here. That is, the security argument that ''SK already exists and does everything you want, modulo a TTP'' is unlikely to forestall the market argument that the world wants PK, whether or not TTPs will be imposed.

This problem is ameliorated by using PK to encrypt/decrypt only small amounts of data (hashes, SK keys). But even then, in encryption mode the cost of PK encryption of the key has to be added to the already non-trivial cost of the SK encryption of the data.

(b) Certificate validation takes a relatively long time to do, because of the checking of CRLs (including retrieving certificates and CRLs from network repositories) that has to happen. And this is just for simple identity certificates (those binding a public key to some attribute, typically a stringname). If additional short-lived certificates are being used (‘‘policy-augmented certificates’’, containing authorization information, analogous to DCE privilege service tickets), the problem is exacerbated.

Further PK performance problem are introduced with smartcards (which we are considering a PK(-related) technology for the purposes of this paper).

In general, to improve performance, protocols should be designed to minimize PK operations (consistent with security goals).

2.7. Storage/Archive

PK makes higher demands on storage:

(a) PK keys (i.e., key-pairs) are much longer than SK keys.

(b) Also, since PK services (principally signing) are expected to be very long-lived, certificates must be stored for very long periods of time. In particular if the legal system accepts digital security as the basis for contracts,[11] data and its associated security context (such as certificates) may have to be stored for up to a century (depending on the time-value of the data). This poses unprecedented problems. For example, aspects of the certificate validation procedure (including CRLs) must be archived, with high assurance. Also, hardware and software must be supported which can handle archives that potentially may be over 100 years old.

These two facts actually interact: The longer the time that a PK key-pair must remain secure, the longer the keys must be. (The same is true of SK, of course.)

2.8. Random Number Generation

PK seems to require more random bits than SK. A large supply of random bits is especially required of nonce keys, both PK and SK (PK often uses nonce SKs to protect offline communications).

This can be an issue because generating random numbers of good cryptographic quality (namely, non-predictability even if all previously generated random numbers are known) is non-trivial and time-consuming, especially for simple machines (such client machines).

2.9. Cost

From a cost viewpoint, PK is usually considered to be more expensive, for a number of reasons, such as:

(a) Patent protection of PK technology that doesn’t exist for SK technology. Despite widespread skepticism about the validity or enforceability of these patents, they have historically had a strong chilling effect on the deployment of PK technology, reportedly because of costly licensing issues (or at least because of all the confusion!). Fortunately, even though there are hundreds of patents on PK (and SK, and hashing) technology, some of the most interesting patents expire relatively soon.[12] Note that it is legal (according to patent

---

11. Of course, this raises all sorts of non-technical issues (software disclaimers, due diligence, liability, etc.) which are beyond the scope of this paper.

12. Based on the old Patent Office law of 17 years from date of issuance (the new law is 20 years from date of filing, but is inapplicable to these patents), the expiration dates of the three best-known patents are as follows. #4,200,770 expires on 4/29/97 (Diffie-Hellman, based on discrete logarithms in the underlying multiplicative groups of finite fields; the patent only directly covers key exchange, but is also claimed by its holder to apply to similar algorithms such as El Gamal encryption/decryption, and perhaps also to analogs of discrete logarithm algorithms applied to other groups, such as elliptic curves over finite fields). #4,424,414 expires on 8/19/97 (Hellman-Merkle, based on the knapsack problem; as described in the patent this is now known to be insecure because the private key can be derived from the public key in feasible time, however this patent is also claimed by its holder to apply to the abstract notion of PK cryptography and all its embodiments — a concept contrary to ordinary patent law). #4,405,829 expires on 9/20/00 (RSA, the most popular of all PK algorithms, based on integer factorization, patented in the U.S. but not in the rest of the world).

law) for companies to develop products now using the patented algorithms (which are generally publicly known), provided they do not ship them until the patents actually expire.

(b) It is often coupled with use of hardware smartcards, which are expensive (though some software substitutes do exist). The use of smartcards has certain security benefits, but if the smartcard issuer installed the card's private key, then the unique advantage of PK (''nobody knows the private key except the user'') is negated.

(c) The cost of running the PKI may be greater than the cost of running the SK infrastructure (i.e., the existing DCE/Kerberos-based scheme), because of PK's greater overall complexity. Admittedly this is a weak statement, as it depends on a lot of variables about the management structure of products, but it's pretty clear that the administration of the PKI will not be significantly simpler than the administration of the SKI.

2.10. Deployment (Installed Base); Reuse of Infrastructure

From a deployment viewpoint, PK is viewed as being a long-term solution because the underlying PKI services are not available yet (and won't be for a number of years), while SK has a large and growing installed base (witness Microsoft's stated intention to base its future distributed security on Kerberos). The case is well argued as follows ([Yaksha]):

> [W]e believe that the effort required to get a multi-vendor supported standard authentication system whose security properties have been widely examined is probably the hardest part of implementing a new system. For the most part, this effort has already been exerted on behalf of Kerberos . . .
>
> Having observed the fairly tortuous and time consuming process the Kerberos community has wound through to finally arrive at what is a mature, and from all appearances a fairly secure, standard, we are of the firm conviction that any attempts to improve Kerberos would do so with only minimal impact to the protocol and the source tree.
>
> . . .
>
> While the justification for [the preceding] requirement [that key escrow authorities should have access only to short-term session keys, not long-term user keys] is grounded in a debatable philosophical stance, the next requirement is based on something more concrete — money! Requirement: *It is very desirable that the key escrow system reuse the security infrastructure necessary for other security functions, such as key exchange, digital signatures and authentication.* In theory, it is possible there could be distinct security infrastructures for different security functions [such as: authentication; signing; keys exchange; keys escrow]. The problem with such an arrangement is cost. Each of these separate keys has an associated infrastructure for generating keys, resetting keys when needed, revoking keys, and so on. From a user perspective, it may well be the case that the distinct infrastructures translate into distinct keys to remember or numerous smartcards to carry. Finally, quite apart from cost, multiple systems increase complexity, which significantly affects the ability to maintain the desired security functionality.

2.11. Management

From an administrative viewpoint, PK is sometimes considered easier to manage than SK, because certain administrative tasks (particularly generating identity certificates) are done infrequently. Namely, a single CA is considered able to service many more users than a key distribution center since a user only has to rarely interact (re-register) with a CA. That conclusion may been called into question because the real bottleneck is the labor cost (e.g., face-to-face meeting) of properly assured account initiation, which cannot be short-circuited (i.e., it's really the same as the SK case). Namely, the CA must conduct careful checks before issuing a certificate containing user-supplied information (otherwise, what is prevent a imposter from claiming false information, such as someone else's email address?).

What's worse, PK appears to silently transfer to *users* certain elements of maintenance that may more properly belong with administrators (see [Davis]).

In any case, the bulk of account management is involved with ongoing maintenance of things other than simple identity certification (such as groups, roles, and other privilege attributes).

2.12. Scalability

From a scalability viewpoint, PK is sometimes considered superior to SK, because SK involves a centralized server that may become a processing bottleneck: the key distribution service (KDS) must be visited before a client's initial contact with a given server. This is considered to be an avoidable bottleneck, because in PK there is no KDS in the loop.

Even ignoring the fact that PK certificates are bigger than SK tickets, this argument assumes that authentication traffic is a significant part of total network traffic, but that is not the experience with SK. For example, the KDS may be replicated.[13]

But more importantly, the argument fails to take the PK certificate validation problem, especially revocation, into consideration. Thus, a more realistic view of PK client/server authentication shows that it uses even more client/server processing power, contacts more network services, and uses more bandwidth than SK:

(a) Client fetches server's certificate (from some certificate repository, e.g., from a directory service).

(b) Client validates server's certificate. This involves checking CRL(s) (retrieved from some CRL server).

(c) Preceding two steps are done recursively (checking certificates of CAs), until root CA is reached.

(d) Server fetches client's certificate from repository. (Alternatively, client sends its certificate to the server, using bandwidth.)

(e) Server validates client's certificate (which again involves checking CRL(s)).

(f) Preceding two steps are done recursively by the server, just as the client had to do.

If the certificate and CRL servers involved in PK authentication can handle the traffic, then so can KDS servers.

Both SK and PK may use either a *hierarchical* arrangement of their trust trees (normally tied to naming trees), or a ''web-of-trust'' arrangement of bilateral agreements (as in PGP or [SDSI]), or a hybrid (as in DCE already, which allows for bilateral agreements within the trust hierarchy), so this is not a distinguishing scaling factor. What is an issue for DCE, however, is whether or not the PK trust hierarchy is the *same* as the DCE SK trust hierarchy. If not, this kind of complexity could lead to administrative headaches, and even security flaws. For example, there could be multiple different trust paths between principals, with potentially differing levels of trust (for example, some trust paths through the combined DCE SK and PK CA hierarchies might be able to authenticate a client to a server, but some might not).

(Another argument raised in favor of PK's scalability is that of management scalability, namely, fewer management operations are required to support the same population of users. This is discussed in the Management section.)

2.13. Root Key

From an assurance viewpoint, the difficulty of out-of-band certification of the PK *root* (or *master*) key(s) has been called into question. Namely, all keys except root key(s) are packaged in, and protected by, certificates, but root keys cannot be so protected (''self-certification'' doesn't work — the certifier cannot certify itself), so they must be protected in some other way (e.g., cached on disk perhaps protected by a password, or kept in a smartcard). In such an uncertifiable state, root key(s) are susceptible to compromise, and must be certified out-of-band of the normal certification scheme (i.e., validating signatures on certificates, and checking CRLs). Of particular interest is what happens when a root key is compromised (e.g., who issues the CRL, how is it validated, and how is the PK world rebooted, including reissuing new smartcards that stored the root key). This represents serious security administrative burdens, especially on end-users if they must maintain constant vigilance over their root keys. To quote [AndRoe], ''In our experience, the likelihood of master key compromise is persistently underestimated.''.

---

13. Replication always introduces security weaknesses, but with the DCE 1.2 PK enhancements the Registry need store only very few secrets (cross-cell surrogate keys and its own secret key), and the Keystore server does not store extremely sensitive data (it stores private keys that are password-protected).

2.14. Revocation

Revocation is much easier with SK than with PK. In the words of [Davis], ‘‘Revocation is the classic Achilles' Heel of public-key cryptography’’. If PK were only being used for simple (human-to-human) applications such as email, revocation wouldn't be much of a problem (and could be solved by simple web-of-trust solutions such as PGP or [SDSI]). But if PK is to become the infrastructural security underpinning for all (program-to-program) applications, it is a major problem.[14]

In particular, the terrific scaling difficulty of timely and secure PK revocation is a major issue. Every time a public-key is used it must be validated by verifying its certificate, and particularly checking CRLs; which in turn involves validating the public-keys of the issuers of the certificate and CRL themselves; which in turn again involves validating *those* public-keys; etc. These steps could be completely or partially skipped (e.g., by caching in memory or on disk, or by limiting the depth of the CRL checking depending on ‘‘quality-of-security’’ parameters), but that would compromise security (for example, because of the potential compromise of cached data such as CRLs, or because a certificate may have become revoked during the period of caching). The [SDSI] paper advocates using reconfirmation periods instead of CRLs, but unless the reconfirmation period is exceedingly short (on the order of hours, like Kerberos tickets, which requires online services) that cannot be considered sufficiently secure without some sort of supplementary high-priority revocation capability (which brings you back to CRLs again).

2.15. Key Management

From a key management viewpoint, long-term PK keys are used more often (for signing and key-exchange) than long-term SK keys (which are essentially only used at login time, assuming that servers use the user-to-user protocol). That presents a key management problem. It's particularly a problem if these keys appear in memory or on disk, where they can potentially be corrupted or stolen. Smartcards are one way to address this problem but have their own issues (such as performance and cost), as discussed elsewhere.

2.16. Passwords

From the viewpoint of quality of passwords (a.k.a. passphrases, passcodes or personal identification numbers (PINs)), SK systems with a TTP can vet the password for strength. In PK systems without a TTP, this can't happen without compromising the password (which protects the private key). This is even true in the case of smartcards: if an authority vets the smartcard's PIN, then that authority could have escrowed the PIN so it could be used illicitly later.

On the other hand, PK can be used for perfect forward security, i.e., secure establishment of a new password even if the old password has been compromised.

2.17. Availability (Online Vs. Offline)

To many people, a pure PK environment implies ‘‘no dependence on online trusted servers’’. I.e., the only trusted server is an offline CA. Offlineness is considered desirable because:

(a) It implies that high-availability (including replication) need not be engineered into the security infrastructure. That's good, because high-availability is difficult and expensive, and can be a security problem if datastores holding secrets need to be replicated, because every such replica is a weak link in the security chain. (Note that DCE 1.2 does not need to store secrets if PK login and smartcards or the Keystore server are used. And with the user-to-user protocol, application servers do not need to store their long-term keys either.)

---

14. This, in fact, seems to be the one of the core themes pervading the challenge of integrating PK with SK and DCE. Namely, PK is still in its formative stages, and the foundational work on PK is still forthcoming from the academic world. Naturally, the straightforward realizations of this academic work apply best to simple human-to-human applications (e.g., ‘‘mental poker’’). Hence the attempt to make PK the basic underlying security paradigm for a different realm, namely high-performance, general purpose computing environments (such as SK and DCE are designed for), is highly nontrivial.

(b) It enhances security because CAs are easier to protect when they are offline. This gives them logical security, in addition to physical security — something that can't be attacked can't be compromised.

As far as general objections to high-availability are concerned, it would seem that the coming global information infrastructure (GII) is going to have to support high-availability of its basic services anyway (both security-related and otherwise, including end-applications), otherwise large-scale economic activity, which is the target use of the GII, will not be able to depend on it. Examples of highly-available electronic infrastructures already exist in such areas as the telephone and broadcast entertainment industries (including the cable industry, and analog and digital cellular services industries), and there is no reason to doubt that high-availability will have be a characteristic of the GII as well.

As far as security is concerned, it is indeed desirable that no stockpile of secrets (such as a classical Kerberos datastore, or a key escrow datastore) exit, because that makes a tempting target whose compromise is catastrophic to the network. However, that does not necessarily imply offlineness, as DCE 1.2 (which employs trusted online servers but does not require secret storage) already demonstrates.

The online vs. offline debate is touched on in context throughout this document, and will not be repeated in detail here, but here is a list of some of the more important infrastructural services that would seem to be needed online, or else a good offline substitute designed:

(a) Revocation.

An available authentication server (with short-lived tickets, like Kerberos) or CRL authority (or their agents, with constantly updated CRLs) is necessary for ''bounded'' (guaranteed timely) revocation.

(b) Dynamic administration of security accounts.

New users, groups, attributes (e.g., signed PACs), etc., can be administered in realtime (a very desirable service that users are likely to demand) only with a highly-available service. Examples of uses that require knowledge of which attributes a user is entitled to are *least privilege* (attribute-narrowing) and *delegation* of partial rights (as opposed to full-right *impersonation*).

(c) Per-policy-domain privilege attribute vetting and mapping.

Policy may demand highly-available per-domain attribute manipulation, via a policy information database.

(d) Key escrow.

Depending on key escrow policy, the escrow agent might have to be online (e.g., if only short-term session keys are escrowed, and the escrowing must be done at key generation time with positive acknowledgement from the escrow server).

(e) CA Agent. Even if the CA is offline, its agent must be online, for batching jobs to and from the CA.

(f) Certificate repository (e.g., directory service, database or email service) must be online.

(g) Time service. Must be online, almost by definition.

(h) Audit service. Obviously, any audit service must be online, because the events it audits happen in real time.

(i) Notary service. Must be online, for the same reason that audit service must be online.

(j) Probably some other infrastructural services not thought of here need to be online — not to mention all the *application* services that must be online before any real work can get done . . .

Unless good arguments can be given to the contrary, it is plausible to conclude that whatever combination of PK and SK is used, many highly-available (''online''), trusted, secure services will be required. Some of these may be designated ''more trusted'' than others (such as the CA), but as usual with security, there is no reasonable place to stop.

3. SPECIFIC ISSUES RELATED TO PK, SK, AND DCE

This section discusses issues involving PK, SK and their hybridization, as they specifically relate to DCE.

3.1. Review of Security Features Supported by DCE

For the convenience of readers whose familiarity with DCE is shaky, it may be worth giving buzzword-level lists of the security features supported by the various existing DCE releases. More to the point, these lists also act as a reminder of the features that must continue to be supported gracefully into the indefinite future as we add PK to DCE.

As always, Release Notes should be checked for limitations on advertised functionality. (For example, the Release Notes for DCE 1.1 state that delegation chains were limited to the initiator's cell, and that the hierarchical cells feature were to be supplied in the Warranty Patch; and in DCE 1.2, third-party global groups couldn't really be used in ACLs, only global groups that exist in the client's or server's cell.)

3.1.1. DCE 1.0

(a) Crypto primitives: DES, MD4, MD5. (CRC32 is also used in RPC, though that isn't cryptographically strong.)

(b) Registration service: Registry server (datastore of account information). Replicated (together with KDS and PS). Registry editor.

(c) Kerberos authentication: KDS server, consisting of AS and TGS services, including direct cell-to-cell cross-registration.

(d) Privilege service: PS server, issues PACs to clients and does cross-cell (policy-domain) privilege attribute mapping.

(e) Authorization: ACLs, ACL managers, access algorithm. ACL editor.

(f) Protected RPC: Six levels, especially session integrity and confidentiality.

(g) ID mapping facility.

(h) Password import/export, password override.

(i) Server key management facility.

(j) Login facility, including security client daemon.

3.1.2. DCE 1.1

(a) Hierarchical cells (transitive trust). Security administrators need to configure the *names* of the foreign cells they want to permit communication with, but secure exchange of *keys* is required only for those in the trust hierarchy (plus peer links between disjoint hierarchies).

(b) Dynamic schema in Registry: ERAs.

(c) Authorization enhancements: Delegation, restrictions, EPACs, extended access algorithm.

(d) Pre-authentication: Thwarts offline eavesdropper dictionary attack.

(e) Password management: Strength checking, reuse, expiration, generation; customizable.

(f) Login denial: Thwarts online (multiple login attempt) dictionary attack.

(g) Extended login ERAs: Login set, policy check, attach environment, environment set.

(h) Audit.

(i) ACL manager library, including datastore library.

(j) Unified management: `dcecp` and `dced`.

(k) GSS-API and Extended GSS-API.

(l) Group override.

3.1.3. DCE 1.2

A few more words will be said about the features in the following list, since DCE 1.2 is still in pre-release stage.

(a) Kerberos support: Guaranteed (i.e., actively tested) support for non-DCE implementations of Krb5.

(b) User-to-user protocol: Kerberos service tickets encrypted in server/KDS session key (so servers no longer need to keep their long-term secret stored in keytab file).

(c) Global groups: Allow principals in one cell to belong to groups in other cells.

(d) Smartcard support: System internal personal security module (PSM) layer abstracts smartcard functionality. (Implementable under PAM SPI, under consideration as OpenGroup standard.)

(e) Private key storage server (PKSS, or Keystore Server): Stores users' private keys (password-protected; PKSS does not know password) as replacement or migration aid for hardware smartcard or token card. Uses Diffie-Hellman key-exchange to protect the password-protected private key against eavesdropper dictionary attack.

(f) PK login: Enables users to login using a private key they possess (stored either in local file or PKSS, with hooks for smartcard), using X.509-based mutual authentication protocol (but not actual X.509 certificates). This ''takes (most) long-term keys out of the registry'', i.e., users no longer need to share a secret with the KDS, greatly easing recovery in the event of KDS compromise. (In the absence of a PKI, the registry stores users' public keys, and `dced` stores the KDS's public (''root'') key.)

(g) Certificate management API: However, this is not integrated with the other DCE 1.2 features, no specific certificate formats are supported, and no CA hierarchy is assumed.[15]

Altogether, the DCE 1.1 and 1.2 enhancements effectively eliminate nearly all complaints about DCE/Kerberos password-based authentication weaknesses (though cross-cell surrogate secret keys still remain in the registry). See [BelMer].

Note that only PK login (PK authentication of user to KDS) is supported in DCE 1.2 — not full end-to-end PK authentication of clients to arbitrary application servers. Once the user is logged in, all authentications are via ordinary secret-key based DCE privilege tickets. Servers can also use the same PK login to get their TGTs, then use the user-to-user protocol thereafter. (Incidentally, in the PK login code, the user's private key is not delivered to client application level, it is destroyed by the login code after it receives the TGT. So, today, this private key couldn't be used for full end-to-end PK authentication with servers, even if the client knew how to do that.)

3.2. Full End-to-End PK Between Clients and Servers

In order for clients to authenticate themselves to servers in a PK environment (unilateral authentication, which may be appropriate for some limited environments), the client needs access to its own private key, and the server needs access to the client's public key (certificate). This means that a lot of private-key manipulation machinery must be built into clients (perhaps via smartcard), and a lot of certificate-manipulation machinery (including CRL checking) must be built into servers.

In the reverse direction (mutual authentication, which is appropriate for most environments), in order for servers to authenticate themselves to clients (including privacy-protection), the reverse sets of machinery are needed, since the client needs access to the server's public-key (certificate), and the server needs access to its own private key.

Thus, full end-to-end PK between clients and servers requires quite a lot of trusted code in the clients and servers. This may not be much a burden for servers (if those are large, well-protected and well-administered machines), but it could be a considerable security burden for client machines. See [Davis] for more on this point of view.

---

15. But the certification API could be used to construct an ERA trigger-server that would retrieve the DCE PK authentication and DCE PK key encipherment public keys from a PK repository if one were available. Thus while users still need accounts in the DCE Registry, the actual values of their public keys could be tied into some CA-maintained repository.

One SK/PK hybrid, that supported by DCE 1.2, is to use PK for login, then thereafter use SK between clients and servers. This has a number of advantages, as discussed throughout this document, and it does not preclude future incorporation of full end-to-end PK between clients and servers, to support those policies that require it. In [NTW] the designers state that they believe this achieves ''. . . 80 percent of the benefits of integrating public key with Kerberos''.

One major issue that needs to be dealt with in full end-to-end PK is non-identity-based access controls (authorization), such as access by *groups*, or by *roles*, or even *anonymous* access. In situations such as these, where no signature is necessary or desired, some scheme for binding attributes to principals must be agreed upon. It's easy to do that with a TTP, as DCE does. But if, as stated above, policy demands ''full end-to-end PK only'', and no TTP is allowed, life becomes harder. This is discussed in more detail below.

3.3. Multi-Crypto

In terms of the APKI, DCE protected RPC appears as a *connection-oriented peer-to-peer secure protocol*. In this context, ''connection-oriented'' means that security state is maintained by the client and server, so it need not be transmitted with the communicated data. (In particular the security state is not available to third parties, so it cannot by itself be used to provide non-repudiable services.) In DCE RPC, the manner in which multiplicity of security service mechanisms are currently supported is at API level, primarily by the routines `rpc_binding_set_auth_info()` (for clients) `rpc_binding_inq_auth_caller()` (for servers). There are three dimensions of mechanisms supported by parameters to these calls:

(a) Authentication service. This refers to end-to-end client/server authentication (not the DCE 1.2-style login authentication). In a full end-to-end PK environment, new values of this parameter must be supported, corresponding to the PK authentication method used. If the PKI supports more than one such method, then more than one value of this parameter must be supported.

(b) On-the-wire protection level (integrity, confidentiality) between client and server. Currently this parameter indicates DES-based protection only. New values of this parameter corresponding to other algorithms must be supported in a PK environment.

(c) Authorization service. Name-based and PAC-based authorization are currently supported. Those should both continue to be supported. Other authorization services (e.g., anonymity) do not appear to be well-formed enough to recommend adding them yet.

One thing that would make sense as a future enhancement would be to expand the number of dimensions here. For example, different parameters for SK algorithm and for key length, or different parameters for PK algorithm and hash algorithm for signatures. That would add complexity, so some sort of profile utility would also probably have to be supported, to bundle together sets of frequently used options.

A suggested way to actually implement multiple mechanism negotiation in DCE RPC is that the server should export the security mechanisms it supports to the CDS namespace[16] (just as today it exports the RPC bindings it supports, with `rpc_ns_binding_export()`). The client would then import this information (similar to `rpc_ns_binding_import_*()`). Depending on design decisions (such as degree of transparency to application programs, and degree of integration with IDL/ACF and NIS), this may or may not require revving RPC protocol version numbers. (For an analogous situation, in the arena of internationalization instead of security but which may be applicable here, see [RFC 41].)

By the way, multiple PK key-pairs per principal should be supported. Those will be needed, for example, for signature keys vs. key-exchange keys, and for business vs. citizen-of-U.S. vs. citizen-of-world (where there may be exportability issues on the type of key-pair that can be used), etc.

Along these lines, *key-vectoring* (i.e., key-usage controls) will probably have to be supported (e.g., via the X.509v3 key-usage extension field) — e.g., signature key-pairs shouldn't be used for key-exchange (because, for example, that might bypass key escrow). However, that's hard to do, and as usual patents may apply.

16. Or LDAP, or whatever — for most security purposes the directory service doesn't really matter very much.

Exportability is going to be a major issue before serious work on multi-crypto confidentiality (including both algorithms and key-exchange) can be committed.[17] Preliminary discussions with the NSA has revealed they view mere multi-crypto *capability* (even in the absence of actual cryptographic *algorithms*!) as an ITAR-controlled munition under the vague catchall category of ''ancillary equipment'' (in the sense of [ITAR: 121.1 XIIIb5]), especially for a source-code product like DCE. On the other hand, a number of legal proceedings are progressing against the ITAR export controls (especially the Bernstein case, where a judge has ruled that software is ''speech'' protected by the First Amendment). A recent turn of events is the Clinton administration's decision to allow strong multi-crypto confidentiality provided certain (unspecified) requirements regarding key escrow are met, and industry reaction to that announcement (including the formation of a Key Recovery Alliance). The situation is very unstable.

3.4. Smartcards

Smartcards are tamper-resistant[18] hardware tokens, invariably associated with PK technology (even though they work equally well with SK), which run two kinds of software: a simple operating system and cryptographic functionality. For our purposes here, we'll restrict attention to fairly full-featured smartcards: they have a persistent datastore (file system) that holds at least the user's certificate(s) (or a hash thereof) and associated private key(s); and they have the capability to perform all the usual cryptographic operations such as generating random numbers and keys, hashing, encryption and decryption, etc. Access to smartcard functionality is protected by a password. I/O to the smartcard is commonly done via a smartcard reader attached to a computer, though some smartcards have onboard keypad and readout.

The key security idea of smartcards is to put all the compromisable security-critical entities into hardware, out of harm's way (software is considered too vulnerable to attack). The security-critical entities in question are things like PK private keys and their associated certificates (or at least their hashes), generated SK keys, cryptographic algorithms including hashing, and various operations such as random number generation (for key generation). Since the card is tamper-resistant, the only way to compromise the card to have physical possession of the card, and to know the password (the card disables itself a few consecutive password attempts, so dictionary attacks cannot succeed). This provides a high degree of assurance against compromise, and it gives the user a tangible feeling of security (the user notices if the smartcard is missing).

While smartcards have some indisputable advantages, they also have a number of potential disadvantages:

(a) Despite the desire to isolate compromisable entities in hardware, some part of the trust will have to exist in software anyway. Of course, whether or not this is a significant problem depends an a number of factors. For example, some environments might consider short-term session SK keys (such as those distributed by Kerberos) too sensitive to leave in software, and prefer to keep them ''on the smartcard'' (actually, wrapped in long-term public keys, which themselves never leave the smartcard). But the same environments are happy to use cached CRLs (without revalidation), stored in software. This is inconsistent.

(b) Cost. Smartcards are more expensive than software, in almost every dimension: such as design, manufacture, maintenance, deployment, upgrade (to new algorithms as old algorithms become obsolete), compromise-recovery (e.g.,. of a root certificate stored on millions of cards), etc. So environments that employ smartcards can only justify doing so by proving that their security advantages outweigh their added cost. On the other hand, some manufacturers are starting to integrate cheap smartcard readers into keyboards, and in any case hardware costs for both the card and the card-reader will come down over time — perhaps the incremental cost of smartcards over software will even become negligible.

(c) If a smartcard is initialized by an authority other than the card's owner, then that authority must be trusted with knowledge of the private key(s) on the card, i.e., it is a TTP. This negates PK's primary advantage (of

17. Note that DCE 1.2 supports PK for *authentication* only, so it didn't have to face major exportability problems beyond those already faced by DCE 1.0 and 1.1. Even for authentication, switchable algorithms will soon be necessary (because of the may PK authentication technologies that will be need to be supported in the future), though doesn't fall under the category of multi-crypto confidentiality. Full end-to-end PK does fall under the category of multi-crypto confidentiality, though.

18. It is more prudent to speak of ''tamper-resistant'' devices than to speak of ''tamper-proof'' devices. This has been recently reinforced with fault-induced attacks, which probe security tokens via faults induced by the attacker.

the private keys being known only by their owners, not by a TTP). Hence, a smartcard scheme that purports to be more secure than software solutions must include a way for card owners to initialize cards using machines and software the card owner trusts (software solutions, such as the DCE 1.2 Keystore server, can do this easily if policy allows it).[19]

(d) If a computer is trusted, then software solutions can easily provide all the security guarantees of a smartcard, but an untrusted computer can easily defeat software solutions (namely, by capturing passwords and/or keys, and masquerading as the user). Thus, smartcards only make security sense if the computer (including its card reader) the smartcard is used with is untrusted. On the other hand, if the user of the smartcard is not communicating directly with the smartcard but is using the computer to do so, then an untrusted computer can simply lie to the user about what the card is doing, and it can use the card for its own illicit purposes (at least while the card is plugged in). Hence, in the worst case, a smartcard that doesn't have a trusted path to it (e.g., its own onboard keypad and readout) doesn't make sense. (Again, this drives up the cost of the card.)

(e) Performance with smartcards is poor. Performance is dependent on at least these aspects of the card:

(i) The performance of the hardware itself is typically not a critical issue (because special-purpose cryptographic chips optimized for smartcard operations are used). However, the small memory space of smartcards can make some calculations, such as key generation, very slow.

(ii) The major bottleneck with smartcards come at the thin I/O transmission line between the card and card-reader. This bottleneck has major performance impact in two ways:

[a] The amount of data that has to be manipulated on-card (as opposed to off-card) makes a big difference, because that data has to be read on and off of the card to do the cryptographic operations. For example, producing an encrypted hash (for signature purposes) can be done in two ways: either by doing both the hashing and the signing on-card, or by doing the hashing off-card and then the signing on-card (note that hashing does not require access to the card's private key, only the signing operation does). The former is slower, of course, but is also more secure.

[b] The amount of data stored which must be uploaded to the host can also make a big difference. E.g., just storing hashes of certificates on-card, instead of the certificates themselves, improves local performance, but that gain may be offset by the concomitant necessity for off-card retrieval of the certificates (probably from a remote certificate repository). Recalling that the card may be required to support multiple certificates (minimally, the user's certificate and the CA's certificate), and that PK certificates are fairly large (because public keys are fairly big), makes is clear why this could be a problem.

(iii) If the smartcard supports only one cryptographic context at a time, then the attendant swapping of contexts will cause performance to suffer even more.

(f) Since every smartcard must be continuously available whenever the principal it represents is active and using the card's cryptographic services,[20] machines that support multiple simultaneously active principals must be equipped with a rack of smartcard readers (otherwise the user(s) would have to continually swap the smartcards in and out of the reader, reinitializing them each time). This is particularly the case with server machines which support multiple DCE servers. In particular, such server machines have to be physically secured (because you shouldn't leave a live smartcard unattended), just as in the SK case. (Actually,

---

19. The standard process for distributing smart cards currently does not fit the model of card owners initializing their own cards. A batch of smart cards is shipped to a purchasing organization. A master card for the batch is shipped by a separate channel. The PIN for the master card is shipped by a third channel. Initializing the smart cards in the batch requires possession of the master card and its PIN. For some cards, users can change their keys and/or PINs once their card has been initialized, but this still means the user can be impersonated (at least for a time) by the TTP responsible for initialization (and many users will neglect to change their keys and/or PINs anyway).

20. It is possible to bind a session to electrical connectivity of the smartcard, even if the card's services are not being actively used. This may have value, but it not currently common. Such binding requires trusting the user to do the disconnect.

the same holds for client machines.) It also implies that at boot time the administrator(s) will have to be physically present to type in the passwords for those smartcards. Thus, automatic reboot (i.e., without an administrator physically present) after a power-hit cannot be implemented.[21] This drives up the expense and administrative complexity of running a PK environment.

Note that a DCE 1.2 server (which logs in and has a login context) can be Keystore-based (as opposed to smartcard-based), provided it uses with its client the DCE 1.2 user-to-user protocol (because, the Keystore-based server, like Keystore-based users, retains in memory only the session key carried in its TGT, not the private component of its PK key-pair).

(g) Further, in DCE, all hosts (a.k.a. ''machines'') are themselves principals. Therefore, in a pure DCE/PK environment using smartcards, every machine must therefore have its own smartcard — *in addition* to the smartcards of the other principals (humans and/or servers) that use the machine.[22]

In the case of a server machine, the additional cost and complexity of a smartcard reader for the machine principal itself may be bearable because it is just a small incremental cost over the cost of the machine itself and the rack of smartcard readers already needed by the DCE servers on the machine.[23] But for a desktop client, the requirement of having two smartcard readers instead of just one may not be bearable (both cost-wise and as an administrative headache, even if the machine principal's smartcard is needed only during initial boot or initial start-up of the machine's daemons).

This raises the question: Is it really necessary for a desktop client to be a DCE principal? This topic is covered in the following section.

(h) What happens when the information on a smartcard becomes invalid, or even worse, when the tamper-resistance assumption of smartcards becomes invalid (witness the recent fault attack on smartcards)? The disaster recovery scenario is unpleasant. It's not too bad when a single smartcard is compromised, such as when its PIN is lost. But suppose a root key stored on the card is compromised, or the algorithms implemented by the card become insecure. Then the whole population of users is at risk, until new cards can be programmed, manufactured, and reissued. This is very hard to do. It's much easier to do with software.

The above problems may be ameliorated by replacing hardware smartcards with software substitutes, such as DCE 1.2's Keystore server. However, this substitution may not be available to all environments, such as those that *mandate* smartcards (such as certain national governments). Some other environments may be partially able to substitute software for hardware. For example, some environments may require a hardware smartcard only for those principals that demand confidentiality services — in that case, principals that do not demand confidentiality (i.e., demand only authentication, authorization, integrity, signing, non-repudiation, etc.) can use a software solution.

3.5. Machine Principals

In DCE, every host (a.k.a., ''machine'') has a principal identity. For various reasons (such as the smartcard question on client machines, in the preceding section), it is a legitimate question to ask what we would be giving up if some machines (in particular, client machines) didn't have principal identities. The following are some of the normal uses of DCE machine principals (with special attention to client machines):

(a) Certification of login context. The DCE login facility supports the ability for a program (in particular, the operating system's login program) to ''certify'' a user's login context, i.e., ensure that the login context is an authentic one in the sense of authentically representing a real account in the DCE Registry. This involves

---

21. Unless you leave the smartcard's PIN in a file so a boot-time daemon can read it, or some such hack subverting security. But even that's not possible for some smartcards (e.g., those with their own keypad and readout).

22. As usual, there's a security advantage to using smartcards for the machine principal. The current standard configuration for DCE 1.2 is that machine principals are SK-based. DCE 1.2 PK login currently assumes that the machine principal has been able to authenticate itself successfully, and it is then responsible for obtaining a trusted copy of the KDC's public key. If the KDC's public key is stored on the smartcard, then this assumption could be relaxed.

23. The rack of smartcard readers may be unavoidable in environments whose policy requires a smartcard per principal. Some environments may permit multiple sub-identities to be stored on a single card under a single master identity, so that would lessen the rack of card-readers problem. Of course, environments that simply assume a trusted server machine could avoid the smartcards on the server altogether.

guarding against a ‘‘multi-prong attack’’ (where an attacker simultaneously masquerades as a logging-in user and the Kerberos server, and might even attempt to masquerade as the user’s machine principal). The DCE machine principal is used to thwart this attack, to the extent made possible by local operating system security. Without this ability to certify login context, machines and their local operating systems (and other local per-machine applications that don’t have network identities of their own) will be uncertain of users’ network identities, and cannot make local access decisions on the basis of users’ network identities. They may have to implement local login programs of their own, not relying on users’ network identities — contrary to the goal of single-signon.

(b) Pre-authentication, for login of secret-key based users. If the machine doesn’t support login of secret-key based users (e.g., if the only users of the machine were smartcard-based), this functionality wouldn’t be needed.

(c) Getting users’ private-keys from the DCE Keystore Server, for login of Keystore-based users (the DCE 1.2 protocol for this uses a Diffie-Hellman exponential key exchange which is authenticated and integrity-protected using the strong secret key of the machine). Again, if the machine doesn’t support login of Keystore-based users, this functionality wouldn’t be needed.

(d) DTS time client (to authenticate the DTS server to the client machine, to guarantee the time is authentic). Trusted time appears to be necessary in any real-world distributed computing system. If trusted timestamps were never relied upon, it might be legitimate to dispense with secure time services. While this may be possible in protocol design (by replacing protocol uses of timestamps with challenges/responses — for example, the Neuman-Stubblebine protocol [Schneier]), it is undesirable (and perhaps impossible) to remove all security dependencies on time. Two security services which appear to have a hard dependency on trusted timestamps are notarization services (which by definition binds a timestamp to a document[24]) and audit service (where timestamps are required to trace event scenarios). Finally, removing time services from the security architecture would remove a basic tool from the application programmer’s arsenal. This seems like a severe restriction.

All-in-all, it appears that secure time is extremely convenient, if not an actual necessity. In a PK environment, for example, secure time is necessary for validating certificate lifetimes and CRLs. The granularity and/or accuracy could be relaxed for long-lived certificates, but in the case of short-lived certificates (such as ‘‘X.509 PACs’’) the issues are similar to DCE. (By the way, for the purpose of disseminating secure time, DTS servers require signature capability but clients require only validation capability, which can be done in software.)

(e) Audit.

(f) Delegation-through-machine (‘‘attach-environment’’). Some access control schemes are ‘‘location’’-based, in the sense of only allowing access if the accessing principal is using a specified machine. In DCE, this involves constructing a delegation chain that includes the client’s machine, and this can’t be accomplished unless the machine has a principal identity (the machine-to-identity binding must be stronger than IP address, for security reasons).

(g) `dced`, for host management.

3.6. Cells

There are several ways that a notion of ‘‘cells’’ (or ‘‘realms’’, or ‘‘domains’’, etc.) are useful, even in a PK environment:

(a) As a basis for an authentication hierarchy (CA hierarchy in the case of PK).

24. Signature schemes that don’t employ trusted notaries are vulnerable to forgery by back-dating messages if private keys are lost or stolen. For this reason, all certificates carry validity timestamps, which are subject to being overridden by CRLs.

(b) As a unit of management/administration.

(c) As a unit of authorization (‘‘foreign’’ groups and principals on ACLs).

(d) As a policy domain. (In this sense, DCE’s privilege server acts as a per-cell policy server/mapper.)

3.7. Authorization: PACs, ACLs

Authorization in DCE is based on privilege attribute certificates (PACs) and access control lists (ACLs). In the current SK environment, the Registry Server (RS) acts as an online trusted repository of principal attributes, and the Privilege Server (PS) is an online trusted server that issues Kerberos(-style) tickets, called privilege tickets (PTkts), containing or bound to PACs.

The TTP/online nature of the RS and PS are accepted features of the DCE authorization structure. In a hybrid SK/PK environment this scheme (of an online PS, which issues PACs to principals to use for authorization) could well continue to work essentially unchanged (as in DCE 1.2). But in a PK environment that is required to be ‘‘pure’’, PTkts would have to be replaced by ‘‘privilege certificates’’ (which may be thought of as ‘‘X.509 PACs’’) — that is, certificates signed by some trusted ‘‘privilege authority’’ (PA), binding principals to their attributes. Depending on the usage model and services provided, such a PA may or may not have to be online (an offline PA might not adequately service a dynamically changing environment, such as one that supported least privilege).

Sealing PACs inside (or otherwise permanently attached to) a long-term certificate (or inside a smartcard) may be unacceptable, because of the relatively dynamic nature of privilege attributes. But PACs could be attached to short-lived certificates (just as they are attached to short-lived SK tickets).[25]

Another use of the DCE PS is to vet PACs in cross-cell operations (policy mapping, as mentioned above).

3.8. Authorization: Groups

Groups denote sets of principals that are created (by group owners or administrators) for use in authorization decisions (group entries appear in ACLs). Therefore, group definitions must be secure, in the sense of being trusted by the servers that use them for access control purposes. In DCE, group definitions are stored in the Registry, and issued as part of client credentials (in PACs or EPACs, bound to the client’s ticket).

In a PK environment, group membership could be specified in a certificate (of the form ‘‘principal P belongs to group G’’; in some schemes, there is even a key-pair associated to a group, though these schemes are not very popular). The authority signing such a certificate must be trusted by the server using the certificate for authorization purposes.

Furthermore, since group membership can change dynamically, the group server probably needs to be online (this is yet one more argument that trusted online servers are needed). Since client frequently want to apply least privilege (i.e., restricting their access credentials to the minimum necessary to get a given job done), clients need some way to remove privileges from PACs. Today that is done by requesting the PS to issue a restricted PAC, though in a PK environment it could be done by having the client sign a certificate (of the form ‘‘only use my ADMIN identity/role’’). This is less clean, but it works.

3.9. Authorization: Delegation

‘‘Simple’’ delegation, in the sense of ‘‘signature authority’’, is easy with PK: just have the user sign a statement that delegates the user’s signature authority to a delegatee. What’s harder is the complex delegation of access rights, as is done in DCE. There is a lot a machinery involved using DCE’s current SK architecture that would have to be re-architected to use PK (specification of delegate- and target-restrictions, required- and optional-restrictions, chains of delegates, access decision function).

---

25. Detailed designs are still immature. One natural scheme (in the X.509 case) is to bind the PAC certificate (which carries no key, and is signed by the PA) to the identity certificate (which carries a key, and is signed by the CA), via the identity certificate’s serial number (trusted to be unique).

3.10. Audit

Full end-to-end PK does not allow centralized monitoring of user actions. With SK (especially the DCE 1.1 pre-authentication feature), it is possible to determine when a user ''logs in'', and to audit that action at the authentication server (AS sub-service of the KDS). With SK it is also possible to audit the service tickets granted by the TGS, and thus to determine exactly which services a user may had access to during a certain period of time. These things can't be done centrally if there is no central authentication or privilege server that has to be visited.

3.11. Caching

There is a tradeoff between caching and security. Namely, if cached security information becomes stale (e.g., a certificate is revoked), or the cache itself is compromised (either by unauthorized reading or writing of the cache), then security is compromised. On the other hand, if caches are not used, performance will suffer greatly. Persistent (multi-session) caches, or shared caches, exacerbate the problem.

For example, SK sessions are typically authenticated only at session setup time, and a session key is established which is then cached and used for the remainder of the session. It would be a great burden on applications if they could not use session keys, and instead had to negotiate a new nonce key for each message.

Similarly for caching certificates, or CRLs, or any other security-relevant data.

3.12. Integration with RPC

All new PK functionalities, such as signing, should be integrated with RPC, i.e., some programmer-friendly interface to them should be made available (more friendly, that is, than a low-level cryptographic API interface). Beyond that, most of the work to be done in RPC is already covered under the Multi-Crypto section, above.

3.13. ID Mapping

The current ID mapping facility maps between human-oriented identifiers (stringnames) and authorization-oriented identifiers (UUIDs). In a PK environment, it would be useful to add another dimension to the identity map, namely security-oriented identifiers (public keys). Note that public keys are the one indispensable piece of information contained in (identity) certificates, i.e., the base object to which other security attributes (such as stringnames, UUID identifiers, groups, etc.) are attributed.

(Another aspect of ID mapping, namely the mapping of non-DCE PK principals to DCE principals, is mentioned elsewhere in this paper.)

3.14. Retrofit DCE Core to PK

Currently, the DCE core services (such as RS, PS, CDS, DTS, PKSS, etc.) only support SK. They must continue to do so, for compatibility reasons. However, some organizations might have all-PK policies. For those, it may be necessary to outfit the DCE core services to also support PK (via PKSS or smartcards). The DCE client will also need certificate-manipulation machinery, including CRL checking.

4. STRAWMAN PROPOSAL FOR INTEGRATING PK INTO DCE

This paper is intended to spark debate, not to make concrete demands for future releases of DCE. It is in that vein that we just offer some suggestions here: as a catalyst for discussion.

As has been mentioned a number of times, we focus on a hybrid approach to incorporating PK into DCE. There are a number of degrees of SK/PK hybridization in DCE that one could imagine, each with its own set of tradeoffs. A list in order of increasing PK-awareness would go something like this (other combinations not included in this list may also be possible, but this list is probably sufficient for most purposes):

(a) SK only (no PK).

  This is what DCE 1.0 and 1.1 do. With the DCE 1.1 enhancements most of the limitations of Kerberos are overcome, however cleartext long-term secrets are still stored for all accounts, so a compromise of the security datastore is catastrophic.

(b) PK login, then SK after login.

This is what DCE 1.2 adds. Clients (and servers, and machines)[26] can use PK to get their TGT, but then use SK (user-to-user) between clients and servers. DCE 1.2 has no support for many features that most people consider ''characteristic'' of PK environments (such as end-to-end client/server PK authentication, certificates, signatures, multi-crypto, key escrow, etc.). Systematic use of this hybrid gets rid of stored cleartext long-term secrets from the system (including from server keytabs), except for intercell authentication (KDS servers do not login to one another, so the PK feature is not available to them). Note that an advantage of this hybrid is that certificate processing (e.g., fetching new certificates and CRLs) can be isolated to the Registry, not forced onto every principal (assuming each `dced` can be trusted to hold the KDS's public key securely, so each principal can authenticate the KDS at login time — that's not unreasonable since machines have to be pretty highly trusted anyway).

(c) PK intercell.

This hybrid is the same as the preceding one between clients and servers (either intracell or intercell, including user-to-user), but it additionally allows cell administrators to authenticate cells via PK, instead of via the current out-of-band SK surrogate cross-registration (directly or hierarchically). Thus, in a multi-cell authentication path, the cell-to-cell authentications would happen via PK instead of SK. In the absence of a PKI, this would still require out-of-band cross-registration of public keys, so revocation would still be a problem, etc., but at least the system would be ''PKI ready'' (when a suitable PKI was available).

(d) Add PKI to DCE, and support for other PK environments.

This means adding characteristic PK features to the system, such a CA hierarchy and certificate validation (including CRL checking) via that hierarchy (instead of the out-of-band exchanges of public keys in the preceding hybrids), PK signatures, key escrow, etc. See [APKI] for the full story. Preferably, rather than being a ''part'' of DCE, the PKI system would be free-standing (licensed separately from DCE), and usable by non-DCE environments as well as by DCE. Additionally, a ''PK-DCE'' mapping should be supported (e.g., mapping between X.509 principals and DCE principals).

(e) End-to-end PK on demand for client/server authentication, using the PKI, but with some online trusted services (TTPs of various flavors) allowed.

There is a whole scale of possibilities here, depending on exactly which online services are allowed (see the Availability section). Applications doing end-to-end PK would never get tickets, i.e., would not visit the KDS. The online/offline status of the PS would be the first one to be considered (dynamicity of privilege attributes, delegation, policy domain mapping, etc.). The big change with end-to-end PK authentication over preceding hybrids is that clients and servers would have to be endowed with the full certificate validation machinery, relegating to them (instead of just the Registry) the heavy burden of certificate processing. Given that the real promise of PK is the trustlessness that goes with not having to rely on any TTPs, it's a real question whether this hybrid really makes a lot of sense. For example, if caches are to be used anyway, then CRL batch processing could be done and cached on the Registry. I.e., the incremental advantages of this hybrid over the preceding one, if any, need to be studied more carefully.

(f) End-to-end PK on demand, with no online trusted services.

Due to the large number of more-or-less trusted services that need to be present in a full-featured computing environment (as discussed in the Availability section), this hybrid would be highly problematic.

Any scheme to incorporate PK in DCE should support many options, i.e., should provide a mechanism that can support many different policies. This is because different applications and sites will have varying requirements, and in any case is compatible with the usual precedent of DCE (where RPC supports multiple communication protocols,

---

26. All the ramifications (especially compatibility) of servers and machines doing this do not appear to have been documented very well yet. If a server uses PK login, then it must register to use user-to-user (because it has no long-term key to decrypt a conventional TGT); but if it wants to, can it *also* support pre-1.2 clients (such clients cannot do user-to-user; does the server need a separate principal identity for that?). (In the case of `dced` itself, `dced` can do PK login, and can still support DCE 1.1 third-party pre-authentication of clients, by sharing a SK with the registry for that purpose.)

authentication protocols, authorization protocols, protection levels, etc.). For example, a typical policy might require PK login but SK client-server communications (such a policy should be enforceable via the DCE login set ERA). Of course, among these options there must be a least common denominator that DCE always supports, to allow maximum interoperability.

Needless to say, for product reasons compatibility must always be maintained from release to release. In particular, SK must always be supported, not matter how much PK added to DCE. In particular, DCE must have no mandatory dependence on PK (or smartcards, etc., for that matter). I.e., all core DCE services must necessarily be available by SK (and PK too, if PK-only clients are supported). Thus for example, when PK(-like) services (such as signatures) are added to DCE RPC, those services must also be available to SK-only applications (see [Yaksha]).

In light of the above degrees of hybridization, the following strawman proposal is a natural, conservative course for DCE incorporation of PK to follow. Of course, if/when standardized interfaces are available (see [APKI]), those should be used by the implementation, so that other PKIs can be substituted for the ‘‘DCE PKI’’.

Strawman proposal: On the basis of the preceding analysis, the items on the following list seem to fall out as the most reasonable PK features to concentrate on adding to DCE in the near future. Further discussion will of course be necessary to assess this claim, and to assign priorities.

(a) PKI. DCE needs the services of a PKI (see the discussion of this in the list of hybrids just above in this section). It would be nice if the general services of [APKI] were available via standard interfaces so DCE could simply consume them, but in the absence of such a thing DCE should implement a PKI (possibly separately licensable).

(b) PK login to KDS should make use of said PKI. In particular, CRL checking of client certificates should be done (instead of trusting the Registry to securely hold the client’s public keys). (However, as has been discussed, the client machine can still be trusted to hold the KDS’s public key, so the client machine doesn’t have to do CRL checking.)

(c) PK intercell (see future work promised in [NTW]).

(d) Multi-crypto.[27]

(e) Key escrow. Range of options as noted earlier (all private keys, just session keys, etc.). Very configurable (including turning escrow off).

(f) Signatures. Implemented under PSM.

(g) Smartcards for login. DCE 1.2 has the hooks for this, but we need a real proof of concept (Fortezza might make a good testbed because of its ready market).

(h) *Not* full end-to-end PK, because that gets into some gray areas that still seem to be too challenging to tackle (as discussed throughout this paper). Instead, continue to use DCE SK once logged in: software DES (or 3DES, etc.), and especially DCE authorization (PACs, ACLs, delegation, etc.), which is where DCE still offers a large value-add.

In goes without saying that in the early support of PK in DCE, only the most stable and mature services and algorithms should be supported. (E.g., investigate Yaksha for signatures and key escrow.) The more speculative technologies should be allowed to cook longer, though DCE should contain hooks for them wherever predictable.

Finally, even though this hasn’t been emphasized in this paper, it should be borne in mind that end-users are ultimately interested in the total life-cycle cost of the products they buy, not its purchase price. Hence, a high premium should be placed on the overall deployment and maintenance costs of DCE, even at the expense of lessened concern about its cost of development.

---

27. Except that because of our international character, it would be foolish for OSF to implement a non-exportable multi-crypto solution. This is a major issue we’ve tried working, but with no resolution in sight yet (sigh) . . .

5. ACKNOWLEDGEMENTS

Many of the observations in this paper arose originally in discussions with end-users who are ‘‘believers’’ in PK, but who have become concerned about the challenges of merging PK with the existing Kerberos-based SK infrastructure. However, the author accepts the blame for misrepresenting any of their concerns.

The author had intended to enlist the help of the DCE development community (especially those at DEC, HP and IBM), and of the OSF Security SIG, in producing this paper, lack of resources unfortunately prevented that from happening (except for two notable individual exceptions).

REFERENCES

Since it is assumed the reader has a strong background on the subject matter, the following references are focused and represent only a small part of the literature. Failure to cite a work should not be taken as an indication of its lack of relevance or importance.

[AndNeed] R. Anderson, R. Needham, ‘‘Robustness Principles for Public Key Protocols’’, Crypto ’95.

[AndRoe] R. Anderson, M. Roe, ‘‘The GCHQ Protocol and its Problems’’, Cambridge University Computer Laboratory, ftp://ftp.cl.cam.ac.uk/users/rja14/euroclipper.ps.Z, undated.

[APKI] B. Blakley, ‘‘Architecture for Public-Key Architecture’’, Internet RFC draft, November 1996.

[BelMer] S. Bellovin, M. Merritt, ‘‘Limitations of the Kerberos Protocol’’ Winter 1991 USENIX Conference Proceedings.

[Davis] D. Davis, ‘‘Compliance Defects in Public-Key Cryptography’’, 6th USENIX Security Symposium, 1996.

[DaSw] D. Davis, R. Swick, ‘‘Network Security via Private-Key Certificates’’, Operating Systems Reviews, 1990, pp. 64–67. Also, ‘‘Workstation Services and Kerberos Authentication at Project Athena’’, MIT Project Athena technical report (I only have the draft dated 3/17/89).

[ITAR] International Traffic in Arms Regulations, 22 CFR Parts 120−130, Federal Register Vol. 58, No. 139, July 22, 1993, pp. 39280−39326.

[LABW] B. Lampson, M. Abadi, M. Burrows, E. Wobber, ‘‘Authentication in Distributed Systems: Theory and Practice’’, ACM SOSP, 1991, pp. 165−182.

[Laurie] B. Laurie, ‘‘A Supplementary Analysis of the Royal Holloway TTP-Based Key Escrow Scheme’’, http://www.algroup.co.uk/crypto/rh.html, Nov 16, 1996.

[Mitch] C. Mitchell, ‘‘The Royal Holloway TTP-Based Key Escrow Scheme’’, Royal Holloway, University of London, ftp://ftp.dcs.rhbnc.ac.uk/pub/Chris.Mitchell/istr_a2.ps, June 8, 1996.

[NTW] B. C. Neuman, B. Tung, J. Wray, ‘‘Public Key Cryptography for Initial Authentication in Kerberos’’, Internet RFC draft (draft-ietf-cat-kerberos-pk-init-01.txt, expires Dec. 7, 1996).

[RFC 8] B. Blakley, OSF-RFC 8.1, ‘‘Security Requirements for DCE’’, October 1995.

[RFC 41] M. Romagna, R. Mackey, OSF-RFC 41.2, ‘‘RPC Runtime Support for I18N Characters — Functional Specification’’, November 1994.

[RFC 63] R. Salz, OSF-RFC 63.2, ‘‘DCE 1.2 Contents Overview’’, May 1996.

[RFC 68] A. Anderson, S. Cuti, ‘‘DCE 1.2.2 Public-Key Login — Functional Specification’’, February 1995.

[RFC 94] M. Heroux, ‘‘A Private Key Storage Server for DCE — Functional Specification’’, November 1996.

[SDSI] R. Rivest, B. Lampson, ‘‘SDSI — A Simple Distributed Security Infrastructure’’.

[Schneier] B. Schneier, ‘‘Applied Cryptography’’, 2nd edition, Wiley, 1996.

[Yaksha] R. Ganesan, ‘‘The Yaksha Security System’’, Comm. ACM, Mar. 1996, Vol. 29, No. 3, p. 55–60. Also, R. Ganesan, ‘‘Yaksha: Augmenting Kerberos with Public Key Cryptography’’, Proceedings of the Internet Society Symposium on Network and Distributed Systems Security, Feb. 1995.

AUTHOR'S ADDRESS

Walter Tuvell
Open Software Foundation
11 Cambridge Center
Cambridge, MA 02142
USA

Internet email: walt@osf.org
Telephone: +1-617-621-8764