Compliance Defects in Public-Key Cryptography

Don Davis*

March 10, 1997

Abstract

Public-key cryptography has low infrastructural overhead because public-key users bear a substantial but hidden administrative burden. A public-key security system trusts its users to validate each others' public keys rigorously and to manage their own private keys securely. Both tasks are hard to do well, but public-key security systems lack a centralized infrastructure for enforcing users' discipline. A *compliance defect* in a cryptosystem is such a rule of operation that is both difficult to follow and unenforceable. This paper presents five compliance defects that are inherent in public-key cryptography; these defects make public-key cryptography more suitable for server-to-server security than for desktop applications.

1 Introduction

Public-key cryptography is uniquely well-suited to certain parts of a secure global network. It is widely accepted that public-key security systems are easier to administer, more secure, less trustful, and have better geographical reach, than symmetric-key security systems. However, it is *not* widely appreciated that these advantages rely excessively on end-users' security discipline. In fact, the reason public-key security doesn't need a trusted key-management infrastructure is that the burden of key-management falls to public-key clients. With public-key cryptography, clients must constantly be careful to validate rigorously every public key they use, and they must husband the secrecy of their long-lived private keys. It turns out that these tasks are harder than they seem.

End-users are unwilling or unable to manage keys diligently. Perhaps surprisingly, it's impossible to automate asymmetric key management completely; certain security details remain for human intervention, such as Root-key validation, passphrase choice, and *clients'* physical security. Even where automation is possible, as with revocation-list checks, scaling problems and performance costs make short-cuts likely. If users or developers skip these details, there is no way to detect their omission or to audit the consequences. I have coined the name *compliance defect* for this situation: a rule of operation that is difficult to follow and that cannot be enforced. Compliance defects undermine the security of public-key cryptography. When users fail to manage their private keys securely, or when they fail to validate each other's public keys rigorously, then authenticity and privacy guarantees weaken, and everyone's security deteriorates. Users' behavior is the weak link in any security system, but public-key security is unable to reinforce this weakness.

This is not to say that compliance defects make public-key systems worthless. Rather, compliance defects just make public-key security unsuitable for desktop applications. Only sophisticated users, like system administrators, can realistically be expected to meet fully the demands of public-key cryptography. Accordingly, I suggest that public-key cryptography is best suited to securing communications between servers, between sites, and between organizations. Only in such large-scale infrastructural applications does public-key's geographic reach justify its substantial administrative burden of constant vigilance.

2 Public-Key Infrastructure: Review

A public-key security system comprises three infrastructural services.

- The *Certification Authority* (CA) signs users' public keys,
- The *Directory* is a public-access database of valid certificates,
- The *Certificate Revocation List* (CRL) is a public-access database of invalid certificates.

*Affiliation: Independent Consultant, 1318 Comm. Ave #16 Allston, MA 02134; *don@mit.edu*

In a wide-area network, each of these services may be deployed as a hierarchy of servers. For example, in a hierarchy of CAs, each CA has its own public-key, which is signed into the CA's own certificate by the next higher CA in the tree. However, the "Root CA," at the top of the hierarchy, has no one to sign its public key.

In the rest of this section, I summarize the administrative features of public-key security systems, with emphasis on the issues of key-handling discipline that I'll describe in the rest of the paper. [1] Key-management issues arise at the following crucial moments in the life-cycle of a user's public key certificate:

1. Key-Creation:

- The user creates a new key-pair.
- The user proves his identity to the CA (not electronically).
- The CA signs a certificate that names the user as the bearer of his new public key.
- The user also receives the Root CA's public-key, for later use.
- The user chooses a secret passphrase, and uses it to encrypt his asymmetric private key.

2. Single-Sign-On

- At login, the user types his passphrase, so as to decrypt his private key.
- With his private key, the user participates in public-key protocols.

3. Authenticating Others

- To communicate securely with other users and with networked services, the user refers to the other parties' public-key certificates.
- The user either exchanges certificates directly with other users, or he get others' certificates from the Directory service.
- Before using a certificate, the user must check the CRL for notice of the certificate's revocation, and must
- Validate the CA's signature. This step is recursive, and ends with the *out-of-band* validation of the Root CA's public key.

4. Password-Change

- The user should regularly change the passphrase with which he decrypts his asymmetric private key.

5. Key-Revocation

- Certificates are timestamped to expire after a few months or a year.
- If a user's passphrase or his private key is compromised, then he must inform the CRL administrator, who disseminates a notice that the corresponding public-key certificate has been revoked
- The user should check the CRL every time he uses a certificate, because the CRL may be updated at any moment.

The reader will notice, on reviewing this chronology, that a public-key user is frequently required to protect and validate a variety of symmetric and asymmetric keys. Unfortunately, a public-key infrastructure cannot help the user in these tasks, nor can it compel his compliance.

3 Compliance Defects

Public-key Cryptography has five unrealistic rules of use, which I call *compliance defects*. These defects correspond one-for-one with the crucial moments in a key-pair's life-cycle:

1. Authenticating the User (Issuance): How does a CA authenticate a distant user, when issuing an initial certificate?
2. Authenticating the CA (Validation): Public-key cryptography cannot secure the distribution and validation of the Root CA's public key.
3. Certificate Revocation Lists (Revocation): Timely and secure revocation presents terrific scaling and performance problems. As a result, public-key deployment is proceeding *without* a revocation infrastructure.
4. Private-Key Management (SSO): The user must keep his long-lived private key in memory throughout his login-session.
5. Passphrase Quality (PW-Change): There's no way to force a public-key user to choose a good passphrase.

[1] A similar summary for symmetric-key security systems appears in the Appendix. My outlines are loosely based on the X.509 [11] and Kerberos systems [15, 20].

The creation and revocation defects can in principle be remedied with centralized infrastructure, but the other three problems cannot be solved without the user's cooperation and intervention. Further, extra infrastructure entails scaling problems that lead immediately to shortcuts, trading away security in favor of performance. Recent proposals for public-key-based commerce have typically been naive in just this way, by ignoring the need for this extra infrastructure when in reality, the need can but momentarily be deferred.

3.1 Authenticating the User

One of the great promises of public-key cryptography is that a Certificate Authority can serve many more users than can a key-distribution service, because users only rarely have to interact with the CA. Indeed, since users get certificates monthly or annually, a CA's per-user load is a hundredth to a thousandth of a key-server's. Even accounting for the CA's greater crypto overhead, we might expect a CA to serve up to a million or more users. Unfortunately, this cheap scaling is a false promise.

The fly in the ointment is that there's more to issuing a certificate than merely calculating a digital signature. A public-key certificate is an assurance about the identity of the corresponding private key's owner. Just as the user can't trust an electronic delivery of the Root-key (see below), the CA can't trust electronic assurances of new users' identities. In both cases, what's required is a secure "out-of-band" communication. Ideally, the CA sys-admin should personally meet the new user and check his identifying documents (driver's license or passport), before signing a statement about who holds what key. Such face-to-face meetings are routine for user-account creation in the smaller installations that use symmetric keys, but truth be told, meeting a million users face-to-face isn't easy. Further, in the public-key world, users and CAs usually are widely separated, so universal face-to-face certificate issuance is really practical only for PGP hobbyists [25]. Properly authenticated certificates will have to be expensive, because of the labor cost in a face-to-face identity check. For most people, a certificate's ownership will be no more airtight than a credit card's privacy, especially since credit-card companies will be issuing certificates. Often it will be even weaker than this, since Verisign is planning to issue its lowest-level certificates by e-mail. Perhaps the only U.S. organization that already has the infrastructure necessary for correctly issuing public-key certificates is the U.S. Postal Service [21]. Certainly, the degree of certainty that one requires for a certificate varies with the application. A full discussion of such issues and their relevance to the present argument is beyond the scope of this paper, but it's clear that unauthenticated issuance offers *no* security guarantees.

It is possible to use a symmetric-key security system to authenticate a public-key certification-request. MIT has added a PGP-signing service to the Kerberos authentication system. [18] In this scheme, the authenticity of the certificate's name-to-key binding is as sound as the Kerberos account's creation was. If the user-accounts administrator checked IDs in face-to-face meetings, the Kerberized CA's certificates will be meaningful. If instead the users can register themselves remotely, then the certificates will be all but meaningless. [2]

In sum, CAs cannot scale as well as might seem possible, because account-initiation is less a technical scaling problem than a social one.

3.2 Authenticating the CA

It is a telling and ominous fact that every electronic-commerce protocol specification explicitly disavows all responsibility for the validation of the Root CA's public key [14, 16, 22, 23]. "Outside the scope of this document" is a typical waiver. [14]

Before using a public-key certificate, a user must authenticate it by checking its certifying signature and the signature on each public key in its chain of certifying authorities. It's commonly forgotten that public-key cryptography cannot afford the user any automatic procedure for validating the top-level CA key. To make sure the top-level CA key is authentic, the user has three choices:

1. Hand-checking it against an authentic paper copy;
2. Making sure that the CPU's copy is incorruptible;
3. Using a separate security system, like a smartcard or Kerberos [15, 20], to convey an authentic copy to the CPU.

It is not sufficient to pass the top-level keys inside the application-client's executable, as Netscape's Web browser does [14, 5]. Even if the executable is signed, we still have to authenticate the signature's validation-key. [3] If the attacker replaces one of the client software's top-level CA keys, by patching the

[2] At MIT, staff members usually get their accounts face-to-face, but students usually do it remotely.

[3] It is often suggested that the Root-CA can sign the executable that contains the Root-CA key, but an attacker can do this with a false CA-key, too. Such self-signed certificates are essentially meaningless.

executable, for example, then the attacker can cause the client to accept forged certificates. Unhappily, when executables come as freeware and from file-servers, this key-substitution is easy to do. Once the user accepts a forged certificate, the attacker can pose as the application server. To escape detection, the attacker just plays as a man-in-the-middle. To prevent the MITM properly, a public-key protocol should sign the plaintext, then encrypt the signature, then sign the ciphertext – clearly a burdensome process. [4] It may be more practical to ensure that the Root key, once validated, cannot be corrupted. One way to guarantee this is to keep the Root key on a smart-card. Other, weaker safeguards are to keep the Root key under the passphrase's encryption, or equivalently, for the user to sign his copy of the Root-key himself.

3.3 Certificate Revocation Lists

Before we use a public key, we must validate the key's certificate in two ways: we must check the issuing CA's signature, and we must check the current Certificate Revocation List to see if the public key is still active. The CRL is part of the public-key infrastructure's account-management system. The other parts are the CA and its practice of issuing limited-lifetime certificates. It might seem that a user needs to check his cached certificates only when he acquires them, but this is not true. It is a simple matter for a virus to corrupt a certificate cache, and a certificate may be revoked just before use, or even just after it enters the cache.

A fair, if simplistic, rule of thumb is that the cost of key-issuance plus the cost of revocation is a constant [7]. For symmetric-key systems, these costs are roughly equal, but for public-key systems, certificate revocation is much harder than issuance. Revocation is the classic Achilles' Heel of public-key cryptography. When a user's public key must be removed from use, the only way to enforce prompt revocation is to check every certificate before use against a Certificate Revocation List. Naturally, CRLs must come from a secure and highly-available service. Further, to check the CRL server's own public-key certificate, the user must refer to a higher-level CRL server, because a CRL server cannot testify about its own certificate's currency. Thus, the public-key infrastructure needs a hierarchy of CRLs, just as it needs a hierarchy of CAs. However, this real-time reliance on a centralized infrastructure negates one of the main advantages of public-key cryptography – as originally conceived, public-key protocols would avoid dependence on centralized points-of-failure. Further, note that a rigorous check of a certificate's validity requires that the public key of each CRL in the chain to the Root has to be revocation-checked, just as with signature validations. However, while a signature-check takes around 10 milliseconds, a long-haul CRL check will often take 100 or more milliseconds. This extra performance burden makes it likely that applications will often avoid revocation-checking the CRL's own certificate. This compliance defect undermines the security of any application that uses public-key cryptography.

Clearly, the timely management of CRLs is an important scaling bottleneck. The size of the CRL can be minimized by using an access-control system to revoke access, but efficient ACL management for very large networks is another unsolved scaling problem. The need for prompt revocations becomes especially acute if digital signatures have financial or contractual import. There seems to be little progress towards a national CRL mechanism.

3.4 Private-Key Management

In order to use his public key for sending and receiving secure messages, a user must either enter his passphrase anew for each use, or he must keep his private key in memory throughout his login session. Note that clients apply the private key not only to sign and decrypt e-mail, but also to initiate sessions with secure network servers. Clearly, users will not accept software that forces them to re-enter passphrases all day long, so in practice their keys will stay in memory. This is a substantial security exposure, because it exposes a long-lived secret, the private key, to physical theft. For example, if the user leaves his keyboard unattended, or if his laptop is stolen, his private key will likely be compromised. Similarly, viruses and Trojan-Horse programs have been built to steal long-lived keys. [24] So, the private key is only as secure as the user's computer. Even if the user's private key is stored in an encrypting smartcard, so that the key itself doesn't reside in memory, the card still must stay in the reader throughout the session, and so is still vulnerable to theft.

Symmetric-key systems often are designed to replace long-lived passwords in RAM with short-lived session keys, so as to minimize the passwords' in-memory exposure Short-lived asymmetric key-pairs are not very useful, though. For signature operations only, it is possible to avoid keeping the private key on hand, by using a secret signature as a temporary private key. [8, 12] This temporary key's signature can

[4] Simply signing the ciphertext doesn't work; see [3, 10, 2].

be checked with the user's public key. [5] However, if the user is to be able to decrypt private messages, he has to keep his private key in memory, and in plaintext form, throughout his login-session. The only way to keep the in-memory key safe is to keep the user's computer physically secure, and to forbid all remote access by outsiders. Clearly, users will not reliably observe these precautions.

3.5 Passphrase Quality

A public-key system has no way to enforce expiration or quality controls on passphrases, because users don't share their passphrases with any security service or administrator. It's possible, of course, for the user's local passphrase-handling software to apply such controls, but if the user finds the controls inconvenient, he can just use a more lenient program to encrypt his private key.

Without effective passphrase-QA, users' private keys are only as secure as the filesystem on which they are stored. For example, if the encrypted private keys are stored on a networked filesystem, many will be utterly vulnerable to guessing attack. This threat makes it unsafe for a user to access the net from different machines, because such logins would require unauthenticated access to the user's passphrase-encrypted key, and so would expose the key to off-line dictionary attack.

In contrast, it's easy for a trusted-party KDC to enforce a quality-control policy on each user's password, because the user must share his password with the KDC, anyway. Typically, the KDC enforces the password-expiration controls when the user logs in, and only enforces the quality-controls when the user changes his password. Then, the KDC can apply various rules and filters to ensure that the password is hard to guess. For example, the Kerberos system has incorporated most of the features of U. Texas' `npasswd` command [9], password-expiration, and other password-QA features, in a flexible password-policy mechanism. [6] Another valuable approach is proactive dictionary-checking. Purdue's OPUS project developed a filter which quickly checks passwords against a set of dictionaries, but without divulging the passwords to any trusted party. [19]

Left to their own choices, users tend to choose passwords that are easy to guess, and they tend not to change their passwords unless the security system obliges them to do so. Thus, lacking effective password-quality controls, most public-key systems are vulnerable to off-line guessing attacks. An organization's first line of defense in data security is to enforce good password hygiene, so for corporate networks, this defect is a grave one.

Table 1 summarizes the compliance defects I have discussed.

4 Transferring Administrative Burdens

A symmetric-key KDC must be highly trustworthy and highly available. In comparison, a public-key system is easier to administer centrally, because a public-key infrastructure's trust and availability requirements are more relaxed:

- The CA doesn't have to be highly available, because users rarely need new certificates. The CA is a trusted service; it cannot eavesdrop on encrypted messages, but a corrupt CA can forge a key pair and certify it in a user's name. Thus, users do have to trust the CA not to issue false certificates.

- The Directory is in essence a convenience; it saves users the trouble of exchanging their certificates with each other. The Directory is unable to forge certificates, so it requires no trust, but it should be highly available.

- The CRL has to be trusted to disseminate revocations "promptly;" depending on the application's criterion for "promptness," this may require high availability.

So, we see that some components have to be trustworthy, and some have to be highly available, but trust and high-availability are generally not required simultaneously of each public-key service.

Symmetric-key systems have another administrative burden from which a public-key infrastructure is free. The public-key infrastructure is not a bottleneck in the network, because the CA, Directory, and CRL servers don't have to mediate in every secure communication, as a KDC must do. [7] This improves not only the network's performance, but its reliability,

[5] Let N be a public message, and denote its secret signature by N^d. To sign a message m, the signer calculates $S_m = (N^d)^m$; S_m and N together make up the temporary key's signature. To check the temporary-key signature, the verifier calculates ${S_m}^e$ and N^m. In a valid signature, these values will be equal, because $(N^{dm})^e = (N^{de})^m = N^m$

[6] This password-policy mechanism is part of the Krb V5 Admin server software, which OpenVision Technologies wrote and contributed to MIT's Kerberos source-distribution. OpenVision has offices in Cambridge, Mass., Pleasanton, Calif., and London, UK.

[7] The CRL is a bottleneck, but many applications can temporarily waive CRL-checks, during lapses in the CRL's service.

	Public-Key	Symmetric-Key
Adding New Users	bad scaling	bad scaling
Revocation	bad scaling	easy scaling
Out-of-Band Validation	Root Key: hard, frequent	1^{st} PW: easy, only once
Theft Exposure	long-lived key valuable	short-lived key little value
Password-Quality	optional	enforced
Network Bottleneck	CRL service	KDC
Physical Security	client, CA, CRL	KDC
Key-Mgt Responsibility	end-user	sys-admin

Table 1. Compliance defects and administrative differences between public-key and symmetric-key security systems.

too, because the KDC is a symmetric-key network's "single point of failure."

In summary, a public-key security infrastructure has four advantages over a symmetric-key KDC:

- Less trust,
- Lower availability demands,
- Better performance,
- Better reliability

However, these attractive features come at the cost of transferring corresponding burdens onto users. The first such transfer is well-known: public-key cryptography entails a lot of local computation. Poor local performance is the price of avoiding the KDC's bottleneck in network performance and reliability. Essentially, users avoid waiting through one or two seconds of extra network-latency, by spending a comparable period on bignum arithmetic.

The second administrative transfer is the focus of this paper: the public-key infrastructure is less trustful and less available, and hence is easier to administer, but only because the hard part of public-key administration is local and cannot be centralized. Contrary to general belief, public-key cryptography does not abolish administrative trust and diligence. Instead of the users having to trust and rely upon a central organization, every user is responsible for administering his own security. What's worse, an organization's overall security depends primarily on all users to be diligent in their key-management duties.

This reliance on users' diligence is utterly unrealistic. Anderson has collected many case-studies of poor security practices among financial users of symmetric-key security [1]. He consistently found that application-programmers and end-users do not understand, and will not perform, simple key-management duties. Since symmetric-key users have a much lighter key-management burden than public-key security would impose, it is plain that compliance will be the weak link in public-key-secured networks, too.

5 Repair

For a mass-market public-key system to solve these problems, it would have to incorporate highly-available, trusted, and secure servers. Such solutions would add centralized infrastructure costs to public-key's already substantial performance costs. At the cost of introducing administrative trust, symmetric-key systems solve all but the problem of long-distance account-creation, which is hard for any security system.

We can combine both cryptosystems' administrative benefits, by restricting public-key deployment to servers, and by using symmetric-key protocols for desktop clients. [5] The clients' KDC can enforce password-quality, issue short-lived keys, validate servers' public keys, and maintain CRLs. For signatures and asynchronous messaging, a symmetric-key-based signature system can mediate between native symmetric-key users and external public-key users. [13, 6] This hybrid security system would put public key-pairs only in the hands of well-trained sysadmins, and would also minimize the CRLs' scaling problems. Hybridization trades away theoretically perfect privacy, so as to strengthen public-key's practical weak link: user compliance.

Smart-card hardware can repair some compliance defects, but they fall far short of completeness. Smartcards are completely effective only for Root-key validation. For passphrase QA and private-key

management, smartcards just substitute the problem of physical security. Smartcards offer no help for the problems of authenticated issuance and revocation.

6 Conclusion

Compliance defects impede the sound management of keys and of user-accounts. These defects have arisen from the introduction of public-key cryptography into mass-market software As public-key security was originally envisioned during the '70's and '80's, sophisticated users and sensible key-hygiene were taken for granted. For example, Privacy-Enhanced Mail's designers explicitly expected that the professionals who then used e-mail would be able to hand-check their copy of the Root-CA's public key. With the advent of mass-market electronic commerce, this assumption no longer obtains. Nowadays, the security system must be transparent wherever possible, and where transparency fails, it must enforce good key-hygiene.

Public-key cryptography is actually no more "trustless" than symmetric-key security systems. Public-key's decentralized nature actually places a lot of trust on *users*, that properly belongs to the security infrastructure and its administrators. Up to now, this trust in users' discipline has been implicit, and has received little or no attention in discussions of the Internet's security infrastructure. However, it's time to re-assess public-key's "trustlessness," as we approach the large-scale deployment of public-key protocols for electronic commerce.

7 Acknowledgements

Dan Geer, Barry Jaspan, Win Treese, Jon Gossels, Karl Andersen, and Brad Johnson were helpful when I discussed these ideas with them. I also received helpful critique from the USENIX referees.

References

[1] R.J. Anderson, "Why Cryptosystems Fail," *Comm. ACM*, v bf 37 no. 11 (November 1994), pp. 32-40

[2] R.J. Anderson, R.M. Needham, "Robustness Principles for Public-Key Protocols," *Advances in Cryptology – CRYPTO '95*, Springer-Verlag, Berlin, 1995.

[3] M. Burrows, M. Abadi, and R. Needham, "A Logic of Authentication," *Proc. R. Soc. Lond. A* 426(1989) pp. 233-271.

[4] D.A. Curry, *UNIX System Security: A Guide for Users and System Administrators*, Addison-Wesley Professional Computing Series (Reading, Mass.) 1992.

[5] D. Davis, "Kerberos Plus RSA for World Wide Web Security," *Proc.* 1^{st} *USENIX Workshop on Electronic Commerce* (NYC, 7/95), pp.185-8.

[6] D. Davis and R. Swick, "Network Security via Private-Key Certificates," *USENIX* 3^{rd} *Security Symposium Proceedings*, (Baltimore; Sept. '92) pp. 239-42. Also in *ACM Operating Systems Review*, v.24, **4** (Oct. 1990).

[7] D. Geer and J. Rochlis, "Network Security: The Kerberos Approach," Usenix Workshop Tutorial.

[8] L. Guillou and J. Quisquater, "A Practical Zero-Knowledge Protocol Fitted to Security Microprocessor Minimizing Both Transmission and Memory." *Advances in Cryptology – EUROCRYPT '88*, Springer-Verlag, Berlin, 1988.

[9] Clyde Hoover wrote the `npasswd` command at U. Texas at Austin: *ftp://emx.utexas.edu/pub/npasswd*. For a concise description, see [4], p. 171.

[10] C. I'Anson and C. Mitchell, "Security Defects in CCITT Recommendation X.509 - The Directory Authentication Framework," *ACM Comp. Comm. Rev.*, (Apr '90), pp. 30-34.

[11] International Telegraph and Telephone Consultative Committee (CCITT). Recommendation X.509: The Directory - Authentication Framework. In *Data Communications Network Directory, Recommendations X.500-X.521*, pp. 48-81. Vol. 8, Fascicle 8.8 of *CCITT Blue Book*. Geneva: International Telecommunication Union, 1989.

[12] C. Kaufman, R. Perlman, and M. Spencer, *Network Security: PRIVATE Communication in a PUBLIC World*, Prentice-Hall Series in Computer Networking and Distributed Systems, (Englewood Cliffs, NJ) 1995, pp. 436-8.

[13] B. Lampson, M. Abadi, M. Burrows, E. Wobber , "Authentication in Distributed Systems: Theory and Practice" *13th ACM Symposium on Operating Systems Principles* pp. 165-182, Oct. 1991

[14] Netscape Communications, "Secure Socket Layer Reference Document," Unofficial Internet Draft.

[15] C. Neuman and J. Kohl, *The Kerberos Network Authentication Service (V5)*, Internet RFC 1510, September 1993.

[16] E. Rescorla and A. Schiffman, "Secure Hypertext Transfer Protocol," Internet Draft RFC, May '95.

[17] R. Rivest, A, Shamir, L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Comm. ACM*, v. 21, **2**, Feb. '78, pp. 120-126.

[18] J.I. Schiller and D. Atkins, "Scaling the Web of Trust: Combining Kerberos and PGP to Provide Large Scale Authentication," *USENIX Winter Conference Proceedings*, January 1995.

[19] E.H. Spafford, "Observing Reusable Password Choices," *USENIX* 3^{rd} *Security Symposium Proceedings*, (Baltimore; Sept. '92), pp. 299-312.

[20] J.G. Steiner, C. Neuman, and J.I. Schiller, "Kerberos: An Authentication Service for Open Network Systems", *USENIX Winter Conference Proceedings*, February 1988. [athena-dist.mit.edu:pub/kerberos/doc/usenix.PS]

[21] *USPS Electronic Commerce Services*, product information sheet (Washington, DC), 1995.

[22] Visa International and MasterCard, "Secure Electronic Transactions Protocol Specification."

[23] Visa International and Microsoft Corp., "Secure Transaction Technology Specifications."

[24] A. Young, M. Yung, "The Dark Side of Black-Box Cryptography -or- Should We Trust Capstone?" *Advances in Cryptology – CRYPTO '96*, Springer-Verlag, Berlin, 1996.

[25] P. Zimmermann, *The Official PGP User's Guide*, MIT Press (Cambridge, Mass.) 1995.

8 Appendix: Symmetric Key-Distribution

A symmetric Key-Distribution Center is a trusted server that knows each user's password. The KDC issues temporary session-keys to users who know their passwords. Each user's initial session-key comes to him under his password's encryption. The KDC then uses this initial key to encrypt the user's subsequent session keys.

1. Account-Creation:

- The user proves his identity to the KDC's administrator (not electronically).
- The administrator creates an initial password for the user, and tells the user to change it immediately.

2. Single-Sign-On

- At login, the user types his password, so as to decrypt his daily temporary session-key.
- The user applies this session-key in a similar protocol repeatedly through the day as he accesses services, gaining a new session-key for each different server.

3. Authenticating Others

- To communicate securely with other users and with networked services, the user applies various session-keys in a simple protocol.
- In each repetition of this authentication protocol, the KDC identifies the session-key's owners to each other.

4. Password-Change

- The KDC can require the user to change his password regularly, as a condition for access.
- When the user changes his password, the KDC can examine it, so as to enforce complexity criteria on the user's choice.
- The KDC stores the new password in the database, in a hashed form.

5. Account-Revocation

- Session-keys are timestamped to expire quickly, usually after 8 hours or even a few minutes. This discourages key-theft.
- If a user's password is compromised, then he must inform the CRL administrator, who manually replaces the user's password, and tells the user to change it immediately..

The KDC's trusted role gives it potential access to all of the system's traffic. In return, the KDC takes responsibility for managing, validating, and renewing all of the system's keys. Thus, compared to a public-key system's security, all of the KDC's links are weakened somewhat, except for the weakest: the user's key-management link is greatly strengthened.