Cryptography: It's Not Just For *Electronic* Mail Anymore

J. D. Tygar   Bennet Yee

March 1, 1993

CMU-CS-93-107

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Sponsored by the Avionics Laboratory, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U.S. Air Force, Wright-Patterson AFB, OH 45433-6543 under Contract F33615-90-C-1465, ARPA Order No. 7597. Additional support for J. D. Tygar was provided under a Presidential Young Investigator Award, Contract No. CCR-8858087, and Motorola Inc.

Keywords: cryptography, franking, electronic currency, mail, postage, stamps, electronic stamps, secure coprocessors, signatures

Abstract

We extend cryptographic techniques to the protection of the application of stamps for mail. We show how to provide *electronic stamps* (using off-the-shelf bar code technology to represent a cryptographic message) to use in a fully integrated franking system that provides protection against:

1. Tampering with postage meter to given the user additional credit;
2. Forged or copied electronic stamps;
3. Unauthorized use of a postage meter; and
4. Stolen postage meters.

We relate the question of electronic stamps to broader issues in electronic currency and secure coprocessors.

1 Introduction

While cryptographic methods have long been associated with mail (dating back to the use by Julius Caesar described in his book *The Gallic Wars* [3]), they have generally been used to protect the contents of a message, or in rare cases, the address on an envelope (this is known as "protecting against traffic analysis"). In this technical note, we describe the advantages of using cryptographic techniques to protect the *stamp* on an envelope!

Consider the case of the US Postal Service, with almost 40,000 autonomous post office facilities, which handles over 165 billion pieces of mail each year. The vast majority are metered, which means that the envelope is imprinted by machine with a postage amount. (Figure 1 shows an example of a metered letter.) Each postage meter is sealed with a postage credit by a post office; as each letter is stamped, the amount is deducted from the machine's credit. Postal meters are subject to at least four types of attack:

1. The postage meter recorded credit may be tampered with, giving the user postage not paid for;

2. The postage meter stamp may be forged or copied;

3. A valid postage meter may be used by an unauthorized person; and

4. A postage meter may be stolen.

With modern facilities for barcoding digital information which is machine readable, it is now easy to replace old-fashioned human readable stamps by stamps which are entirely or partially machine readable. These stamps could encode a digitally signed message which would guarantee the authenticity of the stamp. If this digital information included unique data about the letter (such as the date mailed, zip codes of the originator and recipient, etc.), the digitally signed stamp could be used to protect against forged or copied stamps. A rough outline of how such a system might work was detailed by Pastor [19].

Unfortunately, a digitally signed stamp is vulnerable to four additional types of attack:

1. Cryptographic techniques are vulnerable to misused, leading to a system that can be successfully attacked by an adversary.

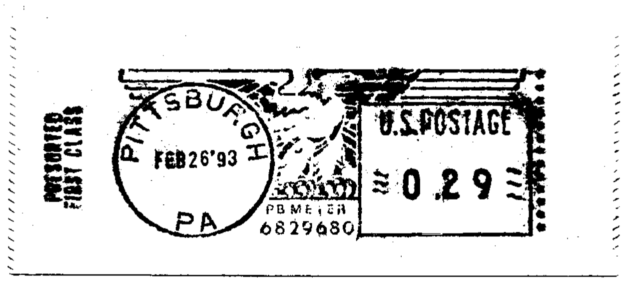

Figure 1: Today's metered letters have a simple imprint that can be easily forged.

2. The postage meter credit may still be tampered with, even if cryptographic techniques are used.

3. A postage meter may be opened up and examined by an adversary to discover any cryptographic keys that are used, thus allowing the adversary to build new bogus postage meters.

4. The protection scheme may depend on a highly available network connecting post office facilities in a large distributed database. Since 39,985 autonomous post office facilities exist, such a network would suffer from frequent failures and partitions. Moreover, with a volume of 165,850,000,000 pieces of mail each year, a database to check the validity of digitally signed stamps appears infeasible.

This technical note outlines a protocol for protecting electronic stamps, and demonstrates that the use of currently available *secure coprocessor* technology can address all of the above concerns. With the use of cryptography and secure coprocessors, both postage meters and stamps can be made fully secure and tamper-proof.

2 Cryptographic Stamps and Electronic Postage Meters

What do we mean by a cryptographic stamp? What properties must an electronic postage meter have? By examining the properties that we wish cryptographic stamps to have, we will obtain the overall system requirements for electronic postage meters.

A cryptographic postage stamp is an imprint that demonstrates to the postal carrier that postage has been paid. Unlike the usual stamps purchased at a post office, these are printed by a laser printer and affixed onto an envelope or a package. Because such printed stamps can be easily copied, cryptographic and procedural techniques must be employed to minimize the probability of forgery.

We use cryptography to provide a crucial property: a malicious user may copy a stamp, but any attempts to *modify* it will be detected. To achieve this goal, we encode as part of the stamp all the information relevant to the delivery of the particular piece of mail — e.g., the return as well as the destination address, the amount of postage, and class of mail, etc — as well as other identifying information, such as the serial number of the postage meter, a serial number for the stamp, and the date/time (a *timestamp*). All of the information is digitally encoded and then *signed* cryptographically, preventing forgeries. This information along with the cryptographic signature is put into a barcode format printed via a laser printer. The encoding format must be easily printable by "commodity" or "after-market" laser printers, it must be easily scanned and re-digitized at a post office, and it must have sufficient information density to encode all the bits of the stamp on the envelope within a reasonable amount of space. Appropriate technologies include Code49[18], Code16K[12], and PDF417 [21, 20, 11]. Symbol Technologies' PDF417, in particular, is capable of encoding at a density of 400 bytes per square inch, which is sufficient for the size of cryptographic stamps needed to provide the necessary security in the foreseeable future. Figure 2 demonstrates the amount of information that can be encoded[1].

The cryptographic signature within the stamp prevents many forms of "replay" attacks. Malicious users will not find it that useful to copy the stamps, since the cryptographic signature prevents them from modifying the stamp to change the destination addresses, etc, so the copied stamps may only be used — barring duplicate detection — for sending more mail to the same destination address. The timestamps and serial numbers also help to limit the scope of the attack, limiting the "lifetime" of copies and permitting law enforcement to trace the source of the attack back to a unique postage meter.

Because the cryptographic stamp also includes source information — the postage meter serial number provides the general physical location of the meter,

[1]This figure was provided courtesy of Symbol Technologies Inc.

Figure 2: PDF417 encoding of The Gettysburg Address

as does the return address — duplicated stamps can also be detected in a distributed manner. Replays are detected by logging the recent, unexpired stamps from all postage meters that have been canceled by the postal system. If ever a piece of mail with a duplicate stamp is found, we will know that some form of forgery has occurred. We will examine the practicality of replay detection later in Section 5.4.

While databases at region offices can deter replay attacks, we need some way to protect the cryptographic keys within the postage meters as well — attackers who gain access to the keys within a secure coprocessor can use them to fraudulently sign as many stamps as they wish. The only way to prevent malicious users from accessing cryptographic keys require not just physically protected memory (such as found in many "dongle" type devices) but also require secure processing of the cryptographic keys. Without the ability to privately perform computations using the cryptographic keys, an adversary may simply place logic analyzer probes to listen in on the address/data buses and obtain key values. Alternatively, the adversary may replace the memory subsystem in the computer with dual ported memory, and just read the keys as they are used. Further, note that even password protected, physically secure memory such as that that provided by some dongles (e.g., Dallas Semiconductor DS142x "Software Authorization Button" with interface DS1410) is insufficient — the PC software must contain the passwords required to access that protected memory, and even if attackers don't know how to disassemble the software to obtain the passwords, they can easily read it off of the wires of the parallel port as the passwords are sent to the dongle.

Private processing of cryptographic keys is a necessary condition for cryptography. Not only is this requirement necessary for running real cryptographic protocols, it is also a requirement for keeping track of the amount of credit that remains in a electronic postage meter. Along with the secure credit counter, more protected computation is required to establish secure channels of communication when the credit value is updated — the electronic postage meter must communicate with the post office when the user buys more postage, and cryptographic protocols must be run over the communication lines (e.g., phone lines via modem or some other kind of direct network connection) to prevent tampering with the update. Establishing a secure communication channel requires cryptography, and again we need a safe place to keep cryptographic keys and a place to perform secure computation.

To achieve private, tamper-proof computation, a processor with non-volatile memory for key storage, and perhaps some normal RAM as scratch space (to hold intermediates in the calculations) must also be made physically secure. We call such a system a *secure coprocessor*[26]. In the next section we will give a brief summary of its properties.

3 Secure Coprocessor Model

A secure coprocessor is an ideal place to perform secure, private computation such as that required in *all* cryptographic protocols. A secure coprocessor is a hardware module containing (1) a CPU, (2) ROM, and (3) NVM (non-volatile memory). This hardware module is physically shielded from penetration, and the I/O interface to this module is the only means by which access to the internal state of the module can be achieved. Such a hardware module can be used to store cryptographic keys without risk of release of those keys. More generally, the CPU can perform arbitrary computations and thus the hardware module, when added to a computer, becomes a true coprocessor. Often, the secure coprocessor will contain special purpose hardware in addition to the CPU and memory; for example, high speed encryption/decryption hardware may be used[2].

[2] High speed DES chips are available from several sources, including Cylinks, Comtech, IBM[17, 29], and DEC[7].

The packaging technology required to achieve the physical protection exists today. Examples of this at varying scales include the μAbyss and Citadel systems at IBM Research [29, 28] and NSA's proposed DES replacement SKIPJACK [2, 22]. Smart cards [14] also achieve many of the same privacy properties by the virtue of their portability: users carry smart cards in their wallets rather than leaving them on top of desks.

Advanced packaging technology can protect secure coprocessors — we assume that the coprocessor is packaged in such a way that physical attempts to gain access to the internal state of the coprocessor will result in resetting the state of the secure coprocessor (i.e., erasure of the NVM contents and CPU registers). An intruder may break into a secure coprocessor and look inside to see how it's constructed; the intruder can not, however, affect or learn the internal state of the secure coprocessor except through normal I/O channels or by forcibly resetting the entire secure coprocessor. The guarantees about the privacy and integrity of non-volatile memory provide the foundations needed to build secure systems.

3.1 Physical Assumptions for Security

Our basic assumption is the existence of private and tamper-proof processing in a coprocessor. Just as attackers can exhaustively search cryptographic key spaces, it may be possible to overcome the physical protections by expending enormous resources (possibly feasible for very large corporations or government agencies), but we will assume the physical security of the system as an axiom. This is a physical work-factor argument, similar in spirit to intractability assumptions of cryptography. Our secure coprocessor model does not depend on the particular technology used to satisfy the work-factor assumption. Just as cryptographic schemes may be scaled to increase the resources required to penetrate a cryptographic system, current security packaging techniques may be scaled or different packaging techniques may be employed to increase the work-factor necessary to successfully bypass the physical security measures.

3.2 Limitations of Model

Even though ideally (for security purposes) we would like to somehow make the scales and the postage printer physically secure also, such a goal is impractical — the secure postage meter must be accessible to ordinary users rather than be locked away in vaults. Furthermore, current packaging technology limits the amount of circuitry that we can protect to approximately one printed circuit board in size due to heat dissipation and other concerns.

Future developments in packaging techniques may eventually relax this and allow us to make more solid-state components physically secure (e.g., more memory), but the security problems of secure communication (for credit transfers) and external mass storage remain a constant.

Countering our desire to encapsulate the entire system within a physically protected environment, the desire to make a system easy to maintain forces us to use modular design and construction techniques. Once a hardware module is encapsulated in a physically secure package, disassembly of the module to fix or replace some defective component will probably be very difficult if not impossible. The right balance between physically shielded and unshielded components will depend on the applications for which the system is intended and the reliability and security properties desired.

4 Electronic Currency and Postage

In the original form of imprints made on letters/packages, postage "stamps" were ink-stamps which showed that the postage has been paid. In the form of the everyday postage stamps that one purchases at the post office, stamps are transferable single-use tokens that enable a piece of mail to be delivered — the cancellation of the stamp uses it up and "pays" for the delivery service. Abstractly, then, we may think of postage stamps as another form of currency: we are doing nothing more than currency exchange when we go to the post office to buy stamps. Unlike the gold standard, stamps-as-currency are backed by this promise of delivery service[3]. In the past people have accepted postage stamps in lieu of currency.

Postage meters provide an intermediate model — a postage meter is a *container* of currency: meters are "refilled" at a post office when currency exchange takes place, and the filled postage meters can then dole out postage as needed at more convenient locations. When postage is used, the meter marks envelopes to show proof of payment. Thus, abstractly, a postage meter is a device that holds currency, allowing only two operations: (1) a post office may increment the amount of available postage, and (2) a user may decrement the amount of available postage, simultaneously stamping a piece of mail with the quantity just decremented. We

[3]Unlike other currencies, stamps have a 1-1 conversion rate with Federal Reserve Notes, and "inflation" occurs when postage increases are decreed.

will show how we can achieve an electronic version of the old physical postage meter that provides the same operational properties along with far greater security guarantees.

Protecting the money (postage) contained in an electronic postage meter is a weaker form of the electronic currency problem: we do not need to transfer currency between postage meters very often, nor are the anonymity requirements quite so strict. We can view the *use* of that electronic currency to stamp mail as simply a purchase using the electronic currency, where the stamp is cryptographically protected from the attacks mentioned above in Section 1.

Before we look at implementing a secure electronic postage meter, we first examine the properties of several electronic currency models and how secure coprocessors enable their implementation.

4.1 Electronic Currency Models

There are several models that can be adopted for handling electronic funds. After briefly describing these models, we examine their properties and deduce their system requirements. Furthermore, we examine whether secure coprocessors can accommodate these requirements. We assume in our analysis that we can equip every workstation or PC with a secure coprocessor.

The first model is the cash analogy. Electronic funds are treated as cash and have the same properties: (1) exchanges of cash can be effectively anonymous, (2) cash can not be created or destroyed, (3) cash exchanges require no central authority. (Note that these properties are not absolute even with cash — serial numbers can be recorded to trace transactions, and the U. S. Treasury regularly prints and destroys money.) Normal postage stamps have much the same properties.

The second model is that of a credit cards/checks analogy. Electronic funds are not transferred directly; rather, promises of payment — perhaps cryptographically signed to prove authenticity — are transferred instead. A straightforward implementation of this model fails to exhibit any of the three properties above; by applying cryptographic techniques[4], anonymity can be achieved, but the latter two requirements remain insurmountable. Checks must be signed and validated at central authorities (banks), and checks/credit payments en route "create" temporary money. Furthermore, the potential for reuse of cryptographic signed checks requires that the recipient must be able to validate the check with the central authority prior to committing in a transaction.

The third model is analogous to a rendezvous at the bank. This model uses a centralized authority to authenticate all transactions and so is even worse for large distributed applications. However, this scheme — and to some extent the previous one — makes the problem of security less difficult. The bank is the sole arbiter of the account balance and can easily implement the access controls needed to ensure privacy and integrity of the data. This is essentially the model used in Electronic Funds Transfer (EFT) services provided by many banks — there are no access restrictions on deposits into accounts, so only the owner of the source account needs to be authenticated.

4.2 Currency Model Analysis

Let us examine these models one by one. What sort of properties *must* electronic cash have and what sort of system requirements do these properties impose on us?

With electronic currency, integrity of the accounts data is crucial. Attackers should not be able to inject or replay messages during currency transfers and otherwise change the outcome of the transfers. This is especially important in the cash model, where secure coprocessors are the users' wallets and, unlike the other models, there are no central agency for resolving disputes.

By using the privacy assumption we can establish a secure communication channel between two secure coprocessors via encryption. This allows us to maintain privacy when transferring funds. By running a key exchange protocol to obtain new, single-use session keys every time we set up a connection with another secure coprocessor, communication based attacks are prevented: data that were recorded from a previous session would be meaningless if subsequently re-injected into the communication channel.

To ensure that electronic money is conserved (neither created nor destroyed), the transfer of funds should be failure atomic, i.e., the transaction must terminate in such a way as to either fail completely or fully succeed — transfer transactions can not terminate with the source balance decremented without having incremented the destination balance or vice versa. By running a transaction protocol such as two-phase commit [5, 1, 30] on top of the secure channel, the secure coprocessors can transfer electronic funds from one account to another in a safe manner, providing privacy as well as ensuring that money is conserved throughout. With most transaction protocols, some "stable storage" for transaction logging is needed to enable the system to be restored to the state prior to the transaction when a transaction aborts. On large transaction systems this typically has meant mirrored disks with uninterruptible power supplies. With the simple transfer transactions needed, the per-transaction log typically is not that large, and the log can be truncated once transactions commit. Because secure coprocessors need to handle only a handful of users (workstations or PCs are typically single user machines) and the users are unlikely to require many concurrent transactions, large amounts of stable storage should not be needed — we have non-volatile memory within the secure coprocessors and we only need to reserve some of this memory for logging.

The transaction logs, accounts data, and controlling executable code are all protected by the secure coprocessor from modification: accounts are safe from all but bugs and catastrophic failures. Of course, the system should be designed so that users should have little or no incentive to destroy secure coprocessors that they can access — which should be natural when their own balances are stored on secure coprocessors, much as real cash inside of wallets.

Note that this type of decentralized electronic currency is *not* appropriate for smart cards unless they can be made physically secure from attacks by their owners. Smart cards are only quasi-physically-secure in that their privacy guarantees stem solely from their portability. Secrets may be stored within smart cards because their users can provide the physical security necessary. Malicious users, however, can easily violate the integrity of their smart cards and insert false data.

This electronic cash transfer is analogous to the transfer of rights (not to be confused with the copying of rights) in a capability based protection system [31]. Electronic cash may be used to purchase capabilities which are *single-use* rights, i.e., tokens that are consumed by leasing a computer program or printing a cryptographic stamp.

What about the other models of handling electronic funds? With the credit cards/checks analogy, the authenticity of the promise of payment must be established. When the computer can not keep secrets for users, there can be no authentication because nothing uniquely identifies users. Even when we assume that users can enter their passwords into a workstation without having the secrecy of their password be compromised, we are still faced with the problem of providing privacy and integrity guarantees for network communication. We have similar problems as in host-to-host authentication in that cryptographic keys need to be exchanged somehow. If communications is in the clear, attackers may simply record a transferral of a promise of payment and replay it to temporarily create cash.

With the bank rendezvous model, the "bank" supervises the transfer of funds. While it is easy to enforce the access controls on account data, this suffers from problems with non-scalability, loss of anonymity, and a very real risk of denial of service from excessive centralization: Because every transaction must contact the bank server, access to the bank service will be a performance bottleneck. The system does not scale well to a large user base — when the bank system must move from running on a single computer to a several machines, distributed transaction systems techniques must be brought to bear anyway, so this model has no real advantages over the use of secure coprocessors in ease of implementation. Users have no assurance that their privacy is being protected, since the central bank computer may expose its logs without the users' knowledge. Furthermore, denying access to the bank host — whether by crashing it directly, by cutting network feeds to it, or just due to normal hardware failures — means that nobody can make use of any bank transfers. This model does not exhibit graceful degradation with system failures.

The secure coprocessor managed electronic currency model not only can provide the properties of (1) anonymity, (2) conservation, and (3) decentralization but it also degrades gracefully when secure coprocessors fail. Note that secure coprocessor data may be mirrored on disk and backed up after being properly encrypted, and so even the immediately affected users of a failed secure coprocessor should be able to recover their balance. The security administrators who initialized the secure coprocessor software will presumably have access to the decryption keys for this purpose — careful procedural security must be required here. The amount of redundancy and the frequency of backups depends on the reliability guarantees desired; in reliable systems secure coprocessors may continually run self-checks when idle and warn of impending failures.

5 Personal Computer Based Postage Meters

By using secure coprocessors in a PC-based system, we can build a secure postage meter that is suitable for many businesses. The equipment required for our electronic postage meter is a secure coprocessor, a PC (which serves as the host for the coprocessor), a laser printer, a modem, and optionally an optical character recognition (OCR) scanner and/or a network interface. Like ordinary postage meters, our PC-based postage meter operates in an office environment where the postage meter is a shared resource, much as laser printers are today.

The basic idea is simple: we obtain the destination and return addresses and weight/delivery class from the user — directly from the word processor running on the user's own PC via the local network[4], by using OCR software and reading directly from the envelope, or by direct user input at the keyboard — and request a cryptographic stamp from the secure coprocessor. The secure coprocessor lowers the credit value inside it, and generates a cryptographically signed message containing the value of the stamp, all of the addressing information, the date, the ID of the secure coprocessor, and other serial numbers. This message (a bit vector) is sent to the PC, which encodes it in a machine readable manner and prints it on the laser printer to be affixed to an envelope or package. Advanced bar coding technology such as PDF417 mentioned in Section 2 may be employed.

5.1 Postage Meter Currency Model

Postage credits held within an electronic postage meter are simpler than general electronic currency because of their restricted usage. Postage credits must be purchased from the post office, and the credits may only be used to purchase cryptographic stamps (or be transferred to another electronic postage meter).

Because of the usage restrictions, we do not have to deal with the general electronic currency problem. We can take advantage of these restrictions to the currency model and achieve simpler solutions. Furthermore, because pieces of mail stamped by electronic postage meters are likely to be all mailed in the same locality, the problem of detecting replays can be solved with much lower overhead than otherwise by exploiting this locality property.

5.2 Reloading a Meter

Only post offices may reload postage meters. Unlike their older brethren, electronic postage meter equipment need not be carried to the local post office when the amount of credit inside runs low — the local post office can simply provide a phone number whereby businesses may call to "recharge" their electronic postage meter by phone, using credit card numbers or direct electronic funds transfer for payment. By equipping electronic postage meters with modems (much like newer vending machines), we can update postage meters' credit remotely and automatically. Note, however, that regardless of whether an electronic postage meter uses a phone or a direct wire connection to update its postage credit, some

[4]The word processing software can even provide good weight estimates since it knows how many pages are being printed.

method must be used to protect the meters' communication with the local post office. Otherwise, a malicious user may record the control signals used to update an electronic postage meter's credit balance and replay that recording into another postage meter. To protect the electronic postage meters from this form of replay attack we can simply encrypt all the data transmitted. By using encryption to protect the communication channel, we also protect businesses' credit card or EFT account numbers being sent over the communication lines from being used by malicious eavesdroppers.

To establish an encrypted communication channel, we must first obtain encryption keys to be used for the channel. Temporary session keys may be acquired by running a key exchange protocol — this minimizes the risks of exposure of the permanent private keys and allows for the use of faster symmetric cryptosystems such as DES for the encryption of the bulk of the data. In addition to exchanging keys, some form of authentication protocol is required to assure that the postage meter is indeed talking to the post office and that the post office is indeed talking to a valid postage meter. Some key exchange protocols such as Diffie-Hellman[6] protocol do not provide any identification information, where as others such as RSA[25] or the Strongbox algorithm [27] may be used to simultaneously exchange keys as well as authenticating identities.

On top of the secure, encrypted communication channel, we may run a transaction protocol such as two-phase commit [10, 9, 8] which makes the entire credit transfer occur in a failure atomic way. The nonvolatile memory within the secure coprocessor provides permanence of the data, and serializability is easily obtained due to the limited scopes of the transactions.

5.3 Cryptographic Protection of the Stamp

Stamps, as mentioned in Section 2, must be cryptographically signed to prevent any alteration. This may be achieved using a public key system such as RSA[25], the Rabin function [23], or the recently proposed Digital Signature Standard[16], either alone or in conjunction with a cryptographic hash function [13, 24, 15].

Cryptographic stamps consist of the cryptographic signature of the source and destination addresses (full addresses, not just ZIP+4), hierarchical authorization number (ID of authorizing post office computer), postage meter serial number, stamp sequence number, amount of postage and postage class, and the time and date. In addition to the signature, this information may optionally appear in unencrypted form or plaintext to simplify processing stages where we require fast, easy access to the addresses. A quick back of the envelope calculation shows that if we allow 6 lines of 40 full ASCII characters for each address, four bytes each for hierarchical authorization number, the postage meter serial number, the stamp sequence number, the postage/class, and the time, this is a little less than 500 bytes of data. The addressing data can certainly be more compactly encoded, and even if we encode a complete plaintext copy as well as the cryptographically signed copy, this is less than 1K of data. It would require just over two and one half square inches when encoded using PDF417.

5.4 Detecting Replays

To make replay detection easier, we exploit the physical locality property: pieces of mail stamped by a postage meter are likely to enter the mail processing system at the same region office. Therefore, cryptographic stamps from the same electronic postage meter are very likely to be canceled at the same region office, and we can simply put a database at that facility containing a list of all the electronic stamps from that meter that have been canceled. If any cryptographically stamped piece of mail is sent to another mail cancellation site than the usual one, network connections can be used for real-time remote access of cancellation databases, or removable media such as computer tapes may be used for batch-style processing. In the case of real-time cancellation, the network bandwidth required depends on the probability of the occurrence of such multi-cancellation-site processing, and on how quickly we need to detect replays. The canceled stamps database at each region office need not be large — because each postage meter can simply supply a counter value in each of its stamps, we need only fast access to a bit vector of the recently used, unexpired stamps. These bit vectors are indexed by the postage meter's serial number, and can be compressed by run-length encoding or other techniques. Only when a replay is detected might we need access to the full routing information.

Detecting replays locally is feasible with today's technology. Using the 1992 figures of 1.6×10^{11} pieces of mail per year handled at 6×10^{2} regional offices, we can obtain an estimate of the storage resources required. Assuming that we set the expiration of cryptographic stamps at 6 months after printing date, it would mean that an average regional office will need to keep track of an average of 1.3×10^{8} stamps at a time. Even if we store one kilobyte of information per stamp (doubling the estimate above) and assume that the entire current mail volume is converted to using cryptographic stamps, this would require only 130 gigabytes of disk storage per facility, well within the capacity of a single disk array system today. A fast, bit vector representation for duplicate detection would require 1300 megabits of storage plus indexing overheads, or just 17 megabytes plus overhead — an amount of storage that can easily fit into the RAM of a single average PC. While additional space may be required for indexing to improve throughput and for replication (disks) to improve reliability, the amount of storage required is quite small.

5.5 Key Management

In the discussion so far, we have had two public keys associated with each electronic postage meter: the first is the key that is used for key exchange when the meter is to be reloaded, and the second is the key that is used to sign cryptographic stamps. Corresponding to each of these private keys is a public key, and these public keys are maintained by the post office computers. The public/private key pairs are created when postage meters are loaded with their software and can never change — at best, an electronic postage meter may be reinitialized by the post office with new keys — the secure coprocessor firmware will only allow private keys to be destroyed and not revealed. While it is possible to use a single public/private key pair for both key exchange and signatures, greater care must be taken to avoid exposing the private key due to different usages by the two protocols.

The cryptographic stamp signature keys and the key exchange keys are protected by the physical security properties of secure coprocessors. The database of electronic postage meter serial numbers and corresponding signature public keys are accessed mostly by the local post office and can therefore be maintained on servers at the local post office along with the canceled stamps data. Cryptographic keys may be scaled in size to provide different amounts of protection. For the RSA or Rabin cryptosystem, 330 bit (or 100 decimal digit) composite numbers suffices to provide security well into the next 50 years. Thus, the storage requirements for the public keys database are modest — rounding to 500 bits of key data and assuming one postage meter for every thousand pieces of mail, this is only two thousand public keys per regional office, or 125Kbytes of data.

A master database that maps from postage meter serial numbers to public key servers may be useful, though the return address data should be sufficient to restrict the number of post offices that must be queried to locate the public key. A master database would have 1.6×10^8 entries; assuming a 32-bit word per entry for a ZIP+4 code, this is a 640 megabyte table. Since this data is used in a read-only fashion and should be accessed relatively infrequently, we may simply just locally cache the portions of the database from a central server as it is needed or even publish the entire database as a CD-ROM. There are few cache coherency problems — in the unlikely case that the public key is most frequently used at the locale of a different postage handling facility than the one in which it was purchased/leased, it would be a simple matter to maintain a duplicate copy of the stamp signing key or a pointer to the other locale.

Unlike the stamp signing key, the public half of the key exchange key of the electronic postage meters is used for encryption rather than digital signatures. They have much the same locality properties as the stamp signing keys, but their usage is very different: key exchange keys are used to create a secure communication channel between post offices and the postage meters for reload, which is an infrequent operation. The post office accounts computers are the only clients for the public key database, and to perform the reverse authentication (proving the identity of the post office computer to the postage meters) the accounts computer must have access to the private half of the post office public key. This key must be carefully protected by a secure coprocessor from exposure, since an attacker who obtains a copy can easily "sell" postage credit to any electronic postage meter. Similarly, access to accounting computers' secure coprocessors must be controlled, both physically and by password protection. Without both of these access controls, attackers may be able to steal an accounting computer's secure coprocessor and just use it to sell postage credits.

To limit the risk of exposing the post office's private key, we can provide a hierarchy of keys. Every post office can have a separate public/private key pair, and these public/private keys need not be known to every electronic postage meter. The postal service maintains a single pair of *master keys*, the public half of which is known to every electronic postage meter. The master key is used to *certify* the post office keys: the public key of each post office is cryptographically signed using the private half of the master key pair. Thus, when an electronic postage meter contacts a new post office branch, the branch may present it with the digitally signed certificate showing its public key is indeed that of a real post office. This public key may then be cached in the secure coprocessor within the postage meter for future use. By using hierarchical keys, we avoid having to protect a single key that must be kept at every post office — if a particular post office branch's keys were compromised, only the postage meters that purchased postage from that branch is affected.

6 Alternative Approaches

There are several alternative approaches to the problem of providing electronic stamps. One is to simply use encrypted serial numbers as stamps — the post office simply logs them as valid stamps as they are sold. When a piece of mail is marked with that number, the postal service accepts it as valid and strikes the number from its logs. The post office ID may be incorporated as part of the serial number, and the locality property discussed above would still apply. Because the serial numbers are encrypted, malicious users can not guess the next stamp number and will not be able to generate new stamps on their own. Honest users, however, are not protected from the attackers who may look over the honest users' shoulders or otherwise copy the stamp numbers for their own use — as long as the attackers use the copied stamps before the legitimate user does, the attack succeeds. Furthermore, until the honest user actually attempts to use his/her stamps, the crime is undetectable. Without a secure coprocessor to hold the stamps and prevent copying, there is no way to prevent this type of attack.

7 Future work

We have argued that a cryptography-based electronic mail system is feasible, and that we could use a secure coprocessor to ensure the security of such a system against attack. In the future, we hope to explore:

- alternative schemes for protecting electronic stamps,
- methods for evaluating the security of arbitrary cryptographic schemes for protecting electronic franking,
- integration with other electronic currency schemes and tamper-proof architectures,
- and integration with potential post office services, such as integration with electronic commerce.

8 Acknowledgments

We are deeply grateful to Paul Goodman for a number of suggestions on the problem of electronic stamps. We would also like to thank Steve Guattery for his helpful comments.

References

[1] Andrea J. Borr. Transaction monitoring in Encompass (TM): Reliable distributed transaction processing. In *Proceedings of the Very Large Database Conference*, pages 155–165, September 1981.

[2] Dennis K. Branstad, December 1992.

[3] Julius Cæsar. *Cæsar's Gallic Wars*. Scott, Foresman and company, 1935.

[4] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, October 1985.

[5] C. J. Date. *An Introduction to Database Systems Volume 2*. The System Programming Series. Addison-Wesley, Reading, MA, 1983.

[6] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-26(6):644–654, November 1976.

[7] Hans Eberle. A high-speed DES implementation for network applications. Technical Report 90, DEC System Research Center, September 1992.

[8] Jeffrey L. Eppinger, Lily B. Mummert, and Alfred Z. Spector. *Camelot and Avalon: A Distributed Transaction Facility*. Morgan Kaufmann, 1991.

[9] James N. Gray. A transaction model. Technical Report RJ2895, IBM Research Laboratory, San Jose, California, August 1980.

[10] James N. Gray. The transaction concept: Virtues and limitations. In *Proceedings of the Very Large Database Conference*, pages 144–154, September 1981.

[11] Stuart Itkin and Josephine Martell. A PDF417 primer: A guide to understanding second generation bar codes and portable data files. Technical Report Monograph 8, Symbol Technologies, April 1992.

[12] A. Longacre Jr. Stacked bar code symbologies. *Identification J.*, 11(1):12–14, January/February 1989.

[13] R. R. Jueneman. Message authentication codes. *IEEE Communications Magazine*, 23(9):29–40, September 1985.

[14] J. McCrindle. *Smart Cards*. Springer Verlag, 1990.

[15] R. Merkle. A software one way function. Technical report, Xerox PARC, March 1990.

[16] National Institute of Science and Technology. A proposed federal information processing standard for digital signature standard. Technical Report Docket No. 910907-1207, RIN 0693-AA86, National Institute of Science and Technology, 1991.

[17] Elaine R. Palmer. Introduction to Citadel - a secure crypto coprocessor for workstations. Technical Report RC18373, Distributed Security Systems Group, IBM Thomas J. Watson Research Center, September 1992.

[18] R. C. Palmer. *The Bar-Code Book*. Helmers Publishing, 1989.

[19] José Pastor. CRYPTOPOST™: A universal information based franking system for automated mail processing. *U.S.P.S. Advanced Technology Conference Proceedings*, 1990.

[20] Theo Pavlidis, Jerome Swartz, and Ynjiun P. Wang. Fundamentals of bar code information theory. *Computer*, 23(4):74–86, April 1990.

[21] Theo Pavlidis, Jerome Swartz, and Ynjiun P. Wang. Information encoding with two-dimensional bar codes. *Computer*, 24(6):18–28, June 1992.

[22] R. G. Andersen. The destiny of DES. *Datamation*, 33(5), March 1987.

[23] Michael Rabin. Digitized signatures and public-key functions as intractable as factorization. Technical Report MIT/LCS/TR-212, Laboratory for Computer Science, Massachusetts Institute of Technology, January 1979.

[24] R. Rivest and S. Dusse. The MD5 message-digest algorithm. Manuscript, July 1991.

[25] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[26] J. D. Tygar and Bennet S. Yee. Dyad: A system for using physically secure coprocessors. Technical Report CMU-CS-91-140R, Carnegie Mellon University, May 1991.

[27] J. D. Tygar and Bennet S. Yee. Strongbox: A system for self securing programs. In Richard F. Rashid, editor, *CMU Computer Science: 25th Anniversary Commemorative*. Addison-Wesley, 1991.

[28] Steve H. Weingart. Physical security for the μABYSS system. In *Proceedings of the IEEE Computer Society Conference on Security and Privacy*, pages 52–58, 1987.

[29] Steve R. White, Steve H. Weingart, William C. Arnold, and Elaine R. Palmer. Introduction to the Citadel Architecture: Security in Physically Exposed Environments. Technical Report RC16672, Distributed Security Systems Group, IBM Thomas J. Watson Research Center, March 1991. Version 1.3.

[30] Jeannette Wing, Maurice Herlihy, Stewart Clamen, David Detlefs, Karen Kietzke, Richard Lerner, and Su-Yuen Ling. The Avalon language: A tutorial introduction. In Jeffery L. Eppinger, Lily B. Mummert, and Alfred Z. Spector, editors, *Camelot and Avalon: A Distributed Transaction Facility*. Morgan Kaufmann, 1991.

[31] W. A. Wulf, E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, and F. Pollack. Hydra: The kernel of a muiltiprocessor operating system. *Communications of the ACM*, 17(6):337–345, June 1974.