A Secure Architecture for Electronic Check Processing

by

Melanie Moy

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
September 2003

Author.. ....
Department of Electrical Engineering and Computer Science
September 8, 2003

Certified By.................... ....
Amar Gupta
Thesis Supervisor

Accepted By...
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Abstract

Traditional paper checks have been estimated to cost US banks over $2.1 billion dollars a year to process [1]. Much of the cost is due to the time and fees related to human intervention and the physical transport of a check. Gupta and Palacios [15] have defined a web-based check processing mechanism; this paper extends the framework they have developed and defines a cryptographic protocol for the transmission of digital checks using cryptographic building blocks like public key encryption and digital signatures.

Table of Contents

1. Introduction

A check is a fairly simple instrument – a properly signed check represents a one-way transfer of money from one account to another – yet US banks spend over $2.1 billion dollars each year to process checks, and the banking industry estimates that a check is processed, on average, about 2.6 times before making it back to the payer in a monthly statement [1]. A check is first deposited at the recipient's bank, where it runs through an electronic check reader and check sorter. The check writer's bank, bank account, and the check number are encoded on the check in magnetic MICR ink. The check reader and sorter read this MICR ink and sort checks according to the check writer's bank. Employees at bank processing centers manually read in the dollar amount off the check and then print the number on the bottom of the check using MICR ink. This MICR ink can be read easily and is used along with the numbers at the bottom of the check to sort the check at each bank. The check must be returned from the depositor's bank to the check writer's bank so that it may be included in the check writer's next bank statement.

The $2.1 billion spent annually by US banks boils down to $1-5 per check [23]. The majority of the per-check processing cost is due to the expense of physically processing the check, not the fees assessed for the actual electronic funds transfer. ACH fees average about 25 basis points, or 0.25% of the dollar amount of the transaction [24]. For checks under $200, this fee amounts to less than $0.50. There are a number of costs associated with the traditional physical processing of a check:

The cost of reading the check amount and imprinting this amount in MICR ink on the bottom of a check.

The cost of sending the check to the check writer's bank

The check writer's bank must then sort the check and place it into the check writer's monthly statement.

The cost of shipping the checks from the depositor's bank to the check writer's bank totals more than $250 million a year [2]. The system is currently in flux; some banks do not insert the physical check into the check-writer's bank statement, and instead only show an electronic image, but, regardless, there are still significant costs associated with check processing. This thesis, in combination with the Web-Based Check Processing architecture built by Palacios and Gupta [15], details a new method of reducing the per-check processing costs via electronic check transmission.

2. Alternate Money Transfer Systems

There have been viable and secure alternatives to checks for over 25 years now. The section below outlines several alternatives to paper checks and discusses previously related work.

2.1 Paypal

PayPal is one of the most successful online money transfer systems. It ties together credit card accounts and e-mail addresses; anyone with an e-mail address can send money to anyone else. PayPal began as a service used primarily to pay for goods purchased at the popular online shopping side EBay [14].

To send money via PayPal, a user simply registers his e-mail address with the PayPal servers. The servers verify the e-mail address by sending an unencrypted, unsigned e-mail to the user. The e-mail contains a random number that only the PayPal servers and the user know, and this shared secret is used to authenticate the user. Once the e-mail address has been confirmed, the user then connects to PayPal via a secure HTTPS connection and links a credit card number with their e-mail address. If the user wishes to transfer money from their account to another user, they notify the central PayPal computers of the recipient's e-mail address and the amount. If the recipient is not registered with PayPal, they must go through a parallel e-mail address confirmation and credit card linking process. PayPal debits the sender's credit card for the specified amount, and then credits the recipient's card.

PayPal is far from a bulletproof solution to Internet payment processing. PayPal's major weakness is that its own security is predicated upon the security of two of its major building blocks: the security of the Internet e-mail transfer system, and the security of the credit card payment system. If an attacker can intercept a user's e-mail, they can initiate money transfers in that user's name with impunity. It's not a trivial problem to intercept e-mail, but it is far from an intractable problem. PayPal also leverages the existing credit card billing infrastructure. Credit card numbers can be easily stolen, though this is of little concern for PayPal; the credit card companies (i.e. Visa, MasterCard, et al) shoulder much of the burden for any fraudulent credit card transactions.

2.2 CheckFree

A consumer-to-business (recurring) transaction consists of payments that individuals make to commercial establishments on a regular basis; for example, phone bills or electric bills. Currently, Checkfree (www.checkfree.com) is the leading solution to the consumer-to-business problem. Checkfree allows users to access and pay their bills over the Internet free of charge to any one of Checkfree's 260 subscriber companies. The service is offered for free because the subscriber companies pay Checkfree a fee for its services. Checkfree offers a second service which allows users to make payments to anybody, not just the 260 subscriber companies, but this service costs users anywhere from 0 to 15 dollars a month for a set of transactions. The reason for this is that electronic payments are not possible for payees who are not subscribers to the CheckFree service. Instead, a paper check is printed and physically mailed to the intended recipient using the information and payment amount submitted to service provider. The cost of the printing and mailing of the checks is covered by the users' $15 fee. [14]

2.3 Debit Cards

A consumer-to-business (one-time) transaction consists primarily of one-time payments for goods and services; for example groceries or gifts. Previously, bank checks or credit cards were the only options for these types of transactions, but the creation of the debit card provided a better solution [14]. A debit card is similar to a bank check in that money is withdrawn from a user's account, but it does not require any paper or handwriting.

Rather, the debit card is swiped like a credit card, and money is automatically withdrawn from the user's account.

A debit card's security entirely depends on the security of the debit card user's PIN. If anyone has this secret PIN, then they can initiate a money transfer.

2.4 Direct Deposit

A business-to-consumer payment consists of payments made by businesses, usually employers, to individuals. These might include payments for salaries or bonuses [14]. The current solution to the business-to-consumer problem is Direct Deposit. While many employees still receive paper checks for their salaries, the overwhelming majority have chosen to use Direct Deposit to electronically deposit their salary directly into their bank account.

2.5 Singapore's Electronic Checking System

The four solutions above have described excellent alternatives to the paper check, but the reality is that paper checks are still an integral part of our economy today. Despite the various alternatives to paper checks, people are still choosing to use the old fashioned check. Since the exorbitant costs associated with paper a check lie in the processing that requires human intervention and the physical transfer of a check, an ideal solution would involve eliminating these factors [13].

In June 1999 BCS/BCSIS was directed to develop an electronic check processing system for all of Singapore [4]. Singapore is a tiny country--the maximum distance that a check needs to go is 20 miles--but despite its small size, it still suffers from the same logistical problems and substantial costs related to paper checks as a larger country does.

Checks are scanned and MICR data are captured at the receiving bank branch. The information is then transmitted to the Singapore Automatic Clearing House (SACH). The SACH continually reads received checks via OCR and makes successfully scanned checks available for processing almost instantly. Checks that could not be successfully scanned are queued for manual recognition.

3. Previous Work: Gupta and Palacios

Despite the various alternatives to paper checks, people still choose to use the old fashioned check. Since the exorbitant costs associated with paper a check lie in the processing that requires human intervention and the physical transfer of a check, an ideal solution would involve eliminating these factors [13].

Gupta and Palacios [15] have developed a novel prototype that provides a framework for a secure web-based check processing architecture. This framework involves three parties: the check depositor's bank, the central check processing server, and the check-writer's bank (also called the check originator's bank). All communication occurs via HTTPS connections over a web browser to a central server. This section describes Gupta and Palacios's framework. Section 6 of this thesis will describe this thesis' improvements to Gupta and Palacios's framework.

3.1 Check Submission

When a customer deposits a check at a bank branch, it is that bank's responsibility to scan in the check and OCR the check's contents. The following attributes are the most important aspects of the check:

Branch ID

Account number

Check number

Amount

The branch ID, account number, and check number are encoded on the check using well-established MICR technology; the ink is actually magnetically charged, and is easy for machines to read. The OCR software must read both the CAB (courtesy amount block) numeral amount and the LAB (legal amount block) written amount, and reconcile any differences between the CAB and LAB to arrive at a single, correct check amount.

As of 1996, OCR check recognition software could achieve 90-95% accuracy rates [20]. The sensitivity curve of the OCR software is extremely important. Because this is a financial application, it is extremely important that the check amount, as read by OCR software, be accurate. Palacios and Gupta [15] provide extensive background on OCR of check data; this is already a well-researched problem, and check OCR software is believed to be significantly robust that commercial check amount OCR is a viable solution.

If the check can't be automatically read in, it is the depositor's bank's responsibility to manually key in the check data. There will presumably be some cost associated with this electronic data entry. While it is the depositor's bank's responsibility to translate the check from its printed form to an easy-to-transfer electronic form, it is perhaps unreasonable to expect the depositor to shoulder the entire burden of electronically transcribing a check. While the specific details are likely to be a source of contention, it is reasonable for the check writer's bank to split the expense and partially defray the cost of processing the check.

3.2 Clearing a check

According to Palacios and Gupta [15] a representative of each bank must check the centralized check processing system daily for pending checks. The representative must verify the electronic check amount with the check image and then, if the check-writer's bank account contains sufficient funds, approve the check. This clearing process generates a "clear code" that is used as proof of the depositing bank's acceptance of the deposit. This clear code is used as a warranty that the transaction has been approved.

Section 6 of this thesis will further refine this protocol and will cryptographically define the "clear code."

3.3 Obtaining the Clear Code

Branches should check the system at any time to check and see whether the checks they submitted have been cleared for deposit by the bank. If there is a valid "clear code," then the branch can proceed with the instantaneous wire transfer.

3.4 Wire Transfer

The bank that clears the check did not initiate the transfer of funds when it cleared the check; according the mutual agreement between all member banks, a bank only has the obligation to validate its checks within one day. When the check is cleared, the check-writer's bank deducts the money from the customer's account, and then places that money in a holding account where it is held until the depositor's bank initiates the funds transfer process. The check writer's bank then waits for a wire transfer request from the depositor with this "clear code."

4. Goals of A Secure Check System

As stated before, the purpose of this paper is to describe a system by which electronic checks can be securely transferred over the Internet. Before the details of the system design are discussed, it is worthwhile to clearly outline the system goals. This section outlines the high level goals that a robust solution should achieve.

4.1 Encryption

Given the sensitive material that electronic checks contain, there is an obvious need for a secure system by which data can be transferred. The first and most obvious necessity in the system is a method for *encrypting* or *encapsulating* the data. The encryption of data is analogous to a "briefcase passing" model: a check is stored inside a briefcase, but the briefcase can only be locked an unlocked by the sender and the receiver, respectively [21]. The sender writes a check, locks it in a brief case, and passes the briefcase to a carrier who does not know what is inside the briefcase and cannot unlock the briefcase to find out. This carrier passes it to another carrier who passes it to another and so on, until the briefcase finally arrives in the hands of the receiver. The receiver can then unlock the briefcase to obtain his check. Despite the many hands that the briefcase passed through—nobody but the sender and the receiver know what was stored inside.

4.2 Flexibility

Aside from the actual encryption of data, the system should also offer *flexibility*. That is, if changes occur in the check processing route, the system should not require redesign or extensive modification. For example, the Federal Reserve has recently tried to implement changes in the check processing system. Specifically, "the Fed says the new system will allow images of checks to be available to any system users shortly after they've been scanned by the Fed...no matter where the item enters the archive it can be accessed from anywhere in the country" [9]. Presumably more changes will be made in the future due to new legislation or simply due to growth in the checking system, so this system should offer secure methods of data transfer that are applicable even after changes have been made to the routing process.

4.3 Scalability

Another feature that is important to consider is *scalability*. Ideally, this system should work for both small and large checking environments. For example, the system should work for a country with a small geographic area, like Singapore, and for a country that has a large geographic area, like the United States. Along the same lines, a design for a secure checking system should facilitate growing checking systems. If a country's checking system is expanding, the system should provide a simple way to incorporate the new growth to the system.

4.4 Authentication

Finally, as in any secure system, the system should include *authentication*. That is, the system should guarantee that each party is who it claims to be for every communication made during a secure session. For example, if a particular check processing unit wants to pass secure information to its presiding Federal Reserve Board, the system should guarantee the check processing unit that it is indeed talking to the federal reserve board and not some other third party. Similarly, the system should guarantee the Federal Reserve Board that the information being received is indeed coming from the check processing unit and not an unknown third party.

5. Cryptographic Background

Before the encryption scheme is described in detail, some background on encryption is in order. This section defines the terms symmetric key encryption, asymmetric key encryption, hash functions, and digital signing.

First, there are two types of encryption: symmetric and asymmetric. Symmetric key encryption was developed first and differs from asymmetric key encryption because it uses the *same key* to encrypt and decrypt files. Secure applications that require passwords, such as most email applications, use symmetric key encryption. A simple example of symmetric key encryption is: two users Alice and Bob decide on the password "2", and they decide to encrypt messages by changing every letter to be the 2nd letter after the original letter. For example, "apple" would become "crrng" because "c" is the second letter after "a", "r" is the second letter after "p", and so on. To decrypt messages, they simply change every letter in the encrypted message to be the 2nd letter before the encrypted letter (so "c" becomes "a", "r" becomes "p", and so on). This is called *symmetric* key encryption because the same key, "2", is used to encrypt and decrypt messages. Some common algorithms used for symmetric key encryption are DES, triple-DES, and AES.

Conversely, asymmetric key encryption does *not* use the same key to encrypt and decrypt messages. The most common form of asymmetric key encryption is public key encryption. Public key encryption works this way: every user has 2 keys, a public key that anyone can obtain and a private key that the user keeps secret. Any user can send a message to another user, Bob, by encrypting the message with Bob's *public key*. When Bob receives the encrypted message, he decrypts it with his *private key*. This method of encryption is called *asymmetric* key encryption because different keys are used to encrypt and decrypt messages. The defacto algorithm used for public key encryption today is the RSA algorithm.

The different forms of encryption both have advantages and disadvantages. The advantage of using symmetric key encryption is that is much faster than asymmetric key encryption, however it has the disadvantage that users must first find a means of securely exchanging keys. In the example above, Alice and Bob simply decided on the password "2", but how did they correspond that this would be the password? They could have exchanged the password over the phone or by email, but these aren't secure methods of exchange because the phone could have been tapped or the email could have been compromised during transfer. In order to exchange passwords, a secure protocol must be used to exchange keys. Conversely, asymmetric key encryption does not require an additional protocol to exchange keys because key exchange is unnecessary. However, asymmetric key encryption is much slower and is thus rarely used to encrypt any long sessions. Often a combination of the two encryption schemes is used to generate a secure session. For example, two users might use asymmetric key encryption to exchange keys that will be used to symmetrically encrypt a session.

The term *hash function* $H(m)$ is used to describe a transformation that takes in some input message m and returns a fixed-size output $h=H(m)$ called a *hash value*. Cryptographic hash functions have the following properties:

> The input m can be of any length
>
> The output has a fixed length
>
> H(x) is relatively easy to compute given x
>
> H(x) is one-way
>
> H(x) is collision-free

The term *one-way* is used to describe a hash function that is hard to invert. That is, given a hash value h, it is computationally infeasible to find some input m such that $H(m) = h$. The term *collision-free* describes a hash function such that it is computationally infeasible to find any two messages x and y such that $H(x) = H(y)$. Hash functions are generally faster than encryption algorithms, and the hash value of a document is generally smaller than the document itself. These two properties make hash functions an ideal method for message integrity checks and digital signatures. Some of the more common hash functions used are MD2, MD5, and SHA [18].

The term *digital signature* refers to the method of using public and private keys to authenticate that a message has come from a certain person. Signing can be done using public-key signatures (similar to public-key encryption described above). To encrypt messages using public-key encryption, a sender encrypts a message with the receiver's public key and the receiver decrypts the message with the private key. Digital signatures reverse the roles of the keys. That is, to digitally sign a document, the signer encrypts the document with his or her private key, and anyone else can verify the signature by decrypting the document with the public key. Signing a large document using public or private keys can be slow, so documents are generally hashed before they are signed. The signer hashes the documents, encrypts the hash of the document and sends both the plaintext document and the encrypted hash to the receiver. The receiver verifies the signature by decrypting the hash and comparing it to a hash of the plaintext document; if the two are the same, then the receiver has successfully verified the signature. The RSA algorithm is also the de facto method for digital signing [19].

Figures 5-1 and 5-2 illustrate the process of digitally signing and verifying a document.

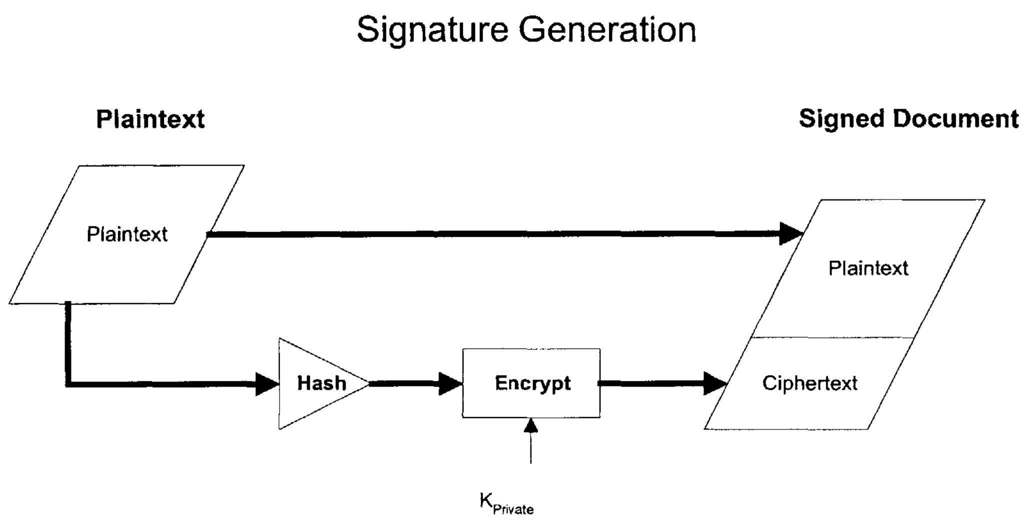


Figure 5-1

Signature Verification

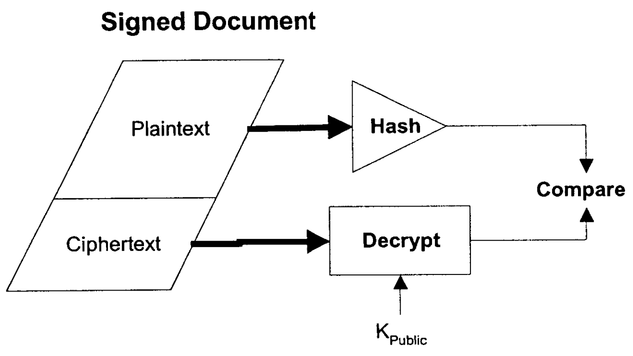


Figure 5-2

6. A Check-Processing Protocol

Gupta and Palacios define a sound framework for secure web check processing, but they leave the precise cryptographic protocols open for future specification. This section defines the specific cryptographic protocol that accomplishes the four design goals defined in Section 4.

6.1 Depositing a Check

Before a check can be entered into the electronic check processing system, its MICR information must be read, and the check amount read in through an OCR process. The Gupta and Palacios paper [15] provides a solid foundation for the this process; in summary, the bank where the check is deposited is responsible for the scanning of the check, the extraction of the check's electronic information, and the check's submission to the central check processing facility.

An electronic check consists of the following six pieces of information:

	Field name	Source	Description
1.	Routing number	MICR	The routing number (also known as ABA Number) uniquely identifies the check-writer's bank.
2.	Account number	MICR	The account number uniquely identifies the customer's account at the bank.

3.	Check number	MICR	The check number uniquely identifies the instrument number for a particular account
4.	Check amount	OCR	The amount is extracted from both the Courtesy Amount Block (CAB) and the Legal Amount Block (LAB). Ostensibly if it cannot be read in via OCR, it must be input by a human.
5.	Image of check front	Scan	The check's image is provided as a reference.
6.	Image of check back	Scan	

These data fields will hereinafter collectively be known as the "check data block" or simply "check data." The depositing bank must upload this check data to the central processing server, and it must generate a digital signature that will later be used to validate the authenticity of the check data.

To generate the digital signature, the depositor' bank simply signs the hash of the check data and a timestamp. Mathematically, this is:

$$S_{depositor} = Sign_{PKdepositor}(Hash(CheckData), timestamp)$$

Section 5 provides an overview of digital signatures and hash functions. One nuance: note that the digital signature is of the hash of the check data. This optimization was made to increase the privacy of the system. Other banks will frequently ask questions such as "was check X" signed? In order to ask this question, the bank must send the central processing system information to identify check X. Rather than send the entire check data structure each time, which contains sensitive information and may be fairly large (because of the scanned check images), the banks can simply send a hash of the check data. It's safe to assume that a hash collision is exceedingly rare, and that a check's hash can be used to uniquely identify each check.

The central check processing system must send back an acknowledgement of the receipt of each check. This acknowledgement takes the form of the server's digital signature of $S_{depositor}$ and a timestamp. Mathematically:

$$S_{receipt} = Sign_{PKserver}(S_{depositor}, timestamp)$$

Depositing a Check

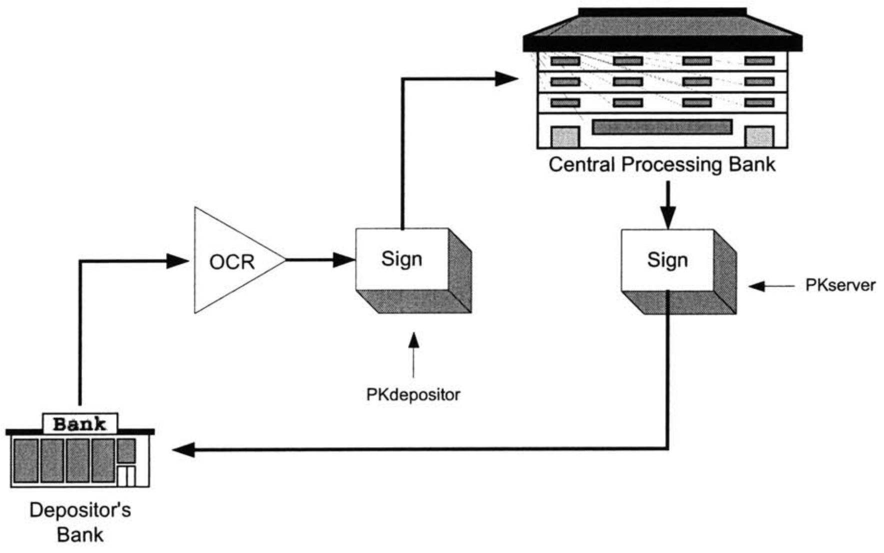


Figure 6-1

6.2 Clearing a Check

Each member bank should check the central processing system at regular intervals and process any pending checks. Gupta and Palacios [15] stipulate that member banks must process any pending checks within one day. The check-writer's bank can either accept the check or decline the check. Gupta and Palacios simply define a "clear code," and leave open the specifics of this clear code. This section cryptographically defines the clear code.

Before the check-writer's bank can accept or decline the check, it must validate the digital signature $S_{depositor}$. To do this, it must first securely obtain $PK_{depositor}$, and verify its authenticity. The exact mechanics of key distribution and authentication are described in Section 7. If $S_{depositor}$ is not a valid digital signature, then the check-writer's bank should decline the check.

This verification step does not prevent physically forged checks from being submitted to the system; i.e. it does not verify the check-writer's intent, it only verifies that the check was deposited at a recognized bank. An unrecognized bank would have an unrecognized $PK_{depositor}$, or a corrupt check would have an incorrect $S_{depositor}$.

To accept a check, the depositor's bank signs the hash of the check data and a timestamp. Mathematically:

$$S_{clearcode} = Sign_{PKoriginator}('accept', S_{depositor}, hash(CheckData), timestamp)$$

A bank might wish to decline a check for any number of reasons:

- The originator (the check writer) might have insufficient funds in his account
- Any aspect of the electronically transcribed data might be incorrect (i.e. the check number or the check amount). The administrative protocols to correct these errors are beyond the scope of this thesis, but, obviously, such errors must be handled by the system.
- If $S_{depositor}$ is not a valid digital signature by $PK_{depositor}$.

To decline a check, the check-writer's bank generates a virtually identical message:

$$S_{decline} = Sign_{PKoriginator}('decline', S_{depositor}, hash(CheckData), timestamp)$$

This clear code, $S_{clearcode}$, is irrefutable proof that the check-writer's bank has accepted the check for further processing. If the bank has generated an accept message, then the protocol dictates that it must honor any wire transfers associated with this check.

Clearing a Check

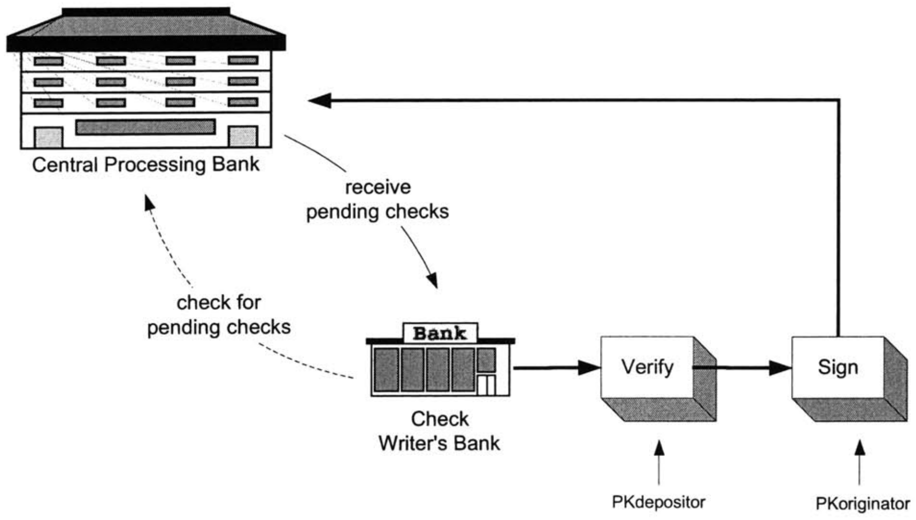


Figure 6-2

6.3 *Obtaining a Clear Code*

The depositing bank must monitor the central check processing system and determine if any of the checks it has posted have cleared, and if they have been cleared, if they have been approved or denied. The check writer's bank is obligated, by the standards set in [15] to process posted checks within one day.

To verify a check's approval or denial, the depositing bank must obtain $PK_{originator}$ and verify that $PK_{originator}$ signed $S_{clearcode}$, and that $S_{depositor}$ and $hash(CheckData)$ both refer to the original check posted by the depositor. If

$S_{clearcode}$ passes all of these tests, then it is a valid accept message, and it can go ahead and originate a wire transfer.

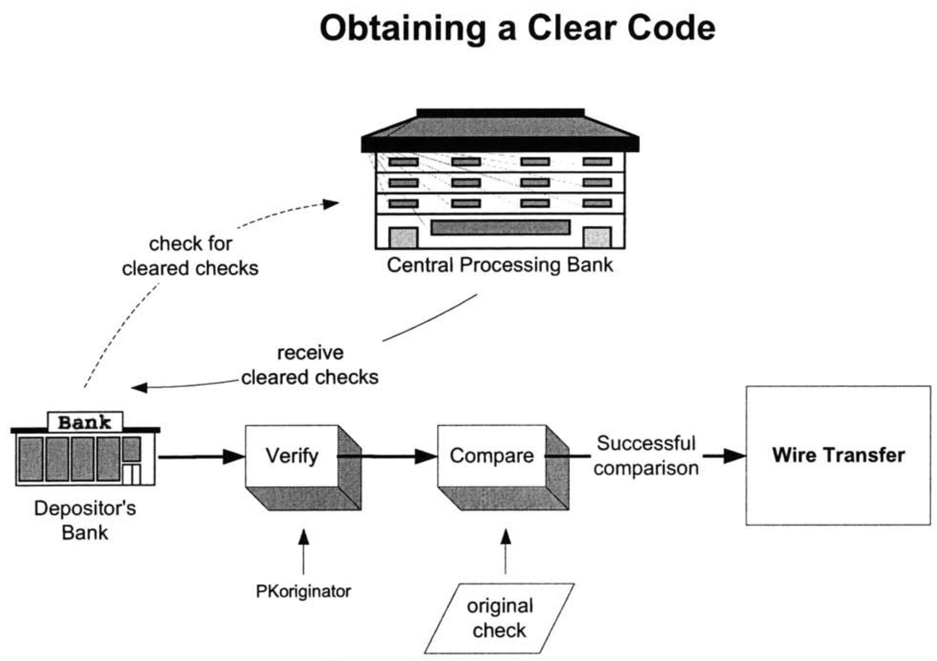


Figure 6-3

6.4 Wire Transfer

To originate a wire transfer, the depositor must send a signed request to the check-writer's bank in the following form:

$$S_{WireTransfer} = Sign_{PKdepositor}(WireTransferRoutingDetails, hash(CheckData))$$

$S_{WireTransfer}$ is a signed message that contains the depositor's wire transfer details. This message must be signed by $PK_{depositor}$; if the message was unsigned, than any fortuitous eavesdropper could send the check-writer's bank a wire transfer initiation message and claim the check-writer's funds.

As described in [15], the check-writer's bank does not initiate the wire transfer upon clearing the check; it waits for the $S_{WireTransfer}$ message from the depositor's bank before proceeding.

The check-writing bank must only initiate one wire-transfer for each check; after a single $S_{WireTransfer}$ is received, it must refuse any additional $S_{WireTransfer}$ messages, even if properly signed, for that specific check.

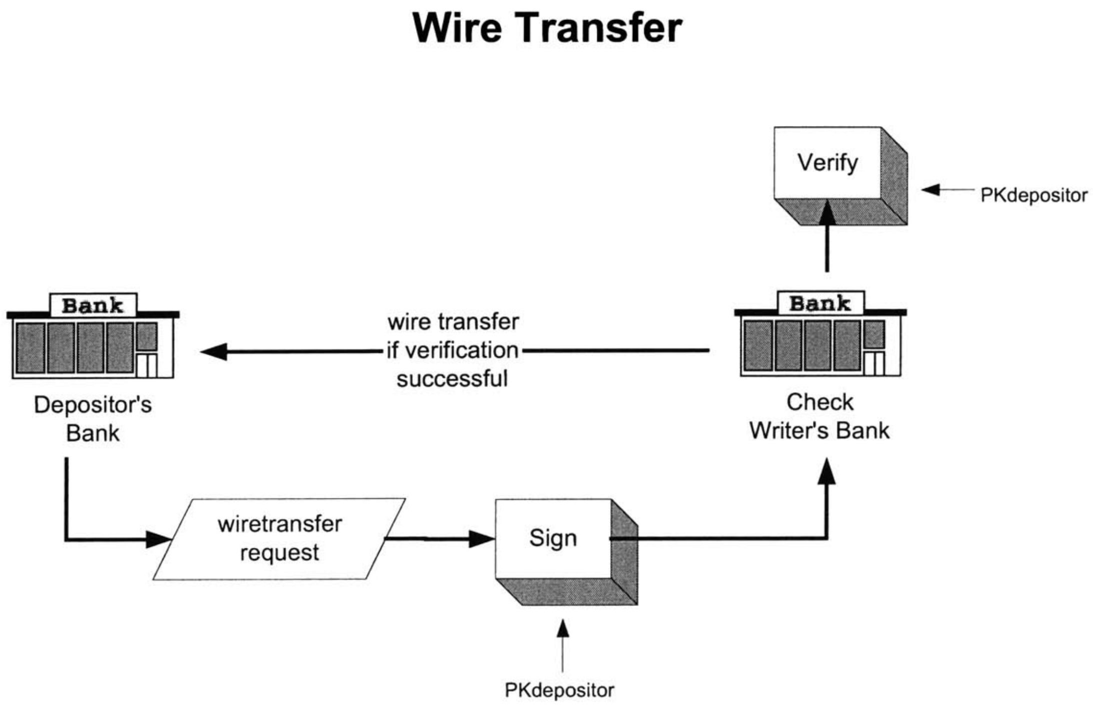


Figure 6-4

7. Authentication Hierarchy

Before any communication is initiated, both parties must be authenticated; that is, it must be verified that both parties are who they claim to be. The authentication system described in this section accomplishes these goals.

7.1 Public Key Infrastructure/Key Distribution

Asymmetric key encryption algorithms such as RSA do a good job of providing secure, authenticated communication between two parties who know each other's public keys, but the encryption and authentication is for naught if you have mistakenly opened up an encrypted connection between yourself and an adversary. Public Key Infrastructures (PKIs) provide mechanisms for the secure distribution of public keys.

This thesis focuses on hierarchical PKIs that terminate at a root Certificate Authority (CA). In such systems, CAs issue certificates binding public keys to identifying information. Hierarchical systems place total trust in the Certificate Authorities; users of the system trust that the CA has performed the necessary due diligence to ensure that a public key that purports to be for "Alice" is actually owned by Alice. The certificates issued by the CAs often take a form dictated by the X.509 v3 protocol. Revocation is performed by CA's using frequently updated Certificate Revocation Lists (CRL's) as defined in Internet RFC 2459 [22].

There are a number of design directions:

A central key server: This design involves a central server that maintains every "individual -> Public Key" mapping in the system. However, this is infeasible because a central authority cannot possibly perform adequate due diligence for millions of users.

A PGP-like Web of Trust: This design involves a decentralized method by which users are authenticated by their peers. For example, if Alice and Bob trust each other, and Bob and Charlie trust each other, then Alice and Charlie can trust each other because they have been deemed "trustworthy" by their mutual friend Bob. A decentralized system has numerous benefits, but a web of trust provides only a "fuzzy" guarantee of correspondence between a specific individual and a specific public key. Consider the situation where you must accept the public key of someone for whom there are 6 degrees of separation—can they really be trusted? In this case, where banks are transferring millions of dollars at a time, a fuzzy assurance of authentication is not good enough.

A hierarchical PKI: In this system, a central CA certifies individual organizations, but each individual organization is responsible for certifying its own users. For example, your group certifies that you are an employee, the department certifies that your group is valid, the company certifies that the department is legitimate, and so on until you reach the Certificate Authority. The check processing system uses such a hierarchical PKI.

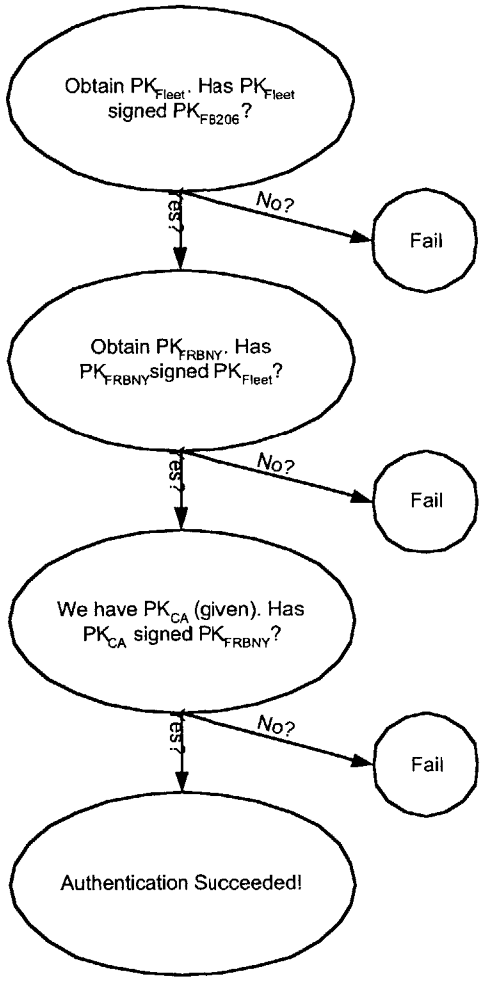
7.2 Authentication Hierarchy

This hierarchical system greatly reduces the burden on any central CA. Rather than certifying the identity of every individual user of the system, the central CA only has to certify the validity of organizations. It must perform the necessary due diligence to ensure that the user who purports to be "Federal Reserve Bank of New York" is in fact the Federal Reserve Bank of New York (FRBNY), but it only needs to do this due diligence once for the entire organization. When it certifies FRBNY's public key, it gives FRBNY complete power over the Second Federal Reserve District for which it has supervisory jurisdiction over [11].

FRBNY can then divvy up its domain into smaller components such as regional offices. FRBNY has supervisory jurisdiction over all of New York State, the 12 northern counties of New Jersey, Fairfield County in Connecticut, Puerto Rico and the Virgin Islands [11]; maintaining a central key distribution server that must authenticate each bank in these areas would be infeasible. Instead, FRBNY can use hierarchy to simplify the task--it could, for example, further subdivide by regional offices or bank corporations, such as Fleet or Chase Manhattan. Each of these can in turn subdivide into individual banks and so on. The key strategy is to use hierarchy to reduce the administrative burden of certifying an individual user to a manageable problem.

The hierarchical protocol outlined in Figure 7-1 below details the certification process required to prove that a given public key, PK_{FB}, is the correct public key for Fleet Bank #206 in Boston, MA.

Figure 7-1: How to Authenicate Fleet Bank 206

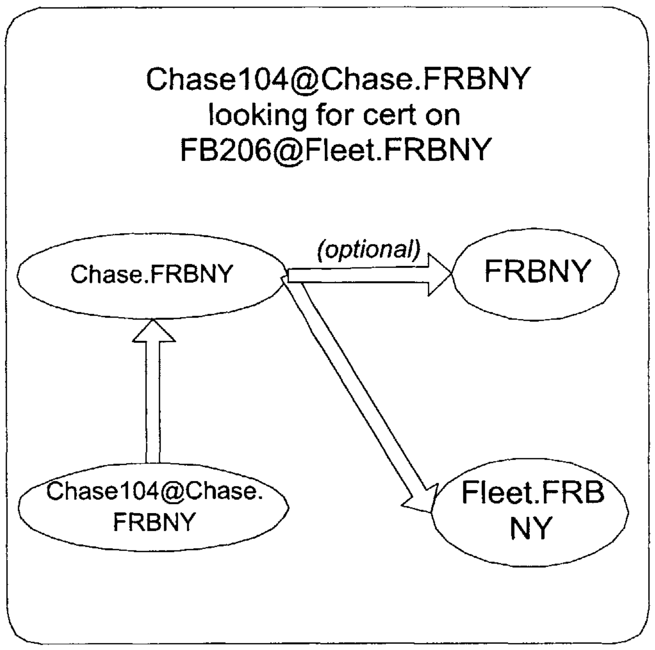

Obtain PK_{Fleet}. Has PK_{Fleet} signed PK_{FB206}?
Yes?
No?
Fail
Obtain PK_{FRBNY}. Has PK_{FRBNY} signed PK_{Fleet}?
Yes?
No?
Fail
We have PK_{CA} (given). Has PK_{CA} signed PK_{FRBNY}?
Yes?
No?
Fail
Authentication Succeeded!

.7.3 *Key Lookup*

The hierarchical design of the system also lends itself to an efficient key-lookup mechanism. A hierarchical, caching key lookup system distributes key lookup traffic. For sake of ease, the notation FB206@Fleet.FRBNY is used to indicate that Fleet Bank #206 (FB206) falls under the domain of Fleet, which falls under the domain of the FRBNY.

This notation is similar to email addresses used at corporations that are divided by groups or departments. This key lookup scheme is very similar to the Domain Name Server (DNS) name resolution scheme and this notation is, in fact, a very useful way to incorporate the key lookup scheme into currently existing technology. An example of this lookup scheme is shown in figure 7-2 below.

Figure 7-2: PKI lookup



Every public key is stored at a primary server that corresponds to the domain under which a user falls. For example, FRBNY.Fleet could be the primary key server for Fleet Bank #206 because it falls under the domain of Fleet. Similarly, FRBNY could be the primary server for Fleet because it falls under the domain of the Federal Reserve Bank of New York.

When a user in the system wants to send data to FB206@Fleet.FRBNY, it queries its local key server. When a key lookup server receives a request for a username's public key, it:

1. Tries to fulfill the request using its database of user keys (i.e. if it is the primary server for a given public key, then it can fulfill the request immediately).
2. Tries to fulfill the request using its cache of 3rd party username -> $PK_{username}$ mappings.
3. Otherwise, it queries the primary server for the username (i.e. Fleet.FRBNY), retrieves the public key, stores it in its cache, and passes the public key on to the user. A key lookup server can optionally retrieve the public keys for the rest of the servers in the lookup hierarchy, as they are needed in the key authentication process. A key server may wish to perform authentication itself to prevent corrupt entries from being stored in its cache.

CRLs are kept by the primary server for a username, and a key server is only allowed to revoke keys that are in its namespace.

This trust model assumes that the public keys for the root CA servers are stored in the software and are correct. If an adversary is able to tamper with the root public keys in a software distribution, then the adversary can undermine the entire PKI infrastructure.

7.4 Goals Achieved

The authentication scheme described above helps one achieve three of the system's goals: authentication, flexibility, and scalability.

Authentication for users is achieved because each user in the system can be authenticated by a user that is one level higher in the hierarchy, and eventually a user will be authenticated by the Certificate Authority who can be supremely trusted.

The scheme also offers flexibility because it allows the system to change without requiring drastic changes in authentication. Take for example, the merging of Bank Boston and Fleet Bank: in the defined authentication scheme, a Bank Boston #54 would have been identified as BB54@BankBoston.FRBNY. When it merges with Fleet Bank and becomes Fleet Banks 307th bank, it can simply change its identifier to FB307@Fleet.FRBNY. This means that the Bank is now authenticated by Fleet rather than Bank Boston. However, the Federal Reserve Bank of New York has to make no changes—only Fleet is required to add new branches to its domain.

Finally, the hierarchical authentication allows a great deal of scalability. As in the previous paragraph, only the bank needed to do minor work to add a new branch to the system. Even if a new bank, Example Bank, were to open up 25 branches in the state of New York, the Federal Reserve Bank of New York would only need to add one new user to its domain, and the Example Bank would only need to add 25. If every user in the hierarchy were to authenticate 100 users in the level below, a relatively low number, and the hierarchy were only 4 levels deep, 100 million users could be authenticated! Considering that users are not individual people, but bank branches or banks, there is more than enough room for growth in the system.

8. Protocol Attacks

The following section describes a number of attacks on the secure check protocol defined in Section 6.

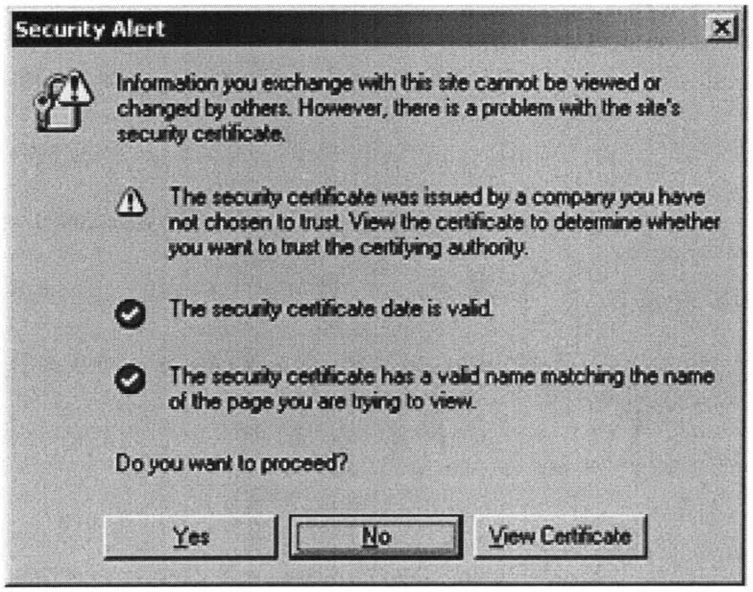
8.1 *Domain Name Spoofing Attack*

This attack allows a clever adversary to convince users of web browsers at the check-writer's bank or the depositor's bank that they are the central check-processing server.

The DNS system [16] is a hierarchical caching system whose sole job is to resolve textual domain names such as 'mit.edu' into IP addresses such as '18.7.22.69'. There are sets of high-level DNS servers that contain information about every host on the Internet, but to prevent these servers from being overwhelmed, this information is propagated down to lower-level servers. Typically, Internet Service Providers (ISPs) maintain domain name servers for their clients. If the ISP's domain name server does not know about a specific name, then the domain name server asks the root servers for information on the domain. If possible, information is provided from a local cache.

According to Gupta and Palacios [15], member banks will access a centralized web site via a web browser. The Internet Service Providers for the member banks will have the job of resolving the domain name of the central check processing server (i.e. "https://webcheck.bank.com") into an IP address ("1.2.3.4"). If an attacker were to hijack the ISP's domain name servers, he could redirect all accesses to "https://webcheck.bank.com" to his own central check processing server (IP address "9.9.9.9").

The hijacker will not be able to obtain a valid web certificate that shows that his new server, "9.9.9.9" is "https://webcheck.bank.com," so a privacy warning dialog will come up. If a bank user clicks "Yes" on this dialog (Figure 8-1), he can immediately redirect the bank user to a similarly named web server owned by himself – for example, "https://webcheck.banks.com" (note the additional "s" on "bank.com"). Because the attacker owns banks.com, then he can obtain a valid web certificate for banks.com and the web browser will not alert the user of any errors.



Information you exchange with this site cannot be viewed or changed by others. However, there is a problem with the site's security certificate.

The security certificate was issued by a company you have not chosen to trust. View the certificate to determine whether you want to trust the certifying authority.

The security certificate date is valid.

The security certificate has a valid name matching the name of the page you are trying to view.

Do you want to proceed?

Figure 8-1

At this point in time, the clerk at the member bank believes that the attacker's central check processing server is the correct central check processing server. The next section,

Section 8.2, discusses the effects of a rogue central server on the system. Web browsers are, by default, configured to accept the SSL certificate of any web server if the SSL certificate is signed by one of a select few recognized root level Certificate Authorities (CAs) such as Verisign or Thawte. Verisign and Thawte will grant a SSL certificate to anyone who can prove that they own the domain name in question – i.e. an attacker would have no problem obtaining a SSL certificate for https://webcheck.banks.com. The solution is to specially configure the web browsers of the member bank computers to only recognize certificates signed by a special bank-only root CA.

8.2 Rogue Central Server

Section 8.1 described how an attacker could gain control of the central check processing server. This section describes how an attacker who has gained control of the central server, regardless of the specific method used to gain control, could disrupt the functioning of the system.

8.2.1 Birthday Paradox: Attack on the Hashes

A rogue central server could, by transferring money between accounts at different member banks, accumulate money. This attack is statistically difficult, but theoretically possible.

In this system, checks are identified by their hashes – i.e. *hash(CheckData)* is assumed to uniquely identify a check. Because hash functions are difficult to invert (see Section 5), given a hash function that produces an *n*-bit output, one would expect to need, on average, 2^{n-1} trials to find a new hash value, *hash(CheckData2)=hash(CheckData)*. SHA1, a popular hash function widely considered to be secure, produces a 160-bit output. 2^{159} is roughly 7.3 x 10^{47}; it is clearly computationally intractable to find a hash collision.

However, an attacker can exploit the birthday paradox. While it might be very difficult to, given a specific *hash(CheckData)*, find another *hash(CheckData2)=hash(CheckData)*, it's significantly easier to find any two hashes *CD1* and *CD2* such that *hash(CD1)=hash(CD2)*. This is the heart of the birthday paradox; you can find a hash collision in about *sqrt*(2^n), or $2^{n/2}$, rather than in 2^{n-1}. For SHA1, this is 2^{80}, or about 1.2 x 10^{24}. Even with the birthday paradox, finding hash collisions is not computationally tractable. Assuming that an adversary has at its disposal 1,000 computers each capable of performing 1 billion hash comparisons per second, it would take about 33,000 years to find a hash collision.

Let's assume that there is an exponential increase in computational power and that it's tractable to find two hashes *CD1* and *CD2* such that *hash(CD1)=hash(CD2)*. CD1 and CD2 represent two checks draw on the same account – the attacker's account – but with different amounts, say X_1 and X_2, respectively. Let's say that $X_1 > X_2$.

A rogue central server can deposit check CD1 with value X_1 in an account it owns at a member bank. When the depositing bank forwards the central server CD1, then the check processing server can replace CD1 with CD2 (because *hash(CD1)=hash(CD2)*). The check-writing server will then deduct X_2 from the attacker's account, but the depositing bank will add X_1 to the attacker's account. The attacker has made a profit of $X_1 - X_2$.

8.2.2 Receipt Forgery

A receipt forgery is not a serious attack. Because the attacker has control of the central server, he could forge the timestamps on the receipts, $S_{receipt} = Sign_{PKserver}(S_{depositor}, timestamp)$. However, this has no monetary impact on the system; these receipts are only used to verify the depositor's submission and to enforce the constraint that member banks must either accept or deny checks in one day.

8.3 Rogue Check-Writer's Bank

The check-writer's bank is only debited money; there is little opportunity for it to profit by taking advantage of the protocol. In fact, it has the most to gain by enforcing the protocol well; it is in the check-writing bank's interest to only send money to valid depositors.

8.4 Rogue Depositors

8.4.1 Traditional Check Forgery

A rogue depositor can forge checks and deposit these in its accounts. However, this is little different from traditional check forgery. This thesis does not address the issue of check forgery; i.e. how to prevent checks from being written without the check-writer's authorization. The web-based check processing system described in this thesis does not make it more or less difficult for traditional check forgery to occur.

8.4.2 Replay Attack

A rogue depositor can replay wire transfer messages, $S_{WireTransfer}$, but the check writer's bank should ensure that it only sends money if the following conditions are met:

- The wire transfer message is from the appropriate member bank – the member bank which initiated the deposit
- At most one successful wire transfer will be sent to the depositor's bank

This prevents a replay attack:

- If a rogue depositor forges a $S_{WireTransfer}$, the check writer's bank will refuse the wire transfer request because the bank requesting the wire transfer does not match the bank that initiated the check deposit
- If an attacker observes the depositor's valid $S_{WireTransfer}$ message and replays this message, the check writer's bank will take no action because it has already initiated a wire transfer for this instrument.

9. System Integration

Given the flexibility and scalability of the system, the electronic checking system above can be easily integrated into any currently existing banking system. Whether simple or complex, the design described above can accommodate for the banking system's current checking route. The design above allows any user to securely pass information onto any other user, so that each stop on a check's route can be made a user in the system. Rather than mailing the physical check to the next stop on the check's route, the user can send the check over the Internet using the scheme described above.

A check that enters the iCheck system might follow such a route:

A user deposits a check at a bank. Checks will be scanned at a particular physical location depending on the checking system. Some possibilities include:

> **Branch-based check processing**: Checks would not have to be shipped anywhere to be scanned, but every bank branch would need to have access to a scanner. Additionally, every bank is legally obligated to store the checks for 7 years so the bank branch would be required to have the storage space to warehouse the physical checks [1].
>
> **Regional processing**: Regional processing is an intermediate solution that requires less shipping than a national scanning architecture, and requires fewer scanners than a branch-based scanning scheme.

National processing: A national processing scheme would require a central check processing location where every check would need to be shipped, but you get the best economies of scale.

ATM-based processing: An ATM-based scheme would require payees to scan their checks in at an ATM. This allows the payee, who has the strongest interest in seeing his or her check get processed quickly and correctly, verify that a check has been properly scanned in. Once the check has been scanned, the electronic picture can be read using OCR at some other location. It is important to determine the percentage of checks that are processed via ATMs to judge the usefulness of this scheme.

Once a check has been scanned and read it becomes an e-check. The bank must store it electronically and credit its customer's account with the appropriate amount. The e-check is then electronically sent to the depositing bank using the encryption scheme described in section 4. The receiving bank must be a user in the encryption system in order to properly receive the e-check.

Whether the bank branch sends the e-check directly to the receiver's bank or to one of its central servers is unimportant because of the flexibility the system offers. In fact, the bank branch can send the information to both the receiver's bank and to its central server. The added advantage of sending e-checks is that the check can be duplicated and sent to two different recipients, whereas a physical check cannot be duplicated and must be sent to recipients one at a time. The ability to duplicate a check helps to save time, and the reduced cost of sending it over the Internet helps to save money.

When the receiver's bank obtains the e-check, it can verify the signature via the authentication scheme described in Section 7. Once a signature has been verified, the bank then stores the encrypted check and debits the money from their customer's account. The check will be noted on the customers' next bank statements.

The scenario described above requires certain new technologies to be integrated at each location that sends or receives a check. Every location receiving a paper check to be scanned would need to obtain scanning hardware and OCR software. Any facility that sends or receives an e-check needs to obtain public keys and needs to install encryption software onto their computers. Facilities that store digital archives of checks would need to purchase hardware to store this data. And finally, banks receiving e-checks need to set up a system to receive the checks and to automatically add the check onto the customer's periodic bank statement.

10. Bibliography

[1] "Banks hope to end paper chase." CNet News.com. October 2002. http://news.com.com/2100-1017-960911.html

[2] Rui Tang. May 2003. http://mit.edu/dewdrop/Public/international_checks/vision_paper.doc

[3] "Section-by-Section Analysis." The Federal Reserve Board. March 2002. http://www.federalreserve.gov/paymentsystems/truncation/proposed.htm

[4] "Cheque Truncation System." BCS Information Systems Private Limited. http://www.bcsis.com/products9.html

[5] http://web.mit.edu/tedlow/profit/upload/

[6] "Improving the Processing of Handwritten Bank Checks." Winnie Chan. May 2002. http://profit.mit.edu/PDFS/WinnieChan.pdf

[7] "Character Segmentation Heuristics for Check Amount Verification." Salman Amin Khan. June 1998. http://profit.mit.edu/PDFS/salkhanthesis.pdf

[8] "QSC (Quick Start Control): A Flexible Control System." Conix Systems. May 2003. http://www.conixsystems.com/products/qsc.shtml

[9] "Fed To Launch New Electronic Check Processing System." Dow Jones Newswires. May 2002. http://mitsloan.mit.edu/news/releases/2002/gupta.html

[10] "Frequently Asked Questions." The Federal Reserve Board. May 2003. http://www.federalreserve.gov/faq.htm

[11] "Introduction to the New York Fed." The Federal Reserve Bank of New York. May 2003. http://www.ny.frb.org/introduce/?expand=1

[12] http://www.soi.wide.ad.jp/class/20000009/slides/08/5.html

[13] "iCheck: An Architecture for Secure Transactions in Processing Bank Checks." Joe Figuero. June 1997.

[14] "Contemporary Bank Check Alternatives." Adam Garner. May 2003.

[15] "Description of Web-Based Check Processing." A. Gupta, R. Palacios. http://www.iit.upco.es/palacios/web_check_processing/intro.html

[16] "RFC 921: Domain Name System Implementation Schedule." Postel, J. http://www.zoneedit.com/doc/rfc/rfc921.txt. October 1984

[17] "The Birthday Paradox" Jenkins, Bob. http://burtleburtle.net/bob/hash/birthday.html

[18] "What is a Hash Function?" RSA Laboratories. http://www.rsasecurity.com/rsalabs/faq/2-1-6.html. August 2003

[19] "Digital Signatures". http://www5conf.inria.fr/fich_html/slides/tutorials/T1/background/RESCUE3.HTM. August 2003

[20] "A Multi-Layered Corroboration-Based Check Reader." Shridar, M. http://www.alumni.caltech.edu/~dave/dasbook/dasbook.html

[21] Schneier, B. "Applied Cryptography: Protocols, Algorithms, and Source code in C, Second Edition". John Wiley & Sons.

[22] "RFC 2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile." Housley, R., et. al. January 1999. http://www.ietf.org/rfc/rfc2459.txt

[23] Gupta, Amar. "Technology – Check Processing." LokVani.com. February 2003.

[24] "Risk to Financial Institutions Using the Automated Clearing House for Large Payments." http://www.nacha.org/OtherResources/riskmgmt/FIRisk.pdf