Available online at www.sciencedirect.com

ScienceDirect

Computer Networks 51 (2007) 751–766

Computer Networks

www.elsevier.com/locate/comnet

Portable security transaction protocol

Glenn Benson *

JPMorgan Chase & Co, Treasury Services, 900 Chelmsford Street, Floor 10, Lowell, MA 01851, United States

Received 28 July 2005; received in revised form 11 June 2006; accepted 13 June 2006
Available online 7 July 2006

Responsible Editor: D. Frincke

Abstract

The Portal Security Transaction Protocol (PSTP) is a new signature technology that adds signature semantics to one-time password technology. PSTP was developed to secure transactions in the financial services industry; however, PSTP may be applicable to signatures in other spaces. PSTP technology provides high signature strength of mechanism without requiring asymmetric key pairs deployed to client machines. PSTP provides cryptographic after-the-fact evidence of a transaction event in a secured log.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Security; Authentication; Signature; One-time password; PKI; Financial institution; Bank; Non-repudiation; Key management; Security protocols; Data integrity; Entropy; Consequential evidence

1. Introduction

JPMorgan Chase Treasury Services is the largest processor of electronic funds globally. On a daily basis JPMorgan Chase Treasury Services processes on average more than USD 3 trillion in its wholesale operations. For the past five years, JPMorgan Chase used Public Key Infrastructure (PKI) [1] signature technology extensively to secure value-bearing transactions. Deficiencies with PKI-based technology impelled JPMorgan Chase to invent a new kind of signature technology called the Portable Security Transaction Protocol (PSTP) [2]. JPMorgan Chase migrated to PSTP its Treasury Services customers who interact via a web browser in order to transfer funds.

PSTP was developed to secure interactive (browser-based) transactions in the financial services industry; however, the technology may apply to transactions in other industries such as health care, insurance, or legal services. The following on-line banking scenario presents an example use case. A user fills out a web-based form deployed to his or her browser. On the form, the user indicates an origin account, destination account, and monetary transaction amount. The user enters authentication credentials and presses a button marked "signature". Next, the user uploads the form to the server. Upon receipt, the server validates the signature before processing the transaction.

PSTP permits both the enterprise and the users to employ the type of authentication credential that

* Tel.: +1 978 805 1046.
E-mail address: glenn.benson@jpmchase.com

best fits their respective needs. In general, market acceptance of asymmetric key pairs deployed to *servers* is good; and acceptance of asymmetric key pairs deployed to *clients* is poor. SSL and TLS [3], for example, enjoy wide-spread usage throughout the Internet when they use asymmetric key pairs deployed to the enterprise's web servers; however, few users install asymmetric key pairs on their browser.

This difference in PKI acceptance when comparing server and clients is not accidental. From the enterprise's perspective, the web servers reside in locked data centers; and a dedicated staff manages the servers. The enterprise manages service interruptions through server redundancy and disaster recovery. The enterprise support staff considers maintenance of security technology to be within the realm of their job descriptions. In contrast, users do not want to be locked to a single machine. If the user's machine fails, or if the user travels, then the user wants to simply open a browser on a different machine. The user's job description does not normally include maintenance of security technology, and as a result the user is not willing to invest time and resources.

Depending upon the relative security of the media used to secure private keying material, a PKI has two deployment choices. Unfortunately, neither choice is a good fit for the users. In the case of private keying material stored in non-secured media, (e.g., a file), the relative strength of the security mechanism is weak. Any intruder who obtains access to the non-secured media could potentially obtain a copy of the private keying material without the legitimate owner's knowledge. Therefore, this deployment choice incurs the relatively high cost and overhead of PKI, without enjoying enough of the security benefits.

On the other hand, secured media for asymmetric key pair-based hard tokens adequately addresses many of the security issues, while raising ergonomic concerns. When a user's machine is not available, smart card [4] technology does not work unless the user can find another smart card-enabled machine. Dongles [5] and USB tokens [6] may be more portable; however, few users would be willing to correctly apply security best practices by unplugging the devices from the machines during periods of inactivity. Furthermore, despite universally recognized USB standards, smart card readers, Dongles, and USB tokens require special installation steps, and may potentially raise device conflicts. Suppose a cash manager's company suffers penalties if the company does not make payments by the end of the day. Unfortunately, on one particular day, the cash manager has the bad luck to find that his or her disk drive crashes. When the cash manager inserts a smart card, dongle or USB token in a new machine, nothing happens because the new machine does not know how to invoke the cryptographic capabilities of the new device. Since the cash manager is not necessarily skilled in computer maintenance, he or she calls the corporate help desk looking for a solution. Hopefully, the help desk operator fixes the machine before the cash manager suffers financial penalties for the late payments.

The National Institute of Standards and Technology (NIST) Electronic Authentication Guideline [7] defines four levels of authentication, each with increasing levels of security. The lowest two levels are levels one and two, and they communicate passwords through various channels. Although financial regulations do not explicitly reference the NIST classifications, one may compare to see that levels one and two are insufficient for use in Internet banking [8,9]. Financial regulators normally consider the equivalent of level three to be the minimum permissible level. At level three, one may use any of the following three types of tokens: (i) soft tokens that contain a shared secret encrypted by a password or symmetric key, (ii) hard tokens that require activation using a password or biometric, and (iii) One-Time Password (OTP) device tokens. For an OTP device token, "authentication depends on a symmetric key stored on a personal hardware device that is a cryptographic module... The device combines a nonce with a cryptographic key to produce an output that is sent to the verifier as a password. The password shall be used only once and is cryptographically generated; therefore it needs no additional eavesdropper protection" [7]. Each OTP device has a unique cryptographic key, and the server gets a confidential copy of this same cryptographic key. Market examples of OTP device tokens are the SecurID [10] and Vasco [11] tokens. Market acceptance for OTP devices in the financial services industry is growing, e.g. [12,13]. Also, the Financial Services Technology Consortium's recent Better Mutual Authentication Project included the goal of improving the adoption of OTP technology [14]. A time-based OTP device token, e.g., SecurID, relies upon a synchronized clock shared between the client's OTP device token and the server. At fixed periodic intervals, e.g., once per minute, the client derives a nonce from the current time, and computes an OTP value by cryptographically combining the nonce with the key. In the same interval, the server expects the client to provide an OTP value derived from the same time and cryptographic key. Generally, OTP device vendors build proprietary methods to protect against clock drift. In addition to validating the correct OTP value, the server normally implements a scheme which protects against OTP value guessing. If the server receives too many OTP value guesses in a short period, then the server temporarily suspends the client's ability to request authentication events.

A properly deployed OTP device token requires at least two authentication factors. The first factor, the OTP value, is a *transient credential* (TC) which is "something you have (for example, ID badge or a cryptographic key)" [7]. The second factor, is a *static credential* (SC) which is "something you know (for example, a password)" [7]. A user does not properly authenticate unless he or she provides both a correct transient and static credential.

At level four, NIST's highest level, NIST permits asymmetric key pair based hard tokens, but not soft tokens and OTP devices. The issue with a soft token is the lack of copy-protected media. If a user has a soft token on his or her machine, the user does not know whether an attacker possesses an illicit copy. An OTP device token suffers a different deficiency. "Like hard tokens, one-time password device tokens have the security advantage that the token is a tangible, physical object. Subscribers should know if their token is stolen, and the key is not vulnerable to network, shoulder-surfing, or keyboard sniffer attacks. Unlike soft tokens or hard tokens, a session key is not created from the authentication process to authenticate subsequent data transfers" [7]. In other words, OTP device tokens have the advantage of copy-protected storage media, but the disadvantage is that the authentication material does not cryptographically bind to data. PSTP addresses this OTP device deficiency. As a result, this paper argues that PSTP should elevate the security of an OTP device beyond soft tokens up to a level comparable with an asymmetric key pair-based hard token.

PSTP is not the only solution which cryptographically binds transient credential to data. However, other solutions are either cryptographically deficient, or ergonomically impractical. A cryptographic calculator (e.g. [11]) is a handheld OTP device token that accepts input, and produces a checksum based upon the input and the transient credential. The user must type the input into the cryptographic calculator, and then read the checksum shown on a display. The user then copies the checksum from the cryptographic calculator onto a form on the user's computer. Ergonomic considerations limit the number of characters that the user may copy from the browser into the cryptographic calculator, and the number of characters in the checksum. Users are not willing to type a long list of transactions into the calculator; and they are also not willing to copy the value of a long checksum off the calculator onto the computer. These ergonomic considerations prohibit the use of true cryptographic message digests on either the calculator's input or output. Therefore, although the cryptographic calculator may provide some level of security assurance, the calculator cannot provide the same high level of assurance that one would expect at NIST authentication level four.

PSTP is a drop-in replacement for a solution that provides asymmetric key pair based signatures. This means that the application may invoke a signature package through a common API that exports asymmetric key pair and PSTP signature syntax. One may configure the package to provide either type of signature without significantly changing the invoking application's software on either the signer (client) or the verifier (server).

The list below summarizes signing requirements shared by both PSTP-based and client-side asymmetric key pair-based signatures:

- *Message authentication*: Using cryptographic means, authenticate the originating user of each transaction, while binding the user to the transaction's data.
- *Message integrity*: Using keyed cryptographic means, ensure that a transaction does not change in-transit. A transaction recipient must receive cryptographic assurance that the received transaction is bit-for-bit identical to the transaction that left the originating user's machine.
- *Replay protection*: Detect, and reject any occurrence of an unauthorized replay.
- *Consequential evidence*: Provide cryptographic after-the-fact evidence of a transaction event in a secured transaction log coupled with a method for validating the after-the-fact evidence. The purpose of the validation is to justify the claim that at the time of the transaction, the message was properly authenticated, inherently possessed message integrity, and was not an unauthorized replay.

- *Entropy and algorithms*: Observe the Uniform Commercial Code UCC4A [15] which mandates "commercially reasonable" security, by enforcing good security practices. In the case of cryptography, employ proper entropy and algorithms. Comply with US and international standards (e.g. [16–21]), and commercially reasonable best practices (e.g. [7,16,22,23]).
- *Secured media*: Ensure that the mechanism permits multifactor authentication [7]. Observe regulatory guidance for strong authentication [8,9] by leveraging the NIST ratings for level three or level four authentication methods.

A signature is not a technology that addresses all security needs. The list below presents some important security issues that are outside the scope of both PSTP and PKI signatures. Normally, applications address these concerns by deploying signatures in concert with other security technology:

- *Secrecy*: If an application requires secrecy, then the application may encrypt the communication link using cryptographic protocols such as TLS or IPSec [24]. Alternatively, the application may implement application-layer secrecy through protocols such as WS-Security [25].
- *Anti-phishing*: Phishing is an attack in which an attacker fools a user into connecting to the wrong site. An application best protects against phishing by locating the mitigating control at the first point of contact between the user and the application. Therefore, most applications implement anti-phishing controls within the context of the login sequence, and as a result, anti-phishing controls are out of scope of the signing technology. Nevertheless, OTP technology thwarts many of the most common phishing attacks. If the user enters an OTP into a malware-infected machine, then the machine may potentially use the OTP in an unintended manner. However, the machine cannot solicit future OTP values from an OTP device token without the user's knowledge.
- *Receipts*: An application may potentially employ signing technology as the vehicle for securing transaction receipts. However, a receipt is a particular use of signing technology; and is not an inherent aspect of the technology itself.
- *Denial of service protection*: An application may use firewalls, intrusion detection, and intrusion prevention to protect against denial of service attacks. These controls are important to an application's security, but are not the responsibility of the signature technology.

This paper provides a comparison of the terms *consequential evidence* and *non-repudiation*. This paper has the position that the term *non-repudiation* has meaning which should have remained insular within the cryptographic community. However, common practices mistakenly apply the term *non-repudiation* to a greater context, where the cryptographic meaning of non-repudiation has little importance from either a legal or business perspective [26]. The binary concept of cryptographic non-repudiation, for example, ignores the crucial issues associated with the media that holds the confidential keying material. In contrast, as opposed to a binary property, consequential evidence is a control with an inherent strength of mechanism metric. This metric takes into account cryptographic assurances, cryptographic key protection, logistics security, and possibly other measures. As in the case of most other security technologies (e.g., authentication, intrusion detection, etc.) one may assess the relative level of assurance provided by the security technologies, and then compare. From this perspective, PSTP compares favorably with PKI.

The purpose of this paper is to define a signature technology that deploys asymmetric key pairs to its enterprise servers without requiring users to accept their own asymmetric key pair credentials. If a particular market is unable or unwilling to accept asymmetric key pairs on the client machines, then PSTP provides an excellent signature technology.

The organization of this paper is as follows: Section 2 presents a high-level summary of PSTP that overviews the mechanism. Section 3 presents an example deployment. Section 4 presents an analysis and a consequential evidence comparison between PSTP and PKI. Section 5 presents related work, and Section 6 is the conclusion.

2. PSTP

2.1. Signature description

A PSTP signature has three components. The authenticator contains credentials that authenticate the signer. The message integrity component secures data, and the key management component fuses the first two components together (Fig. 1).

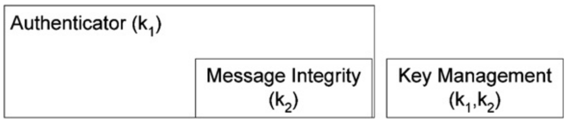

Fig. 1. PSTP signature.

2.1.1. Authenticator

The Authenticator contains the user's unique identifier, authentication credentials, and the Message Integrity Component. In order to ensure confidentiality of the user's authentication credentials, PSTP applies a symmetric algorithm against the entire authenticator.

Authenticator: $SYM(k_1,(USERID,TC,SC,MIC))$

Notation

- $SYM(k,y)$ denotes that symmetric key, k, encrypts information, y for the purpose of providing confidentiality. Specific examples of symmetric cryptography are triple DES [18] or AES [27] in CBC mode.
- k_1: Symmetric key.
- *USERID*: user's unique identifier.
- *TC*: *TC* is the user's transient credential. A transient credential is a credential that changes frequently. An example of a transient credential is a One-Time Password such as that provided by an RSA SecurID[1] [10].
- *SC*: *SC* is the user's static credential – a credential that rarely changes, or only changes upon the user's request. An example static credential is a password.
- *MIC*: *MIC* is the Message Integrity Component described in Section 2.1.2.

From the exclusive perspective of the authenticator, PSTP would not need to define the *TC* separately from the *SC*. Rather, PSTP could simply provide a credential that collects all authentication credentials into a single unit. However, since the Message Integrity Component (described below) handles the *TC* but not the *SC*, PSTP defines separate parameters.

[1] The degenerate case of PSTP has a null transient credential. In this case, PSTP operates correctly; however, the relative strength of the security mechanism is relatively low, and the MIC substitutes the SC for the TC.

2.1.2. Message integrity component

The Message Integrity Component (MIC) submits data into a message integrity function such as an HMAC [21]. The MIC's structure allows validation either at the time of the initial verification, or after the fact. One would compute the after the fact validation by combining the MIC with information submitted by the user or retrieved from logs. This external information consists of the USERID, TC, the signature's unique random number, r, timestamp, and a full copy of the data that was signed.

MIC: $MI(k_2,MD(USERID,TC,r,MD(data))), MD(data)$

Notation

- *MI*: $MI(k,z)$ denotes the keyed message authentication code (message integrity) function. *MI*, using key, k, applies against data, z. A specific example of a message authentication code function is HMAC computed using SHA-1 [1] or SHA-256 [28].
- *MD*: *MD* denotes a message digest function. $MD(x)$ is a deterministic function that maps bit strings of arbitrary length to bit strings of fixed length such that *MD* is collision-resistant and one-way (non-invertible). Examples include SHA-1 and SHA-256.
- r: r denotes a unique random number. PSTP expects r to be unique over time, server reboots, and multiple machines. One method of generating r is to compute SHA1(n, t, serverDN), where n is a unique number generated through a secure pseudorandom number generator; t is the server's current timestamp, and serverDN is the server's unique distinguished name found in the server's certificate. If the server generates multiple nonces with the same value, t, then each nonce must have a different value, n.
- *Data*: Data represents the original data submitted for a signature.
- k_2: k_2 is the symmetric message integrity (HMAC) key.

The *MI* covers the *TC* but not the *SC* due to limitations of consequential evidence validation use cases. A special tool available to a properly authorized administrator may reproduce a past value of the TC, if the administrator has access to the OTP's confidential symmetric key. However, no tool could reproduce the user's password (*SC*) at the time of the transaction. So, unless the verifier were to capture the user's password in a log at the time of the transaction (not recommended), an after-the-fact validation would have no access to the *SC*. Therefore, PSTP splits the authentication credential into two separate components, *TC* and *SC*, and handles the components differently.

At runtime, PSTP allows the server to validate a signature in two steps. The first step validates the signature independently of the data; and the second step validates that the message digest within the signature covers the data being signed. This two-step validation process is common to both PKI and PSTP signatures. If the server's programmers were willing to perform a signature validation in a single step, then the *MD*(*data*) outside of the scope of the MIC would be redundant because the server could compute *MD*(*data*) from the data directly in the unified step. However, pragmatic programming experience with signature validation implores the PSTP specification to adopt two-step validation.

If PSTP were to omit the *MI* operation, then the specification would yield a message digest encrypted by *SYM*. Insufficient cryptographic evidence exists justifying that ANSI and ISO certified symmetric encryption and message digest algorithms may be combined to yield the message digest property. As a result, PSTP takes the more conservative approach of adding the *MIC*.

2.1.3. Key management component

The Key Management Component cryptographically fuses the Authenticator and the Message Integrity Component into a single atomic unit.

$ASYM(e_a,(k_1,k_2))$

Notation

- $ASYM(e,X)$: $ASYM(e,X)$ represents an asymmetric cryptographic operation. Public keying material, *e*, encrypts information, *X*, for the purpose of providing confidentiality.

PSTP employs two symmetric keys due to the principle of key separation which stipulates "that keys for different purposes should be cryptographically separated" [29]. That is, PSTP avoids dual use of a single symmetric key for both confidentiality and integrity. Furthermore, the asymmetric operation of the Key Management Component should handle all of its protected information atomically. In other words, any party viewing a PSTP signature is in exactly one of the following two states. Either the party has sufficient information to discover both of the symmetric keys; or the party lacks the information required to discover either of the symmetric keys. The atomicity property prohibits the state in which a party only has sufficient information to discover one of the two symmetric keys.

Cryptographic literature demonstrates that the notion of atomicity roughly translates to the cryptographic term non-malleability [30,31]. RSA Encryption Primitive with Optimal Asymmetric Encryption Padding (RSAES-OAEP) [32,33] and Key Transport using Elliptical Curve Cryptography as specified in ANSI X9.63 [20] both provide the non-malleability property, and as a result, are appropriate for use in the Key Management Component.

2.2. Signature protocol

As illustrated in Fig. 2, PSTP is a four-step protocol, where the protocol's end-points are a client that initiates a transaction and a server that resides at the enterprise.

- *Step 1*: In the first step, the client downloads from the server two items: a server-generated unique nonce, *r*, and the server's certificate, which contains the server's public key, e_a. The client validates the server's certificate and extracts the public key for subsequent use in Step 2. The vehicle by which the client authenticates e_a is a local matter of a specific implementation and is outside the scope of PSTP. This paper adopts identical notation for the server's public key and certificate, but one may determine the notation's meaning from context.

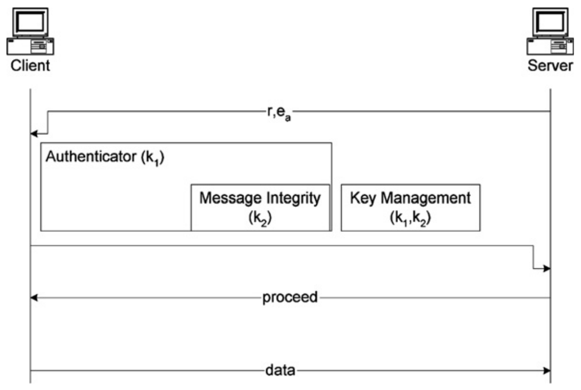

Fig. 2. PSTP overview.

- *Step 2*: The second step is the central cryptographic aspect of the protocol. The client transmits a single message that contains the Authenticator, Message Integrity Component, and Key Management Component as described in Section 2.1.
- *Step 3*: Step 3 is an optional signaling step. The client may optionally ignore this signal and proceed to Step 4 immediately.
- *Step 4*: The client transmits data to the server. The server cross-references this data into the Message Integrity component of Step 2. If the cryptographic seal communicated in the Message Integrity Component does not correspond to the data transmitted in Step 4, then the protocol raises an exception. If PSTP exits Step 4 without an exception, then the server executes the transaction. If the implementation chooses not to adopt the Step 3 option, then Steps 2 and 4 combine.

2.3. Consequential evidence

This section presents an example PSTP architecture that provides consequential evidence. The following list presents the architectural components:

- *Signer*: The signer holds keying material (e.g., a one-time password device token) and creates signatures.
- *Key handler*: The key handler is a trusted party that distributes and manages keys. The key handler holds the PSTP asymmetric private keying material, d_a, and has sufficient information to validate OTP authentication requests.
- *Transaction authorizer*: The transaction authorizer is a trusted party that assists every transaction. The transaction authorizer logs every transaction into an audit trail.
- *Transaction engine*: The transaction engine executes the transaction after the system validates a signature.

Fig. 3 illustrates the example architecture. The respective heavy lines between the Transaction Authorizer and the Key Handler and Transaction Engine denote two bidirectionally authenticated communication links (e.g., TLS) with certificates at both peers. The heavy boxes on the Key Handler and Transaction Authorizer highlight the notion that these parties are trusted to operate within the bounds of their respective specifications. The signer is a client, and the other architectural components are servers.

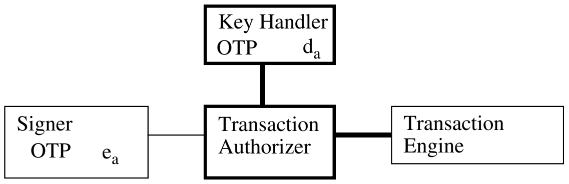

Fig. 3. Consequential evidence architecture.

The signer creates a PSTP signature which encrypts the key management component using asymmetric public key e_a. The Transaction Authorizer receives the PSTP signature and forwards to the Key Handler. The Key Handler applies its private keying material, d_a, to the key management component to reveal the symmetric keys k_1 and k_2.

Next, the Key Handler decrypts the authentication component using k_1 and performs the required authentication of the OTP. If the authentication succeeds, then the Key Handler returns a success code and returns the integrity component key k_2. The Transaction Authorizer validates the transaction using the integrity component key k_2. If successful, the Transaction Authorizer sends the authenticated transaction to the Transaction Engine and logs the PSTP signature into a secured audit trail [34].

One may revalidate a PSTP signature after the fact. First, decrypt the key management component. Next, validate the integrity of the audit trail using cryptographic means that are outside the scope of this paper. Next, apply a tool that validates the OTP after the fact. Finally, revalidate the integrity component.

The security of the architecture relies upon the operations of the two trusted parties: Key Handler and Transaction Authorizer. The following list highlights some of the most important security requirements:

Key handler

- Securely store keying material. Do not share the keying material with unauthorized parties.
- Participate in keying logistics. Distribute the OTP credentials to the correct parties while maintaining the privacy of asymmetric private keying material, when required.
- Participate in the revocation mechanism. Do not permit signatures executed from revoked keys.
- Use best security practices for operational procedures such as backups, reboots, etc.

Transaction authorizer

- Write all received requests to a tamper-evident log.
- Do not perform any action that would impact the integrity of a received request.
- Do not provide transactional information to the Key Handler.
- Do not authorize the Transaction Engine to accept a transaction unless the Transaction Authorizer correctly performs its services.

3. Client deployment

An OTP device token works best when a user is physically present to copy the OTP value (transient credential) from the OTP device token to the computer. Since the standard for remotely deployed human interfaces is a web browser which communicates with a server via HTTP or HTTPS, one would expect the most common use cases for PSTP to involve a web browser. After logging in, a user interacts with a remotely-located server by viewing web pages and submitting information. Eventually, the user reaches a point in the session where the user needs to perform a transaction with sufficiently high risk to require a signature (e.g., transfer funds between accounts). The user fills out a web-based form and signs the form using PSTP. The server then validates the transaction.

3.1. Implementation

Fig. 4 illustrates one example client-side implementation. The user views a browser-based form that prompts for all information required to execute the funds transfer (e.g., origin and destination account, and the dollar amount). The form additionally prompts the user for authentication credentials and presents a signature button. When the user presses the signature button, the form invokes the "request signature" Javascript routine. This routine collects parameters from the form such as the transaction details and the authentication credentials. The Javascript routine formats the parameters and sends all of the parameters to an Applet's "Sign" function which computes the BASE64-encoded PSTP signature. The applet returns the result to the Javascript; and the Javascript populates the form with the PSTP signature result. Next, the user submits the entire form to the server, including the newly populated PSTP signature.

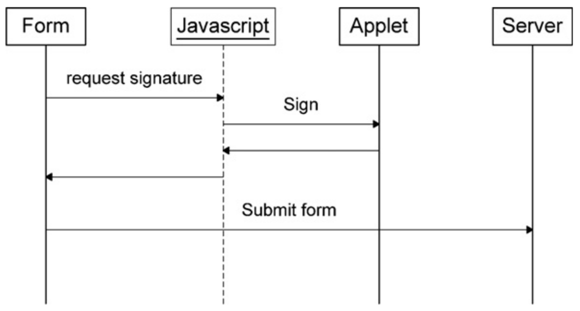

Fig. 4. Invocation sequence.

A common use case in the financial services industry includes a clerk who submits transactions for processing; however, the clerk does not have approval authority. Sometimes, the clerk enters the transactions into browser-based forms, and other times, the clerk uploads transaction files. In some cases, the client's Enterprise Resource Planning (ERP) tool automatically uploads the transaction list to the server directly. Workflow routes the list of transactions to one or more authorized approvers at the clerk's company who continue to interact with the server through their browser. Each of the approvers may view summary information, and drill down to discover the details of individual transactions. Each of the approvers sign the list of transactions using PSTP technology, and the Transaction Engine does not execute until successful validation.

The approver's form presents the transaction list and a "Sign" button. When the approver presses the "Sign" button, the Javascript collects all of the transaction details from the entire transaction list coupled with the static and transient credential, and gives those details to the "Sign" Applet. The Applet creates a message digest over all of the transactions and computes the PSTP signature. The Applet returns the signature to the Javascript which populates the form. The approver submits the entire form to the server for PSTP signature validation.

This example presents one possible deployment model. Other possible models also exist. For example, one may substitute other types of client-side software for the Applet. The constraint is that the system must have the capacity to compute complex cryptographic operations such as asymmetric encryption, symmetric encryption, and message digests. Simple execution environments such as Javascript are usually not sufficient.

The use cases for PSTP and PKI signatures are widespread. If a remotely located party wishes to sign a contract, then the party may potentially leverage signing technology. A financial transaction is one special case of a contract; however, the general case of contracts may extend to any type of binding agreement (e.g., court document submission [35], customer agreements, or service contracts). In general, signature technology obviates the need to optically scan a document and store the original paper.

3.2. Sessions

In an interactive session, the user to first logs in to a website by presenting a transient and static credential. After interacting through the browser, the user views a form that requires a signature (e.g., the user must approve a list of transactions). At the point of the signing event, the user enters a new transient credential; and the user enters same static credential that he or she entered at login time. Since the user may login correctly and subsequently step away from his or her machine, the strength of the initial authentication decays over time. Therefore, adding another prompt for authentication credentials at the point of the transaction event raises security assurance.

If malicious software (malware) exists on the user's machine, then the login event provides insufficient information for the malware to execute signatures. Rather, the malware must wait until the user enters the authentication credentials required for a signature before the malware may attempt to issue an errant transaction. In contrast, an asymmetric key pair device token is more vulnerable to a potential malware attack. Immediately after the user logs in, the state is that an unlocked asymmetric key pair device token has an electronic connection to the user's machine. Malware on the user's machine may interact with the asymmetric key pair device token with impunity executing multiple signatures. Therefore, unlike the case of PSTP, an asymmetric key pair device token has the vulnerability that it may potentially execute signatures without the user's knowledge. However, even in the case of PSTP, the user should only operate on trusted machines. Both OTP and asymmetric key pair device tokens are subject to a malware attack in which the malware changes the data that the user intends to sign.

4. Analysis

This section analyzes PSTP from various perspectives. The goal of this section is to explain intuitive reasoning justifying PSTP. Assumptions are coarse, and do not represent a formal proof of correctness.

In review of Section 2, a PSTP signature has the following form:

$$SYM(k_1,(USERID,TC,SC,MI(k_2,MD(USERID,\\ TC,r,MD\ (data))),MD(data))),ASYM(e_a,(k_1,k_2))$$

4.1. Message integrity analysis

The message integrity analysis argues that receiver does not validate a PSTP signature if an attacker modifies the data in-transit.

4.1.1. Notation

The argument of correctness uses unprimed lower case notation to indicate the values that the user sends, e.g., *tc*; and the notation uses primed notation to indicate the values that the user receives, e.g., tc'. In some cases, one may distinguish PSTP parameters from values by context.

The notation *pstp* and $pstp'$ denote the entire PSTP message at the sender's point of transmission, and the receiver's point of reception, respectively.

$VALID[pstp']$ denotes that the receiver obtains a PSTP message and successfully validates in accordance to the PSTP specification.

$VALIDATED(userid',tc',sc')$ denotes that the receiver obtains the $userid'$ and the associated authentication credentials. The receiver's validation of these credentials succeeded.

4.1.2. Assumptions

Assumption 1 (Authentication)

$$VALID[pstp'] \rightarrow VALIDATED(userid',tc',sc')$$

In accordance with the PSTP definition, the server fails to validate a PSTP signed message if the server receives invalid authentication credentials. Therefore, if the server receives a valid PSTP signed message, then the authentication credentials must be valid.

Assumption 2 (Confidentiality)

$$k1 \neq k1' \rightarrow tc \neq tc'\ OR\ sc \neq sc'$$

We assume that the attacker does not know *tc* or *sc* before the sender initiates the PSTP operation.

Otherwise, authentication would be meaningless. We also assume confidentiality of SYM and $ASYM$, so the attacker could not have discovered the values of tc or sc in-transit. If the attacker substitutes value $k1'$ for $k1$, then the attacker would not have enough information to encrypt the correct tc and sc values. This assumption embeds the fact that the sender uses the correct public key, e_a, of the receiver.

Assumption 3 (OTP authentication)

$$userid = userid' \ AND \ (tc \neq tc' \ OR \ sc \neq sc') \rightarrow \sim VALIDATED(userid', tc', sc')$$

alternatively,

$$userid = userid' \ AND \ VALIDATED(userid', tc', sc') \rightarrow tc = tc' \ AND \ sc = sc'$$

If the user supplies incorrect authentication credentials, then the server does not validate. If the user plays back correct authentication credentials, then one-time password semantics prohibit validation of the previously supplied tc'.

If the attacker breaks another user's authentication credentials, and substitutes the other user's credential then $userid \neq userid'$.

Assumption 4 (Non-malleability)

$$k1 = k1' iff \ k2 = k2'$$

Non-malleability provides the concept of atomicity. Non-malleability ensures that the receiver cannot receive only one correct key.

Assumption 5 (Nonce uniqueness)

$$VALID[pstp'] \rightarrow r = r'$$

PSTP operational semantics guarantee uniqueness of the nonce.

Assumption 6 (Integrity and confidentiality)

$$userid = userid' \ AND \ tc = tc' \ AND \ sc = sc' \ AND \ r = r' and \ k2 = k2' \ AND \ k1 = k1' \rightarrow data = data'$$

An attacker cannot modify the data while keeping the remainder of the parameters unchanged. This assumption embeds the fact that the sender uses the correct public key, e_a, of the receiver.

4.1.3. Justification of message integrity

$$GOAL: \ VALID[PSTP] \ AND \ userid = userid' \rightarrow data = data'$$

The first step of the argument assumes that the recipient correctly validates a PSTP signature, and that the signature arrives from the 'correct' party. If an attacker were to compromise a different user and steal that user's authentication credential, then the attacker would have the ability to create signatures based upon the attacked user's stolen credentials. Therefore, in this argument, we assume execution of PSTP in the presence of uncompromised credentials with the correct userid.

1	$VALID[pstp'] \ AND \ userid = userid'$	Assumption
2	$VALIDATED (userid', tc', sc')$	Assumption 1 – Authentication
3	Suppose $k1 \neq k1'$	Supposition
3a	$tc \neq tc' \ OR \ sc \neq sc'$	Assumption 2 – Confidentiality
3b	∼VALIDATED $[userid', tc', sc']$ Contradiction	Assumption 3 – OTP authentication 2,3b
4	$k1 = k1'$	Supposition is false
8	$k2 = k2'$	Assumption 4 – Non-malleability
9	$tc = tc' \ AND \ (sc = sc')$	Assumption 3 – OTP authentication
10	$r = r'$	Assumption 5 – Nonce uniqueness
11	$data = data'$	Assumption 6

4.2. Message authentication analysis

Message authentication is a direct result of Assumption 1. If the recipient validates a PSTP signature, then the validation includes a step where the recipient validates the sender's authentication credentials.

4.3. Replay protection analysis

Both OTP semantics and the nonce protect against the possibility of replay attacks.

4.4. Key lifetime analysis

With the exception of the asymmetric key pair owned by the server, the PSTP specification exclusively uses one-time keys. This mechanism significantly reduces the risk of key compromise.

4.5. Consequential evidence analysis

The mechanism described in Section 2.3 explains how one may store the PSTP signature in an audit trail and subsequently validate the signature. These observations satisfy the goal of providing consequential evidence.

4.6. Entropy and algorithms analysis

Every aspect of the specific instance of PSTP defined in Section 2 conforms to an approved international standard with the exception of the authentication mechanism, which may be proprietary (e.g., SecurID).

4.6.1. Approved by ISO or ANSI

- RSA encryption used for key transport: [23] (alternatively Elliptical Curve Cryptography [20])
- Triple DES (3DES): [18]
  HMAC: [21]
  SHA1: [1]
  Public Key Cryptography including certificate structures: [19]

4.6.2. Approved by IEEE, PKCS

RSAES-OAEP: [22,32].

4.6.3. Draft status

ANSI X9.44: [17] ANSI X9.44 specifies a key transport mechanism for the financial community. ANSI X9.44 protects against unauthorized change of the transported keys, and ensures that the keys transport as a block (i.e., an attacker cannot compromise or modify only one of the keys).

OASIS Digital Signature Standard (DSS): [36]. Key entropy is an important requirement. Any cryptographic facility that does not have the inherent flexibility to incorporate best cryptographic algorithms should not be considered for use in the space of wholesale financial services due to the high monetary value of transactions. PSTP does not limit key entropy or key length in its asymmetric or symmetric algorithms. PSTP does not mandate a specific message digest or HMAC size.

4.7. Secured media analysis

PSTP operates in the context of Secured Media deployed to users (i.e., OTP device token). The authentication technology requires no electronic connection between the media and the user's machine. The impact is good user mobility between machines, easy installation, and low hardware support costs. Malware on the user's machine cannot initiate a signature event without the user's knowledge.

In contrast, a hard token that holds an asymmetric key pair requires the user to enter an authentication credential such as a password to unlock the token. Subsequently, the token and any potential malware on the user's machine may freely communicate without the user's knowledge. Therefore, the asymmetric key pair hard token may potentially participate in signatures without the user's knowledge.

5. Related work

5.1. Comparison of consequential evidence with non-repudiation

This section discusses three perspectives of the term non-repudiation, and contrasts these perspectives with the concept of consequential evidence.

- *Technical non-repudiation*: The common technical definition of non-repudiation directly references asymmetric cryptography, as in the case of the following US National Institute of Standards and Technology (NIST) definition.
  Non-repudiation [sic] is a service that is used to provide assurance of the integrity and origin of data in such a way that the integrity and origin can be verified by a third party. This service prevents an entity from successfully denying involvement in a previous action. Non-repudiation [sic] is supported cryptographically by the use of a digital signature that is calculated by a private key known only by the entity that computes the digital signature [16,36].
- *Business non-repudiation*: Business non-repudiation takes into account all artifacts of a signature that may impact the business. Example artifacts may include operating assumptions, cryptographic services, and legal council.
- *Legal non-repudiation*: Legal non-repudiation comprises the various national and international laws pertaining to signatures.

Consequential evidence is an artifact of cryptography used to provide after-the-fact evidence. The business or legal frameworks may reference this artifact when assessing. The degree to which the businesses or the courts trust the consequential evidence artifact may be case dependent.

The NIST definition is not sufficient for guaranteeing a significant degree of business non-repudiation because the definition fails to mention important relevant security concepts. For example, the NIST definition does not reference the strength of mechanism in the storage media; and the definition does not mention the timestamp. With respect to the timestamp, consider the sequence of events illustrated in Fig. 5.

In Step 1, User A signs a transaction. In Step 2, user A claims an event which requires a certificate revocation. In Step 3, User A claims to repudiate the signature created in Step 1. User B reverses the order of the operations performed in Steps 1 and 2. That is, first User B revokes his or her certificate, and second, User B signs by applying the asymmetric private key associated with the revoked certificate. At the time of Step 3, a judge should disallow a repudiation claim of User A's signature because User A signed using a valid certificate. However, the judge should allow User B's repudiation claim, because User B signed using a previously revoked certificate. Unfortunately, the judge does not have sufficient information to distinguish between the two sequences.

If a judge were to compare the relative strength of mechanism of a PKI-based signature that has no trusted timestamp, against a PSTP-based signature that leverages an OTP device token based upon time, e.g., SecurID, then which signature merits the superior evaluation? In comparison, when validating a historical signature using PSTP, a verifier may produce an OTP value from a historical timestamp, and cryptographically combining the nonce with the appropriate key. The verifier may compare this OTP value against the one contained in the PSTP signature log. As a result, an OTP device based upon time has an assumed timestamp. This comparison is pertinent to common industry practice today, because Time Stamp Authorities (TSAs) do not yet enjoy widespread acceptance.

If one were to patch the security hole in PKI by adding a TSA, then a judge must evaluate the relative trustworthiness of the TSA's operations when assessing the strength of mechanism of a PKI-based signature. The timestamp authority's evaluation may take into account an audit, e.g., SAS 70 [37], testimonies of some of the trusted timestamp authority's key employees, TSA operating assumptions [38] and possibly other artifacts. These artifacts would be similar in nature to the artifacts used to justify a PSTP signature. Since a PKI's timestamp authority has the ability to manipulate time, it is an ultimate authority that only needs to trust itself. From either a business or legal perspective, if one could show that the TSA deviated from its operating assumptions, then any PKI-based signature created with the assistance of the TSA might be subject to repudiation.

Furthermore, in order to belie credence in a PKI-based signature, a judge must trust the service which operates the Certificate Authority. This means that the Certificate Authority, for example, must not forget to reference revoked certificates on its certificate revocation lists or OCSP responder [39]; distribute certificates to the wrong parties; disclose the root certificate's private keying material to unauthorized parties; fail to secure backups; fail to transfer the latest state of certificate revocation lists to a disaster recovery center; incorrectly authorize subordinate root certificates; or execute other unauthorized actions either erroneously or maliciously. The artifacts that would be needed to claim correct operations of the Certificate Authority may be quite complex. Presumably, the US Federal government agrees with this position because it certifies particular commercial PKI providers, thus implying that signatures created using untrusted PKI providers might be subject to business or legal repudiation [40].

Fig. 6 presents a comparison of the PKI and PSTP infrastructures from an operational perspective. The figure depicts the PKI components above the dotted line and the PSTP components below the dotted line. In order to believe in a consequential evidence claim, the signer, verifier, and judge must "trust" that the components highlighted in the heavy boxes operate in accordance to their specifications. That is, in the case of the PKI, the parties must trust the Certificate Authority and Timestamp server; and in the case of PSTP, the

	Step 1	**Step 2**	**Step 3**
User A	sign	revoke	repudiate
User B	revoke	sign	repudiate

Fig. 5. PKI signature without timestamp sequence of events.

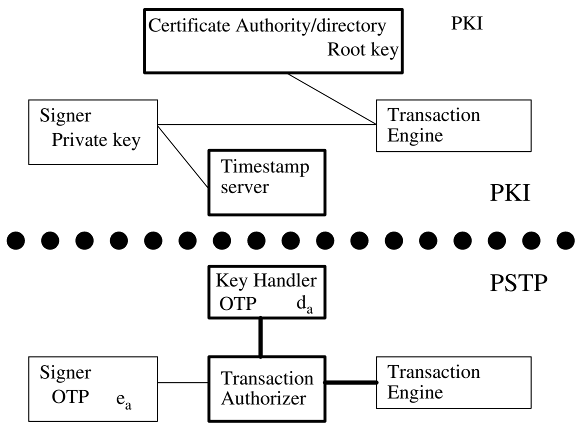

Fig. 6. Consequential evidence comparison.

parties must trust the Key Handler and Transaction Authorizer.

Should a judge assess greater credence in the consequential evidence claims of the PKI or PSTP? The answer to this question may be more dependent upon the specific artifacts, than the mathematics of cryptography. If the judge questions the operations of any of the trusted parties, then the claims of consequential evidence could be weak from either a business or legal perspective.

From a legal perspective, examples exist in which cryptographic strength may be irrelevant. For example, in the United States "E-SIGN does not prescribe requirements pertaining to activities that are governmental (as opposed to business, consumer, or commercial) in nature, including activities conducted by private parties principally for governmental purposes. One example of an activity that is governmental is census reporting and related requirements" [41]. So, in the US, one may use neither PKI, PSTP, or any other kind of electronic signature for governmental activities such as census reports. Furthermore, in accordance with common law, "if a person denies that a signature is his, the relying party has to prove that it is truly that of the person denying it. Onus of proof is on the person seeking to rely on the signature" [42].

One may improve the strength of mechanism of consequential evidence using security mechanisms outside the scope of cryptography, such as separation of duty. In the case of PSTP, an enterprise may potentially rely upon independently trusted services such as those provided by the Verisign authentication service bureau [43] or the RSA Authentication Service [44]. Alternatively, the enterprise may operate its trusted services internally but separated from the remainder of the enterprise (e.g., the Transaction Engine) by a Chinese Wall [45]. Proposed standards such as the OASIS Digital Signature Standard Signature Gateway Profile [46] may provide the standards which help facilitate separation of duty within PSTP.

5.2. Comparison with other protocols

This section compares PSTP with other, similarly constructed protocols.

5.2.1. S/Mime and SET, PGP, and PEM

The Internet Engineering Steering Group tentatively approved the use of RSAES-OAEP for S/Mime in April, 2003 [47]. The draft S/Mime standard specifies a protocol that has some structural similarities to PSTP because it negotiates a symmetric key encrypted with an asymmetric key. However, any signature created using S/Mime requires an additional client-side asymmetric key pair. If one does not elect to use the S/Mime option for signatures, then the protocol lacks the equivalent of PSTP's MIC, and as a result would have no cryptographically-strong integrity guarantee. PGP [48] and Privacy Enhanced Mail [49] have the same general form as S/Mime.

Secure Electronic Transactions (SET) [50] was a standard for signed retail payment transactions proposed in the 1990s. SET requires a client-side asymmetric key pair. SET failed in the market because its infrastructure (including client-side asymmetric key pairs) was too complex for a general, retail environment.

5.2.2. Key exchange without asymmetric Keys deployed to clients

The most common usage of TLS relies upon an asymmetric key pair deployed to a server, exclusively. Common usage authenticates the client by sending a password or other authentication credential through a previously established TLS session. Although this technique authenticates the client, it does not provide signature semantics with consequential evidence. IPSec's draft XAuth mode [51] augments IPSec by authenticating the user as well as the device; however, IPSec does not provide signature semantics.

Other schemes for creating sessions based upon passwords deployed to clients have been proposed. Halevi [52] creates sessions where the server has an asymmetric key pair and the client has a password. Kolesnikov [53] identified security deficiencies which were corrected by introducing a long password. A long password has the property that it adds cryptographic entropy; however, it is too long for a user to memorize or type into a browser-based form. A long password suffers many of the same ergonomic disadvantages exhibited by an asymmetric key pair. That is, if the storage media is not properly secured, then the long password is subject to theft without the owner's knowledge. On the other hand, if one stores a long password on a smart card, dongle, or USB token, then the user must execute a device installation step and cannot easily move between machines.

In comparison of authentication credentials, a password has the advantage that it requires no storage media; however, its authentication strength is low and it does not introduce good cryptographic entropy. An asymmetric key pair has the advantage of that it exhibits good cryptographic attributes; however, its storage media introduces ergonomic concerns. A one-time password has the advantage that it is an authentication factor with good strength of mechanism; however, it does not introduce entropy or bind data to the OTP value. PSTP addresses the deficiencies of one-time passwords. One cannot simply substitute a password-based key agreement scheme for PSTP because the password-based schemes usually rely upon properties of authentication credentials that do not possess either the same security or ergonomics as OTP devices.

5.2.3. HMAC with static key distribution

One could potentially physically distribute HMAC keys via a hardware security module to numerous clients. However, the key distribution method would be manual and cumbersome. These difficulties would prevent frequent HMAC key distribution, and as a result, would encourage re-use of HMAC keys and long key lifetimes.

5.2.4. Software safe

A software safe is a file which contains asymmetric private keying material encrypted using a symmetric key. A user authenticates to a server using an ergonomically-acceptable means such as an OTP. After validating the authentication event, the server returns the decryption key of the software safe. The user applies the decryption key to the software safe, extracts the asymmetric private keying material, and executes conventional asymmetric signatures.

A software safe and PSTP share a similar advantage: both mechanisms provide signature semantics, yet allow the user to authenticate using an OTP device token. PSTP, however, is much more secure. An OTP device token's confidential key never leaves the perimeter of the OTP device token on the client machine. On the other hand, once a software safe decrypts, the safe's contents may be copied.

During the period that a user physically possesses an OTP device token, the user knows that no one else can authenticate or sign on the user's behalf, without the user's knowledge. A software safe provides a lower level of assurance because malware on the user's machine could potentially copy the software safe's contents or encryption key to an unsecured location. While one should be careful when executing a signature in an untrusted environment such as an Internet Café, the vulnerabilities of software safes may be prohibitive.

5.2.5. Cryptographic calculator

As described in the introduction, a cryptographic calculator shares with PSTP the ability to bind an OTP value to data. However, the cryptographic calculator's ergonomic concerns force the mechanism to rely upon checksums that do not have a sufficient number of bits to qualify as a proper cryptographic message digest.

6. Conclusion

The comparison between PSTP-based signatures and PKI-based signatures is complex. Both technologies provide similar functional characteristics (e.g., message authenticity, integrity, replay protection, and consequential evidence). When deployed using properly secured token media, both technologies adequately protect confidential keying material. From a purely cryptographic perspective, a PKI has an advantage because the server has no access to the client's confidential keying material. At first glance, this advantage provides the appearance that a server cannot forge a valid signature on a client's behalf. However, this appearance is merely a façade. The timestamp server can create a signature using a compromised asymmetric key pair, and change history by moving the signing date so that it precedes the date that the asymmetric key pair first appeared on the revocation list. Other nuances in a PKI's Certificate Authority also exist which further degrades the cryptographic advantages of a PKI.

PSTP enjoys some advantages over PKI technology from a security perspective. Since an OTP inherently requires user interaction, malware on a user's machine cannot sign without the user's knowledge. When one takes into account all of the characteristics of PSTP and PKI, both technologies provides excellent security. However, both technologies have caveats and nuances that render a comparison of strength of mechanism difficult to quantify.

One should view PSTP as an alternative to PKI, as opposed to a general-purpose replacement. Straight-through processing which requires no explicit human interaction is a better candidate for PKI than PSTP technology, because no one would be available to copy the OTP token value from the OTP device token to a computer. Furthermore, the administrators of both peers in the straight-through processing system should have the ability and motivation to handle an asymmetric key pair. A browser-based use case, on the other hand, is more suited for PSTP. An OTP device token's lack of electronic connection between the user's machine and the OTP device token has installation, mobility, and security advantages.

PSTP signatures, paper-based signatures, and PKI-based signatures all have inherent costs. Whenever the cost of PSTP signatures is the least of the three methods, PSTP may provide the most viable solution. The clearest examples are environments that already deploy OTP device tokens. The financial service community is a good candidate because the high risk of financial transactions induces the financial community to use strong security. A second candidate is a corporation that deploys OTP device tokens to its employees so that they may remotely login through a VPN. Since the employees already own OTP device tokens, the enterprise may dual-purpose these tokens for PSTP signatures. So, for example, when the employee needs to sign an internal corporate document, e.g., change tax withholding status, then employee may use his or her OTP device token for the PSTP signature. Another use for PSTP is to business process re-engineer a paper-based environment. Consider, for example, a court document submission which requires a series of signatures. If the cost of handling the paper exceeds the cost of deploying an OTP device token, then PSTP may be a good, cost-saving solution.

JPMorgan Chase Treasury Services successfully deployed PSTP technology for browser-based wholesale financial transactions. JPMorgan Chase dropped in PSTP software as a replacement for PKI software without significantly changing its applications' programming logic. This successful deployment provides evidence that PSTP is a viable, currently available technology.

Acknowledgements

I would like to thank Nikhil Sathe, Robert Levine, Russ Logar, Richard Yen, Joseph Calaceto, Nick Williams, Daryl Wooldridge, Bill Mann, Burt Kaliski, Daniel Brown, and Robert Gallant, and the anonymous referees for their analysis and suggestions. I would also like to thank Milind Nagnur, Richard Hultstrom, Bill Flood, Shaheen Abdul Jabbar, Vikrant Sutar, Umesh Sonavane and the remainder of the engineering team for building the software, and deploying into the production environment.

References

[1] ANSI X.9.30:2-1997, Public-Key Cryptography for the Financial Services Industry, Part 2: The Secure Hash Algorithm (SHA1) (revision of X9.30:2-1993).
[2] Benson et al., Portable security transaction protocol, US Patent Application, 15559-20.
[3] RFC 2246, June 2003, T. Dierks, E. Rescorla, The TLS Protocol.
[4] J. Dray, D. Balenson, An overview of the advanced smartcard access control system (ASACS), in: Proceedings of the PSRG Workshop on Network and Distributed System Security, February 1993, pp. 125–133.
[5] http://en.wikipedia.org/wiki/Dongle.
[6] Universal Serial Bus 2.0 specification, April 2000. Available from: <http://www.usb.org>.
[7] Burr et al. NIST Special Publication 800-63, Electronic Authentication, 2004.
[8] Authentication in Internet Banking, Federal Financial Institutions Examination Council, 2005. Available from: <http://www.ffiec.gov>.
[9] Monetary Authority of Singapore, Circular No. SRD TR02/2005, 25 November 2005.
[10] SecurID. <www.rsasecurity.com>.
[11] Vasco. <http://www.vasco.com>.
[12] No Token Resistance Citi rolls out Digipass authentication devices to biz clients, Bank Systems and Technology, 25 May 2006.
[13] RSA Lands ETrade for Risk-Based Authentication, eWeek.com, 1 March 2006.
[14] FSTC Concludes Phase I of Better Mutual Authentication Project, Financial Services Technology Consortium Press Release, 30 May 2006.
[15] Uniform Commercial Code – Article 4A, The American Law Institute and the National Conference of Commissioners on Uniform State Laws, 2003.
[16] Barker et al., NIST Special Publication 800-57, Recommendation for Key Management Part 1: General.
[17] ANSI X9.44 (draft): Key Management Using Reversible Public Key Cryptography for the Financial Services Industry. Draft, 1998.
[18] ANSI X9.52-1998, Cryptography for the Financial Services Industry: Triple Data Encryption Algorithm Modes of Operation.

[19] ANSI X9.57-1997, Public Key Cryptography for the Financial Services Industry: Certificate Management.
[20] ANSI X9.63-2001, Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptical Curve Cryptography, 2001.
[21] ANSI X9.71-2000, Keyed Hash Message Authentication Code (MAC).
[22] IEEE Std 1363-2000. IEEE Standard Specifications for Public-Key Cryptography. Approved 30 January 2000.
[23] ISO/IEC 11770-3. Information technology – Security techniques – Key management – Part 3: Mechanisms using asymmetric techniques.
[24] RFC 2401, November 1998, S. Kent, R. Atkinson, Security Architecture for the Internet Protocol.
[25] A. Nadalin et al, Web Services Security: SOAP Message Security 1.0 (WS-Security 2004), OASIS Standard 200401, March 2004.
[26] A. McCullagh, W. Caelli, Non-repudiation in the digital environment, First Monday 5 (8) (2000). Available from: <http://firstmonday.org/issues/issue5_8/mccullagh/index.html>.
[27] Advanced Encryption Standard, Federal Information Processing Standards (FIPS) Publication FIPS-197, US Doc/NIST, November 2001.
[28] Secure Hash Standard, Federal Information Processing Standards (FIPS) Publication FIPS-180-2, US Doc/NIST, August 2002.
[29] A. Menez et al., Handbook of Applied Cryptography, CRC Press, Boca Raton, 1997 (Section 13.5.1).
[30] RFC 3560, July 2003, R. Housley, Use of the RSAES-OAEP Key Transport Algorithm in the Cryptographic Message Syntax (CMS).
[31] M. Bellare, P. Rogaway, Optimal asymmetric encryption – how to encrypt with RSA, in: Advances in Cryptology – Eurocrypt'94, Springer-Verlag, New York, 1994, pp. 92–111.
[32] RSA Laboratories, PKCS #1 v.2.0: RSA Cryptography Standard, Version 1.5, November 1993.
[33] M. Bellare et al., Relations among notions of security for public-key encryption schemes. Available from: <http://eprint.iacr.org/1998/021/>.
[34] B. Schneier, J. Kelsey, Cryptographic support for secure logs on untrusted machines, in: The 7th USENIX Security Symposium Proceedings, USENIX Press, 1998, pp. 53–62.
[35] M. Abdulaziz, R. Chambers, J. Messing, Legal XML Proposed Standard: XML Standards Development Project – XML Court Document 1.1 Draft Standard.
[36] OASIS, Digital Signature Standard. Available from: <http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=dss>.
[37] http://www.sas70.com.
[38] RFC 3161, August 2001, C. Adams et al., Internet X.509 Public Key Infrastructure Tim Stamp Protocol (TSP).
[39] RFC 2560, June 1999, X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP.
[40] L. Springer, K. Evans, D. Safavian, Electronic signatures: how to mitigate the risk of commercial managed services, Office of Management and Budget, Executive Office of the President, M-05-05.
[41] Guidance on implementing the electronic signatures in global and national commerce act (E-Sign), Office of Management and Budget, Executive Office of the President.
[42] L. Hamel, Electronic signatures, Harvard Law School Clinical Lecture, 14 October 2003. Available from: <http://www.mass.gov/Aitd/docs/legal/hls_clinical_lecture_electronic_signatures.ppt>.
[43] The Verisign Authentication Service Bureau. http://www.verisign.comurl<http://www.verisign.com>.
[44] RSA Laboratories, One-time password validation service, V1.0, Draft 1, 8 April 2005.
[45] D. Brewer, M. Nash, The Chinese wall security policy, in: IEEE Symposium on Research in Security and Privacy, 1–3 May 1989, Oakland, CA, pp. 206–214.
[46] Benson et al., OASIS, Digital Signature Standard, Signature Gateway Profile. Available from: <http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=dss>.
[47] Internet Engineering Steering Group meeting minutes, 17 April 2003. Available from: <http://www.ietf.org/iesg/iesg.2003-04-17>.
[48] RFC 1991, August 1996, D. Atkins et al., PGP Message Exchange Formats.
[49] RFC 1422, February 1993, Kent, S., Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management.
[50] Secure Electronic Transaction Specification, Book 2: Programmer's Guide, v. 1.0, May 1997.
[51] S. Beaulieu, R. Pereira, Extended Authentication within IKI (XAUTH), Internet Draft, October 2001.
[52] S. Halevi, H. Krawczyk, Public Key cryptography and password protocols, in: CCS'98: Proceedings of the 5th ACM conference on Computer and Communications Security, ACM Press, New York, 1998, pp. 122–131.
[53] V. Kolesnikov, C. Rackoff, Key exchange using passwords and long keys, in: 3rd Theory of Cryptography Conference, TCC 2006, New York, NY, 5–7 March 2006.

Glenn Benson is a security architect of JPMorgan Chase Treasury Services. His responsibilities include all aspects of security architecture covering hundreds of applications that cumulatively process more than USD 3 trillion daily. He received his PhD from Georgia Institute of Technology and has worked throughout his entire career in the information security industry. He has five patents, and additional patent applications.