Digital Signatures for Web Applications
– CBT Solution White Paper

AUTHORS:
Jan Sander Jensen (jsander@dk.ibm.com), Thomas Gundel
IBM Crypto Competence Center Copenhagen

Contents

1. Introduction

This white paper describes a solution for digital signatures and PKI called “Crypto-Based Transactions” (or just CBT) developed at the IBM Crypto Competence Center Copenhagen. The document is targeted at solution architects who are considering using the solution to secure a web application or portal. The main focus will therefore be on the functionality provided by CBT and how to incorporate it in an overall architecture. For details on installing and configuring the solution we refer to the product manuals.

The CBT solution has existed for more than a decade and has several variants including an applet version, an API version for thick clients / stand-alone applications and as an HTTP proxy. The present document will focus exclusively on the CBT Thin Client Applet solution for web applications.

The reader is assumed to be familiar with basic concepts within PKI, Java, digital signatures and web applications.

2. Functionality Provided by CBT

This chapter provides an overview of the functionality provided by CBT. First, the fundamental security services needed by most applications are described and subsequently it is shown how they can be realized with CBT.

In the literature, the following generic security services are often identified as requirements for sensitive applications:

- Confidentiality
- Integrity
- Authenticity
- Non-repudiation

All of these can be achieved by a web application or portal by leveraging the security mechanisms provided by CBT. Specifically, CBT uses digital signatures and other cryptographic techniques to implement them:

1. Users will be able to securely logon to the application using a digital signature.
2. Users will be able to digitally sign binding agreements with the service provider[1].
3. Users can be securely registered, have signature keys generated and certificates issued via CBT.
4. True end-to-end encryption can be achieved between the user's browser and a back-end server.

In the following sections each of these functions will be described. More details can be found in section 3 where the message flows for common use cases are discussed.

2.1 Logon

An important security mechanism needed by most web applications is secure logon where the user proves his identity and a session is established. Traditionally, most applications have implemented logon with a static user-ID and password. CBT offers a more secure logon with digital signatures which achieves true two-factor authentication.

The CBT logon is implemented by including the CBT Logon Applet in the web application's logon page. Once the logon page is loaded into the user's browser, the CBT Logon Applet will allow the user to select an authentication key and enter a password/PIN. For supported key stores see section 4.3. The applet will then sign a challenge or time stamp with the user's private key and send the

[1] The organization providing the application or portal leveraging CBT will be called "the service provider". It will also be the relying party in PKI terminology because it provides services to users in response to valid signatures.

signature to the server for verification. Once the signature is verified a secure session can be established and the application flow can continue (e.g. with a welcome page).

Note that CBT logon contains no features for session management, creation of a user context or subsequent access control (authorization). Such functionality can be achieved by integrating CBT with access control systems such as Tivoli Access Manager or with the security mechanisms found in the application server platform (e.g. WebSphere Application Server or WebSphere Portal Server). Such integration is described in section 3.2.1.

2.2 Digital Signing of Agreements

In many applications there is a requirement of being able to enter binding agreements between the user and the service provider. The agreements can e.g. be purchase orders, money transfers, terms and conditions, subscriptions etc. By having the user sign binding agreements the service provider can reduce the risk of doing business electronically considerably.

PKI technology allows the user and service provider to electronically enter a binding agreement. In practice, this is implemented by having the user sign an electronic representation of the agreement with his private key. Since only the user will be able to produce the signature he will not be able to later deny having signed it– a concept also known as non-repudiation. In many states / countries there is a legal framework which makes digital signatures legally binding. Legal aspects of digital signatures are outside the scope of this paper however.

2.3 Secure Registration

The use of digital signatures for logon and signing agreements require that the user has signing keys available and optionally also digital certificates.

Some service providers assume that the user already has obtained such credentials from trusted third-parties such as Certificate Authorities. This has the advantage that a service provider does not have to implement secure registration of users but can provide immediate access to the application based on the credential. CBT is able to leverage credentials from most Certificate Authorities to implement such scenarios.

In other scenarios, the service provider might want to handle the registration of users to remain in tight control with the process. This is commonly the case with internet banking. The registration process will usually be implemented electronically as a flow through a web application possibly supplemented with a traditional process such as identification at a branch office. The result of the registration will be that one or more signature keys (private/public key pairs) are generated on the user's computer and the public parts registered with the user's identity on the server. Optionally, a certificate can be issued containing the user's identity attributes and public key.

CBT Applets and server components exist to implement this registration process; for example, one applet is used to generate a signature key pair on the user's computer and create a certificate request; another applet is used to receive any certificates created by a CA. The applets are designed to be used within the web application implementing the registration process.

2.4 End-to-End Encryption

Traditional web applications use the SSL / TLS protocol to obtain a secure communication channel between the browser and web server. One problem with SSL / TLS is that data is only protected during transport and that the encryption typically terminates at the first web server in the service provider's infrastructure.

In some cases there is a requirement to protect data independently of the transport layer and to ensure that data will only be decrypted at a trusted back-end server. For this purpose CBT has components capable of performing encryption and decryption at the application layer.

CBT end-to-end encryption is implemented on the client side by:

- An extension to the Logon Applet which establishes a (symmetric) session key.
- A cipher applet which performs en- and decryption in the browser. This applet must live throughout the entire browser session.

On the server side, a CBT Session API is used to keep track of the current session key for each user and to perform the actual cipher operations. This API is capable of using IBM cryptographic hardware to speed up the operation and protect the keys.

Note that end-to-end encryption has the clear disadvantage that the application layer becomes aware of the encryption. Thus, the application must decide which data (e.g. part of the HTML) should be protected and call CBT components as needed.

2.5 One-Time-Codes

It is possible to combine digital signatures and one-time-codes (OTC) to achieve a higher level of security. This is particularly relevant if software based keys, or software certificates are used. An additional OTC field can be enabled in the CBT applets. The OTC is then included in the signature.

This can be used with many different OTC schemes, independent of whether the distribution of the OTC code is done via SMS, paper or plastic cards, tokens or other means.

2.6 Fingerprint of client PC

CBT can extract a fingerprint of certain hardware and software on the client. This information can be stored on the server and used in subsequent logins to help determine the level of risk associated with a transaction.

3. Architecture of the CBT Solution

The CBT Thin Client Applet solution is targeted for client-server web applications. In this chapter the CBT components for this environment are described and the overall architecture of a web application using CBT is illustrated.

The CBT components are usually delivered as building blocks which the service provider will need to integrate with their web application / portal and optionally with security middleware. This gives the service provider a great deal of flexibility. On the client side CBT consists of Java applets which are executed in the user's browser. The server components consist of Java APIs[2] which are capable of processing the messages produced by the applets and calling external certificate authorities.

The service provider will usually need to integrate the CBT components with his web application or security middleware. For example, he will need to create a logon HTML page which calls the CBT Logon Applet and further implement server side logic (e.g. servlet or CGI program) which receives the Logon message (signature) produced by the applet and call the CBT server APIs to get the signature verified. Furthermore, the service provider's application will need to keep track of the user session and perform all required logging.

More shrink-wrapped versions of CBT exist including complete J2EE web applications for logon and integration with the Tivoli Access Manager product. These will be briefly described in section 3.2.1.1.

[2] APIs for C and COBOL are also available for selected functionality.

3.1 Architecture Overview

The diagram below shows the typical solution architecture for a web application based on CBT:

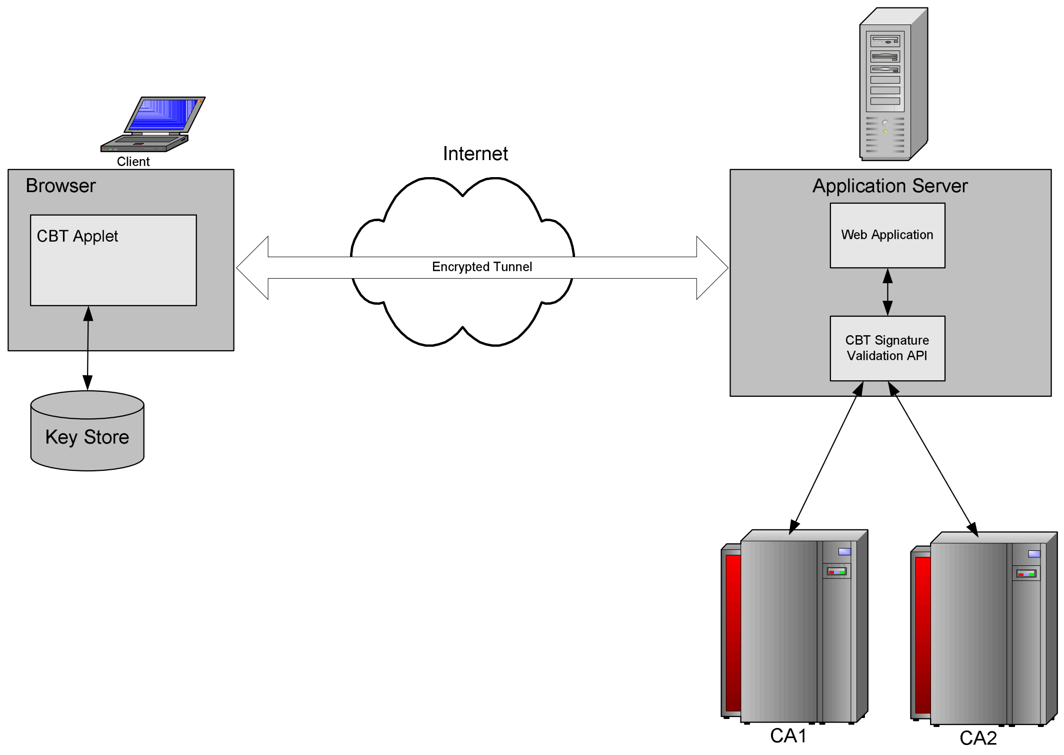


Figure 1: Typical Solution Architecture

On the client side, CBT consists of Java applets running in the user's browser. The applets are code-signed and therefore allowed to step out of the sandbox environment to access key stores on the user's computer. The server launches CBT applets by sending web pages to the browser which reference them via special html tags. The applets will visually control a rectangular area within the web page where they can display a user interface. Colours and fonts can be customized to match the layout of the surrounding webpage.

Once the applets have performed their operation (signing or key generation) they will usually need to deliver a *message* to the server. This message can be a registration message with a certificate request, a logon message with a signed challenge / timestamp or a signed agreement. All messages are delivered from the applets by calling a JavaScript function which must be implemented in the web page. This call-back handler can then contact the server usually by submitting (HTTP POST) a form with the message included as a form variable.

Note that all communication between CBT applets and the server is performed via the user's browser:

- The server passes input parameters to the applets by including them as applet tags in the web pages starting the applets.

- The applets send their messages to the server by calling JavaScript callback handlers which submits the messages to the server.

Since all communication is performed via the browser the CBT solution normally re-uses SSL / TLS connections between the client and server. This ensures a secure communication channel providing confidentiality and integrity of data during transport. The SSL / TLS connection used with CBT is a server-authenticated connection (not mutually authenticated) since the user will authenticate to the server using CBT Logon. CBT has an optional feature which adds strong end-to-end encryption between the client and server at the application layer (SSL protects only the transport layer). SSL / TLS is however sufficient for most purposes.

The messages submitted from the applets will typically be received by a servlet handler (or CGI program) in the service provider's web application. Here the application will invoke CBT APIs to process and validate the messages. The logon and sign messages from the applets comply with the XML dSig standard from W3C [1]. Processing these messages typically includes digital signature verification, certificate and certificate path validation, trust validation and revocation checks of the certificate against relevant certificate authorities. For most scenarios, service providers will only need the CBT Signature Validation API which is capable of performing all necessary validations on messages produced by the applets.

3.2 Logon Flow

In this section we will show the typical flow of events and messages during logon. It is assumed that the web application will itself create the user session.

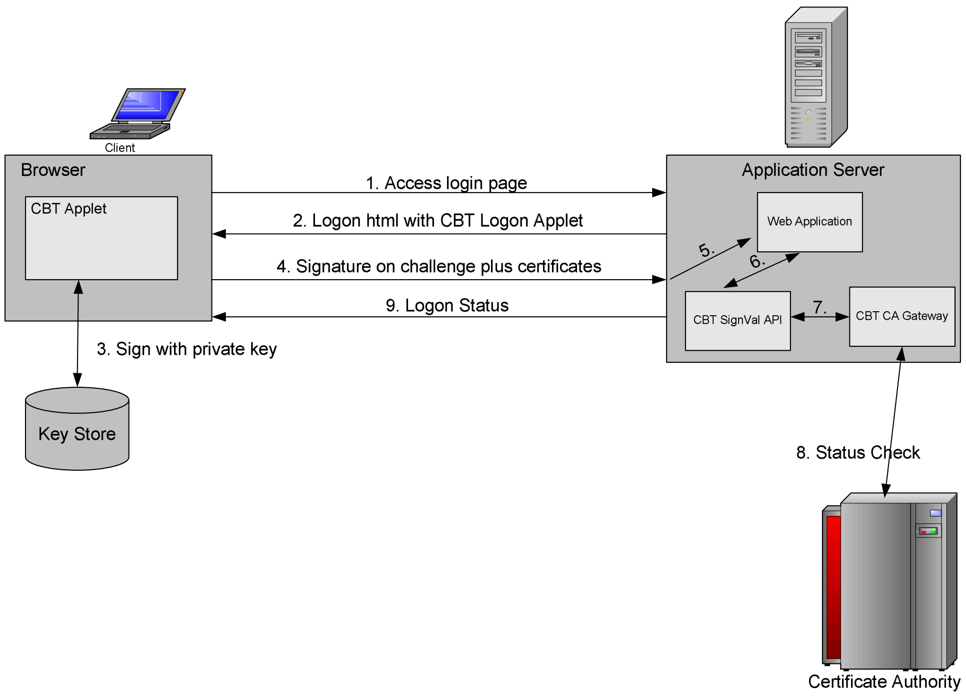


Figure 2: Logon Flow

The sequence is as follows:

1. The user requests (or is redirected to) the logon page for the web application or portal.
2. The server sends a logon html page which calls the CBT logon applet. A challenge value is passed to the logon applet in the HTML page which the user must sign.
3. The CBT Logon Applet displays its GUI which allows the user to select a key for signing and enter a password.
4. The Logon Applet produces a signature with the user's private key over the challenge and submits it to the server.
5. The web application (e.g. servlet handler) receives the logon message sent from the client.
6. The CBT Signature validation API is invoked to validate the logon message.
7. If the logon message includes certificates the CBT CA Gateway is invoked.
8. The gateway calls Certificate Authorities in order to check the revocation status of the certificate(s).
9. If the logon message validates successfully, a session is created in the application or application server, and e.g. the welcome page is returned to the user.

3.2.1 Integrating CBT Logon with Middleware

In the above logon sequence we have assumed that the user session is created in the web application itself in response to a successful logon. In many environments however the service provider will use the session management and access control capabilities of the application server and/or use security products such as Tivoli Access Manager. This allows externalization of the security policy from the business applications which has many advantages. The following sections briefly describe how CBT Logon can be integrated with such middleware components. We will highlight the Tivoli Access Manager product and J2EE application servers but other options (e.g. .Net) naturally exist.

3.2.1.1 Integrating CBT Logon with Tivoli Access Manager

Tivoli Access Manager for e-business (TAM) is IBM's software product for web Single Sign-On (SSO) and centralized access control. For details on the Tivoli Access Manager product itself we refer to IBM's product web sites. Although the following focuses on Access Manager, similar integration can be performed with most competing products (e.g. Netegrity Siteminder).

Access Manager for e-business normally utilizes a reverse proxy component called WebSEAL placed between the Internet and the production network with application servers. WebSEAL authenticates and authorizes all incoming requests before serving them. Further, Access Manager contains a policy server where the central access control policy is defined and an authorization server which handles authorization decisions.

CBT is integrated with Tivoli Access Manager using the External Authentication Interface (EAI) which delegates the authentication of the user to an external application – the CBT Login Application.

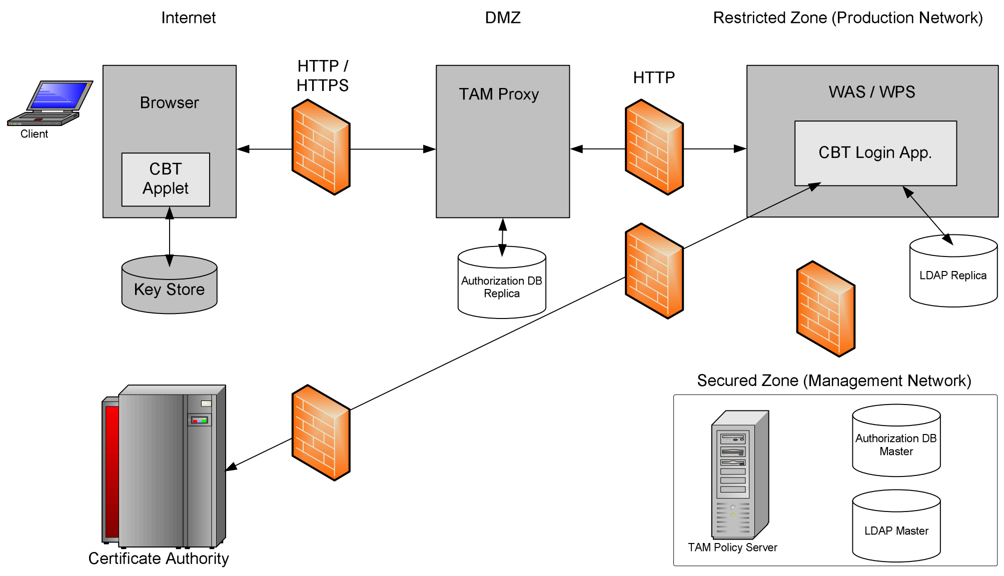


Figure 3: Integration with Tivoli Access Manager

Integration is usually done by modifying the standard TAM logon page to run the CBT Logon Applet. Once the logon message posted from the browser hits the CBT Login Application it calls the CBT server components for validation and mapping of the user's credential (e.g. certificate) to a local user-ID known by TAM. After validation, the local user-ID is given to the proxy which creates a session and handles subsequent authorization decisions. Note that the user-ID can be mapped to generic roles such that the each user does not have to be created in the registry in advance.

For further details on integration between CBT and TAM, please contact the IBM Crypto Competence Center Copenhagen (ccc@dk.ibm.com).

3.2.1.2 Integrating CBT Logon with J2EE Application Servers

In many cases a service provider will use the capabilities of the application server to perform session management and access control. For example, J2EE application servers such as IBM WebSphere Application Server or WebSphere Portal Server provide a wide range of security features that can be leveraged.

Briefly, J2EE containers provide two different ways of defining access control restrictions for its applications: declarative security and programmatic security. Declarative security is the means by which an application's security policies can be expressed externally to the application code. At application assembly time, security policies are defined in an application's deployment descriptor.

Programmatic security is used when an application must be "security-aware". For instance, a method might need to know the identity of the caller for logging purposes, or it might perform additional actions based on the caller's role.

Before access control restrictions can be enforced for a web application, the container needs to know the identity of the user. The J2EE specification defines several types of authentication methods including basic authentication, digest authentication, form-based authentication and client certificate authentication. One might think that CBT should be integrated with client certificate authentication but this is actually not the case. The reason is that this authentication is bound to TLS / SSL which CBT does normally not facilitate; remember that CBT Logon occurs at the application layer as a signed challenge or timestamp. Furthermore, the keys generated by CBT are usually not available to the browser for use in SSL / TLS establishment.

Instead, CBT can be integrated with form-based login. Every J2EE web module can specify a login HTML page containing a form normally used for user-ID and password entry. This page can be modified to start the CBT Logon Applet and hide the normal logon form (it is visually replaced by the Logon Applet's GUI). The logon form has fixed variables defined to carry the user-ID and password and one of them can instead be used to carry the logon message from the applet.

The logon form is normally posted to a servlet called `j_security_check`. With a servlet filter it is possible to intercept this processing either before or after the normal servlet is invoked (see [2] for additional details). In a servlet filter the CBT APIs can be invoked to validate the message and map the user certificate to a locally defined user-ID. An alternative to using a servlet filter is to define a custom JAAS Login module and configure it in the application server. This module will be invoked to validate the logon and should retrieve the user-ID and password via callback handlers (which will return the posted form data). The JAAS login module should then perform signature verification by calling the CBT APIs, create a Principal corresponding to the local user-ID, and add this principal to the Subject provided to the module.

A J2EE server uses some form of registry (e.g. LDAP) to lookup user-IDs and their associated groups. The container will use this information when taking authorization decisions with declarative or programmatic security (the groups in the registry are mapped to roles defined in the application as part of deployment).

3.3 Sign Flow

Signing of agreements by the user is required in many applications. For example, the user wants to perform a money transfer in an Internet banking application or wants to submit changes to his tax declaration in an eGovernment application. Before such sensitive operations are allowed the service provider will generally require the user to express his consent by signing an agreement (message) representing the transaction.

The figure below shows the flow when a message is signed:

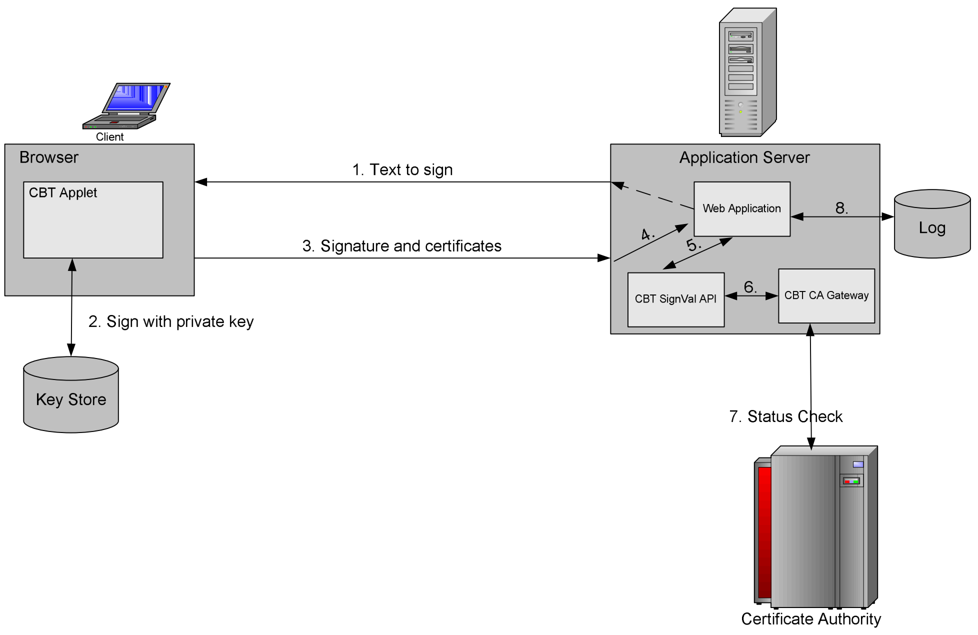


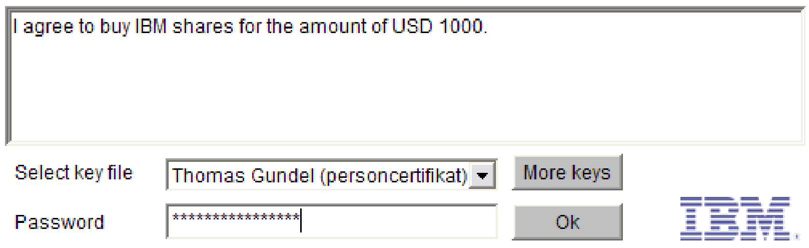
Figure 4: Sign Flow

The events in the figure are:

1. The server application sends an html page calling the CBT Sign Applet and provides as input the agreement to be signed.
2. The CBT Sign Applets starts and presents the text to be signed to the user. The user selects signing key and inputs his signature password. The applet signs the agreement with the user's private key.
3. The signature (and user certificates) is sent to the server for processing.
4. The signed message hits the web application's handler.
5. The web application calls the CBT Signature Validation API to process the signed message.
6. If the signature contains any certificates the CBT CA Gateway is called to validate the certificates.
7. The CBT CA Gateway communicates with the Certificate Authority.
8. The signature and the result of the validation are logged by the application in a database. The log will serve as evidence in case of later dispute and should be tightly managed. If all validations succeed the application can safely perform the pending transaction.

Note that unlike logon, signing is a pure application phenomenon which normally does not require integration with middleware components.

The CBT Sign Applet supports presentation and signing of several different representations of an agreement. The standard format is pure text (containing tabs and newline characters) which will be shown as an integrated part of the applet GUI. An example of this mode is shown below:


I agree to buy IBM shares for the amount of USD 1000.

Select key file: Thomas Gundel (personcertifikat)

Password: ***************

Figure 5: Example of CBT Sign Applet

In the above example, the signed text is very short. If longer text is needed, the applet will enable scroll bars to allow the user to browse through the text. Note also that the size of the applet window is chosen by the web application (by setting applet tags in the HTML page).

Furthermore, the CBT Sign Applet supports agreements being shown by an external program (e.g. a PDF viewer). This can be an advantage for presentation of lengthy or visually advanced agreements. The external program will be called by CBT to render / present the agreement but the CBT applet will still handle the keys and signatures.

To enable workflows with digital signatures on documents the CBT applets also support including attached files in the signature. E.g. if a client or employee have filled out a form or spreadsheet offline it is possible to attach the file and have it included in the signature.

3.3.1 Reducing the necessary number of password/PIN entries

It is recommended that the user enters the password or PIN each time a signature is made. If such a policy is not enforced then non-repudiation can be difficult to argue and it makes it easier for Trojans to sign fraudulent transactions behind the users back.

To avoid entering the password many times it is recommended that the application is designed such that it is possible to sign documents in batches. E.g. if a user makes 3 money transfers to different accounts, he should have the option of deferring the signing step when he enters the information, and then sign all 3 transfers in one go. This is a matter of how the application is designed.

3.4 Registration

A pre-requisite for performing logon and signing with digital signatures is obviously that digital signature keys (and optionally certificates) are available to the user.

Some service providers will use and rely on credentials (i.e. signature keys and certificates) provided by external parties such as certificate authorities. These service providers will not need to implement an expensive registration process (or at least only a lightweight one) and can provide instant access to their applications (e.g. achieving Single Sign-On).

However, other service providers may not be able to use this model; they may want tight control over the registration process to be able to control both security and policy aspects. Furthermore, in many cases no third-party credentials exist which the service provider is both willing to trust and which are technically sufficient. A common example of service providers who normally implement their own registration processes are financial companies.

CBT contains functionality for building a registration application where the users have digital signature key pairs generated and optionally an X.509 certificate issued. If the user only needs to logon and sign towards one service provider (owning the registration application) there is no need to issue X.509 certificates which would allow third parties to trust the user's public key. If the user however must be able to sign towards third parties CBT provides functionality for issuance of X.509 certificates as well. In fact, CBT is able to generate standard certificate requests (PKCS#10) and communicate with an external (or internal) certificate authority. Note that CBT does not implement CA functionality itself.

3.4.1 Registration without Certificate Generation

In this section a basic registration flow where no certificate is issued is described. Instead of issuing a certificate, the user's public key is stored in a central database at the service provider to facilitate later signature verification.

Figure 6 below illustrates the sequence:

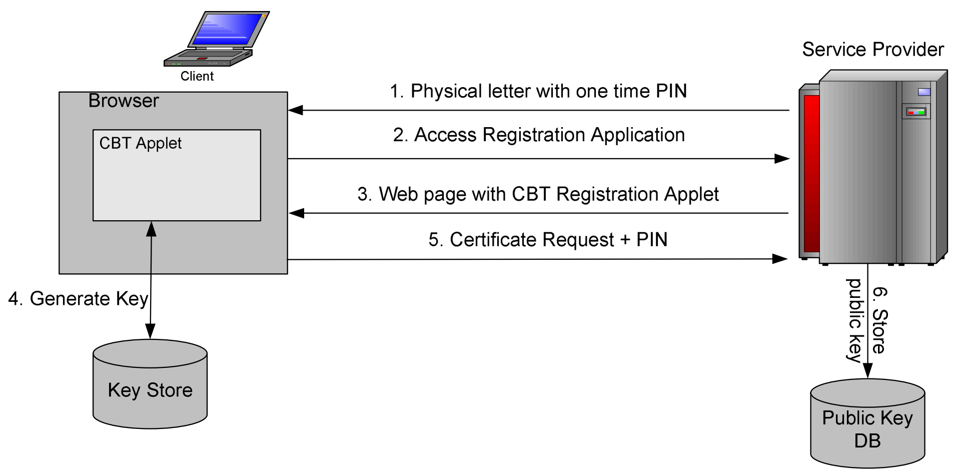

Client
Browser
CBT Applet
4. Generate Key
Key Store
Service Provider
1. Physical letter with one time PIN
2. Access Registration Application
3. Web page with CBT Registration Applet
5. Certificate Request + PIN
6. Store public key
Public Key DB

Figure 6: Registration Flow without Certificate Generation

In the registration flow there is a need for an initial user authentication to bootstrap the system since no signature keys are present yet. There are several ways of handling this in practice - in the example we assume that the user has received a physical, secure PIN letter from the service provider with a one time PIN. IBM Crypto Competence Center can provide components for secure generation and printing of such one time PINs.

The sequence is:

1. The user receives the (physical) letter with a one time PIN.
2. The user starts his browser and visits the registration web application at the service provider.
3. A registration HTML page is returned and starts the CBT KeygenRegister Applet.
4. The applet generates a signature key pair on the user's computer, inputs the one time PIN, and sends a signed certificate request containing PIN and public key.
5. The service provider receives the certificate request and verifies the one time PIN and signature.
6. The public key is stored in a database and can be used for later verification of signatures generated with the corresponding private key.

3.4.2 Registration using a Certificate Authority

In this section we will extend the previous flow to show how certificates can be issued. The first five steps are identical:

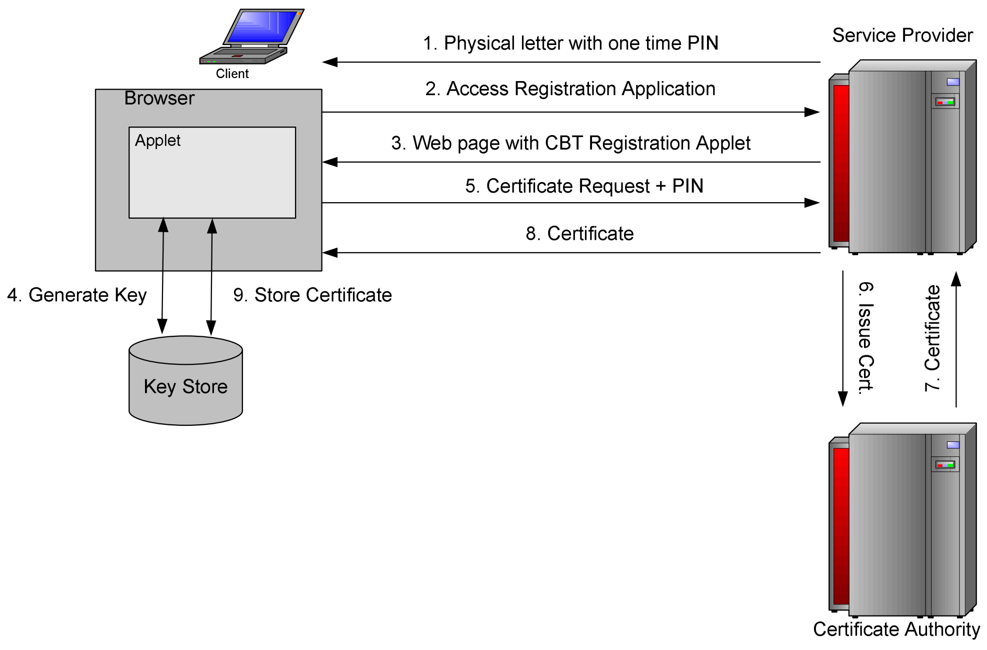


Figure 7: Registration using a CA

The next steps are:

6. Instead of processing the certificate request itself, the service provider forwards it to a certificate authority (which can be internal as well as external). This communication should be authenticated e.g. using SSL / TLS.
7. The certificate authority receives the certificate request, processes it and returns an X.509 certificate (or path) asserting the binding between the user identity and public key.
8. The certificate (or path) is sent to the client via an HTML page which starts the CBT CertReceive Applet. The applet gets as input the certificate (or path).
9. The certificate (or path) is stored in the client's key store.

Here, the service provider acts as a Registration Authority (RA) and handles the identification and authentication of the user. Note that the above setup is just a simplified example and that many variants of CA integration can occur.

The CBT Sign Applet will include any certificate path information present in the client key store as meta data in the signature when signing towards another service provider. This allows the service provider to trust the message based on trust in the issuing certificate authority.

4. Requirements for a CBT Environment

This chapter describes some of the platform requirements for CBT. The description will be high-level and is provided to give an idea of which environments CBT can operate in. For further details we refer to the CBT product manuals.

4.1 Client Platform Requirements

Since the CBT Applets run in the user's browser there must be a Java Virtual Machine installed on the client platform. CBT runs on virtually any VM (including Apple, Microsoft, Sun, and IBM) and only requires version 1.1 of Java.

For communication with the server, CBT Applets normally use JavaScript and LiveConnect. These technologies must be supported by the browser.

The client platform requirements can be met on a wide range of platforms including all versions of Microsoft Windows, most Linux variants and MacIntosh. The list of supported browsers includes Microsoft Internet Explorer, Netscape, Mozilla, Opera, Firefox, Safari and others.

4.2 Server Platform Requirements

The CBT Server components are normally delivered as Java APIs and only require a Java Virtual Machine supporting version 1.3 of Java. They can run on almost any J2EE application server.

Selected parts of the CBT Server functionality can be delivered as C APIs to support .Net application servers and COBOL APIs to support IBM mainframes. Please contact the Crypto Competence Center (ccc@dk.ibm.com) for further details.

Some of the CBT Server APIs support IBM cryptographic hardware for acceleration and improved protection of encryption keys. On Windows and Unix platforms the IBM 4764 Crypto Card can be supported and on IBM mainframes ICSF can be supported.

4.3 Supported Key Stores

The CBT Thin Client Applets support a wide range of key stores on the client including:

Key store type	**Comment**
CBT Legacy	These are legacy CBT key files which can hold user information, a key pair and a certificate. The key files are encrypted with a triple-DES key derived from the user's password.
PKCS#12	This encrypted key file format is a de-facto standard supported by many applications and platforms.
PKCS#11	PKCS#11 is a standard interface to cryptographic devices. It can be used to interface with smart cards, USB crypto tokens and dedicated cryptographic processors. PKCS#11 is available on many operating systems.
MSCAPI	This is Microsoft's interface to crypto services from applications. CBT can use MSCAPI to access crypto service providers (CSPs) belonging to smart cards and USB crypto tokens, e.g. the Aladdin eToken. The concept is somewhat similar to PKCS#11 but is restricted to the Windows platform.
SmartCard over PC/SC	CBT can access smartcards directly over the PC/SC 1.0 interface. The Secure PIN Entry feature of PC/SC 2.0 is also supported. Supported platforms are Windows, Linux and Mac.

4.4 Compression of Applets

The CBT Applet archives are normally code signed and placed on the server as part of the web application. When a web page refers to an applet, the archive will be downloaded to the user's browser as needed.

This approach has many advantages including relief of installing code on the client computers and easy access to upgrade the applets. However, the applet archive will be downloaded once per browser session which incurs an overhead in range of 150-400 Kb. If this overhead is prohibitive the CBT applet archives can be compressed even more by taking advantage of the improved compression features available in later Java versions. See
http://java.sun.com/javase/6/docs/technotes/guides/deployment/deployment-guide/pack200.html

References

[1]: “XML-Signature Syntax and Processing”, W3C Recommendation.

[2]: “WebSphere Application Server V6 Security Handbook”, IBM RedBook.