DATA SECURITY MANAGEMENT

MITIGATING E-BUSINESS SECURITY RISKS: PUBLIC KEY INFRASTRUCTURES IN THE REAL WORLD

Douglas C. Merrill and Eran Feigenbaum

INSIDE

Network Security: The Problem; Why Cryptography is Useful; Using a PKI to Authenticate to an Application; Components of a PKI; Other PKI Benefits: Reduced Sign-on; PKI in Operation; Implementing a PKI

INTRODUCTION

Many organizations want to get involved with electronic commerce — or are being forced to become an E-Business by their competitors. The goal of this business decision is to realize bottom-line benefits from their information technology investment, such as more efficient vendor interactions and improved asset management. Such benefits have indeed been realized by organizations, but so have the associated risks, especially those related to information security. Managed risk is a good thing, but risk for its own sake, without proper management, can drive a company out of existence. More and more corporate management teams — even up to the board of directors level — are requiring evidence that security risks are being managed. In fact, when asked about the major stumbling blocks to widespread adoption of electronic business, upper management pointed to a lack of security as a primary source of hesitation.

PAYOFF IDEA

Although this article focuses on a technology — PKI — it is important to realize that large implementations involve organizational transformation. Many nontechnical aspects are integral to the success of a PKI implementation, including organizational governance, performance monitoring, stakeholder management, and process adjustment. Failing to consider these aspects greatly increases the risk of project failure, although many of these factors are outside of the domain of information security. Successful PKI implementations involve not only information security personnel, but also business unit leaders and senior executives to ensure that these nontechnical aspects are handled appropriately.

An enterprisewide security architecture, including technology, appropriate security policies, and audit trails, can provide reasonable measures of risk management to address senior management concerns about E-Business opportunities. One technology involved in enterprisewide security architectures is public key cryptography, often implemented in the form of a public key infrastructure (PKI). This article describes several hands-on examples of PKI, including business cases and implementation plans. The authors attempt to present detail from a very practical, hands-on approach, based on their experience implementing PKI and providing large-scale systems integration services. Several shortcuts are taken in the technical discussions to simplify or clarify points, while endeavoring to ensure that these did not detract from the overall message.

Although this article focuses on a technology — PKI — it is important to realize that large implementations involve organizational transformation. Many nontechnical aspects are integral to the success of a PKI implementation, including organizational governance, performance monitoring, stakeholder management, and process adjustment. Failing to consider these aspects greatly increases the risk of project failure, although many of these factors are outside the domain of information security. In the authors' experience, successful PKI implementations involve not only information security personnel, but also business unit leaders and senior executives to ensure that these nontechnical aspects are handled appropriately.

NETWORK SECURITY: THE PROBLEM

As more and more data is made network-accessible, security mechanisms must be put in place to ensure only authorized users access the data. An organization does not want its competitor to read, for example, its internal pricing and availability information. Security breaches often arise through failures in authentication. Authentication is the process of identifying an individual so that one can determine the individual's access privileges. To start my car, I must authenticate myself to my car. When I start my car, I have to "prove" that I have the required token — the car key — before my car will start. Without a key, it is difficult to start my car. However, a car key is a poor authentication mechanism — it is not that difficult to get my car keys, and hence be me, at least as far as my car is concerned. In the everyday world, there are several stronger authentication mechanisms, such as presenting one's driver's license with a picture. People are asked to present their driver's licenses at events ranging from getting on a plane to withdrawing large amounts of money from a bank. Each of these uses involves comparing the image on the license to one's appearance. This strengthens the authentication process by requiring two-factor authentication — an attacker must not only have my license, but he must also resemble me. In the electronic world, it is far

more difficult to get strong authentication: a computer cannot, in general, check to be sure a person looks like the picture on their driver's license. Typically, a user is required to memorize a username and password. These username and password pairs must be stored in operating system-specific files, application tables, and the user's head (or desk). Any individual sitting at a keyboard that can produce a user's password is assumed to be that user.

Traditional implementations of this model, although useful, have several significant problems. When a new user is added, a new username must be generated and a new password stored on each of the relevant machines. This can be a significant effort. Additionally, when a user leaves the company, that user's access must be terminated. If there are several machines and databases, ensuring that users are completely removed is not easy. The authors' experience with PricewaterhouseCoopers LLP (PricewaterhouseCoopers) assessing security of large corporations suggests that users are often not removed when they leave, creating significant security vulnerabilities.

Additionally, many studies have shown that users pick amazingly poor passwords, especially when constrained to use a maximum of eight characters, as is often the case in operating system authentication. For example, a recent assessment of a FORTUNE 50 company found that almost 10 percent of users chose a variant of the company's logo as their password. Such practices often make it possible for an intruder to simply guess a valid password for a user and hence obtain access to all the data that user could (legitimately) view or alter.

Finally, even if a strong password is selected, the mechanics of network transmission make the password vulnerable. When the user enters a username and password, there must be some mechanism for getting the identification materials to the server itself. This can be done in a variety of ways. The most common method is to simply transmit the username and password across the network. However, this information can be intercepted during transmission using commonly available tools called "sniffers." A sniffer reads data as it passes across a network — data such as one's username and password. After reading the information, the culprit could use the stolen credentials to masquerade as the legitimate user, attaining access to any information that the legitimate user could access. To prevent sniffing of passwords, many systems use cryptography to hide the plaintext of the password before sending it across the network. In this event, an attacker can still sniff the password off the network, but cannot simply read its plaintext; rather, the attacker sees only the encrypted version. The attacker is not entirely blocked, however. There are publicly available tools to attack the encrypted passwords using dictionary words or brute-force guessing to get the plaintext password from the encrypted password. These attacks exploit the use of unchanging passwords and functions. Although this requires substantial effort, many

demonstrated examples of accounts being compromised through this sort of attack are known.

These concerns — lack of updates after users leave, poor password selection, and the capability to sniff passwords off networks — make reliance on username and password pairs for remote identification to business-critical information unsatisfactory.

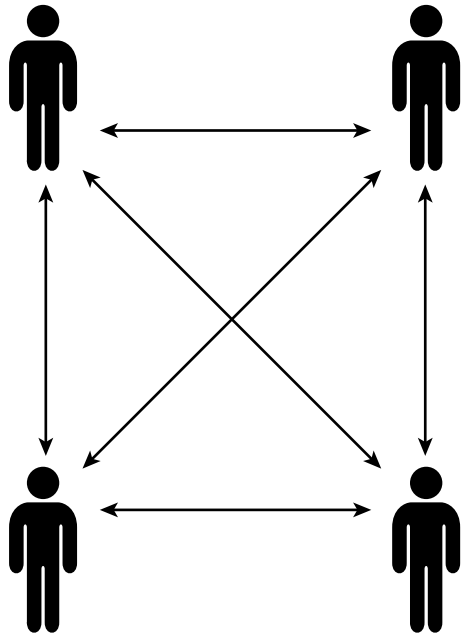
WHY CRYPTOGRAPHY IS USEFUL

Cryptography (from the Greek for "secret writing") provides techniques for ensuring data integrity and confidentiality during transport and for lessening the threat associated with traditional passwords. These techniques include codes, ciphers, and steganography. This article only considers ciphers; for information on other types of cryptography, one could read Bruce Schneier's *Applied Cryptography* or David Kahn's *The Codebreakers*. Ciphers use mathematics to transform plaintext into "ciphertext." It is very difficult to transform ciphertext back into plaintext without a special key. The key is distributed only to select individuals. Anyone who does not have the key cannot read or alter the data without significant effort. Hence, authentication becomes the question, "does this person have the expected key?" Additionally, the property that only a certain person (or set of people) has access to a key implies that only those individuals could have done anything to an object encrypted with that key. This so-called "nonrepudiation" provides assurance about an action that was performed, such as that the action was performed by John Doe, or at a certain time, etc.

There are two types of ciphers. The first method is called secret key cryptography. In secret key cryptography, a secret — a password — must be shared between sender and recipient in order for the recipient to decrypt the object. The best-known secret key cryptographic algorithm is the Data Encryption Standard (DES). Other methods include IDEA, RC4, Blowfish, and CAST. Secret key cryptography methods are, in general, very fast, because they use fairly simple mathematics, such as binary additions, bit shifts, and table lookups.

However, transporting the secret key from sender to recipient — or recipients — is very difficult. If four people must all have access to a particular encrypted object, the creator of the object must get the same key to each person in a safe manner. This is difficult enough. However, an even more difficult situation occurs when each of the four people must be able to communicate with each of the others without the remaining individuals being able to read the communication (see Exhibit 1). In this event, each pair of people must share a secret key known only to those two individuals. To accomplish this with four people requires that six keys be created and distributed. With ten people, the situation requires 45 key exchanges (see Exhibit 2). Also, if keys were compromised —

EXHIBIT 1 — Four People Require Six Keys

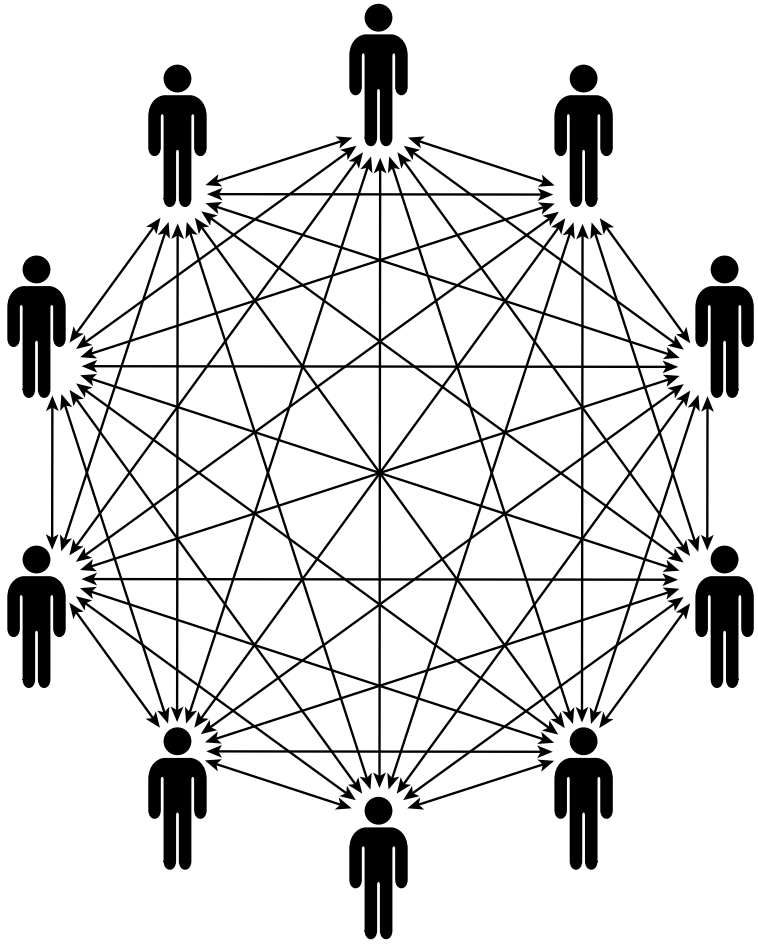


such as would happen when a previously authorized person leaves the company — all the keys known to the departing employee must be changed. Again, in the four-person case, the departure requires three new key exchanges; nine are required in the ten-person case. Clearly, this will not work for large organizations with hundreds or thousands of employees.

In short, secret key cryptography has great power, employs fairly simple mathematics, and can quickly encrypt large volumes of data. However, its Achilles heel is the problem of key distribution and maintenance.

This Achilles heel led a group of mathematicians to develop a new paradigm for cryptography — asymmetric cryptography, also known as public key cryptography. Public key cryptography lessens the key distribution problem by splitting the encryption key into a public portion — which is given out to anyone — and a secret component that must be controlled by the user. The public and private keys, which jointly are called a key pair, are generated together and are related through complex mathematics. In the public key model, a sender looks up the recipient's public keys, typically stored in certificates, and encrypts the document using those public keys. No previous connection between sender and recipient is required, because only the recipient's public key is needed for

EXHIBIT 2 — Ten People Require 45 Keys



secure transmission, and the certificates are stored in public databases. Only the private key that is associated with the public key can decrypt the document. The public and private keys can be stored as files, as entries in a database, or on a piece of hardware called a token. These tokens are often smart cards that look like credit cards but store user keys and are able to perform cryptographic computations far more quickly than general-purpose CPUs.

There are several public key cryptographic algorithms, including RSA, Diffie-Hellman, and Elliptic Curve cryptography. These algorithms relay on the assumption that there are mathematical problems that are easy to perform but difficult to do in reverse. To demonstrate this to yourself, cal-

culate 11 squared (11^2). Now calculate the square root of 160. The square root is a bit more difficult, right? This is the extremely simplified idea behind public key cryptography. Encrypting a document to someone is akin to squaring a number, while decrypting it without the private key is somewhat like taking the square root. Each of the public key algorithms uses a different type of problem, but all rely on the assumption that the particular problem chosen is difficult to perform in reverse without the key.

Most public key algorithms have associated "signature" algorithms that can be used to ensure that a piece of data was sent by the owner of a private key and was unchanged in transit. These digital signature algorithms are commonly employed to ensure data integrity, but do not, in and of themselves, keep data confidential.

Public key cryptography can be employed to protect data confidentiality and integrity while it is being transported across the network. In fact, Secure Sockets Layer (SSL) is just that: a server's public key is used to create an encrypted tunnel across which World Wide Web (WWW) data is sent. SSL is commonly used for WWW sites that accept credit card information; in fact, the major browsers support SSL natively, as do most Web servers. Unfortunately, SSL does not address all the issues facing an organization that wants to open up its data to network access. By default, SSL authenticates only the server, not the client. However, an organization would want to provide its data only to the correct person; in other words, the whole point of this exercise is to ensure that the client is authenticated.

The SSL standards provide methods to authenticate not only the server, but also the client. Doing this requires having the client side generate a key pair and having the server check the client keys. However, how can the server know that the supposed client is not an imposter even if the client has a key pair? Additionally, even if a key does belong to a valid user, what happens when that user leaves the company, or when the user's key is compromised? Dealing with these situations requires a process called key revocation. Finally, if a user generates a key pair, and then uses that key pair to, for example, encrypt attachments to business-related electronic mail, the user's employer may be required by law to provide access to user data when served with a warrant. For an organization to be able to answer such a warrant, it must have "escrowed" a copy of the users' private keys — but how could the organization get a copy of the private key, since the user generated the pair?

Public key cryptography has a major advantage over secret key cryptography. Recall that secret key cryptography required that the sender and recipient share a secret key in advance. Public key cryptography does not require the sharing of a secret between sender and recipients, but is far slower than secret key cryptography, because the mathematics involved are far more difficult.

Although this simplifies key distribution, it does not solve the problem. Public key cryptography requires a way to ensure that John Doe's public key in fact belongs to him, not to an imposter. In other words, anyone could generate a key pair and assert that the public key belongs to the President of the United States. However, if one were to want to communicate with the President securely, one would need to ensure that the key was in fact his. This assurance requires that a trusted third party assert a particular public key does, in fact, belong to the supposed user. Providing this assurance requires additional elements, which, together make up a public key infrastructure (PKI).

The next section describes a complete solution that can provide data confidentiality and integrity protection for remote access to applications. Subsequent sections point out other advantages yielded by the development of a full-fledged infrastructure.

USING A PKI TO AUTHENTICATE TO AN APPLICATION

Let us first describe, at a high level, how a WWW-based application might employ a PKI to authenticate its users (see Exhibit 3). The user directs her WWW browser to the (secured) WWW server that connects to the application. The WWW page uses the form of SSL that requires both

EXHIBIT 3 — Using a PKI to Authenticate Users

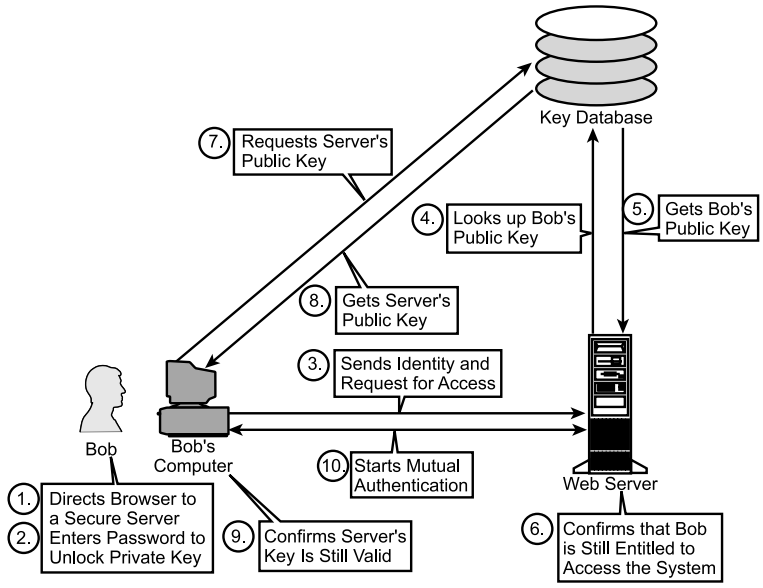
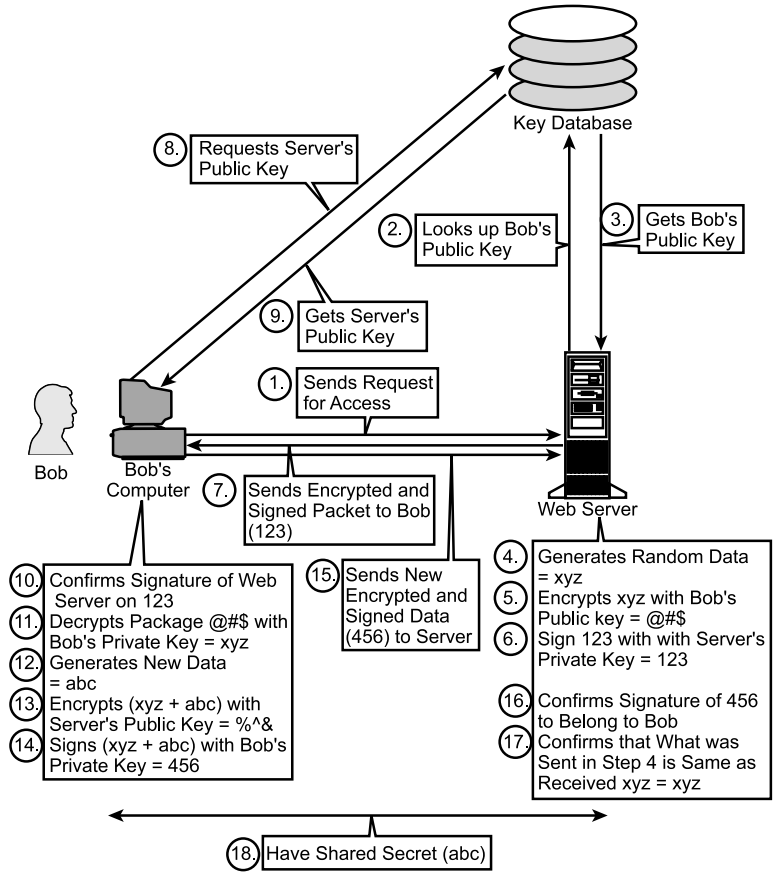


EXHIBIT 4 — Mutual Authentication



server and client authentication. The user must unlock her private key; this is done by entering a password that decrypts the private key. The server asks for the identity of the user, and looks up her public key in a database. After retrieving her public key, the server checks to be sure that the user is still authorized to access the application system, by checking to be sure that the user's key has not been revoked. Meanwhile, the client accesses the key database to get the public key for the server and checks to be sure it has not been revoked. Assuming that the keys are still valid, the server and client engage in mutual authentication.

There are several methods for mutual authentication. Regardless of approach, mutual authentication requires several steps; the major difference between methods is the order in which the steps occur. Exhibit 4 presents a simple method for clarity. First, the server generates a piece of

random data, encrypts it with the client's public key, and signs it with its own private key. This encrypted and signed data is sent to the client, who checks the signature using the server's public key and decrypts the data. Only the client could have decrypted the data, because only the client has access to the user's private key; and only the server could have signed the data, because to sign the encrypted data, the server requires access to the server's private key. Hence, if the client can produce the decrypted data, the server can believe that the client has access to the user's private key. Similarly, if the client verifies the signature using the server's public key, the client is assured that the server signed the data. After decrypting the data, the client takes it, along with another piece of unique data, and encrypts both with the server's public key. The client then signs this piece of encrypted data and sends it off to the server. The server checks the signature, decrypts the data, checks to be sure the first piece of data is the same as what the server sent off before, and gathers the new piece of data. The server generates another random number, takes this new number along with the decrypted data received from the client, and encrypts both together. After signing this new piece of data, the resulting data is sent off to the client. Only the client can decrypt this data, and only the server could have signed it. This series of steps guarantees the identity of each party. After mutual authentication, the server sends a notice to the log server, including information such as the identity of the user, client location, and time.

Recall that public key cryptography is relatively slow; the time required to encrypt and decrypt data could interfere with the user experience. However, if the application used a secret key algorithm to encrypt the data passing over the connection, after the initial public key authentication, the data would be kept confidential to the two participants, but with a lower overhead. This is the purpose of the additional piece of random data in the second message sent by the server. This additional piece of random data will be used as a session key — a secret shared by client and server. Both client and server will use the session key to encrypt all network transactions in the current network connection using a secret key algorithm such as DES, IDEA, or RC4. The secret key algorithm provides confidentiality and integrity assurance for all data and queries as they traverse the network without the delay required by a public key algorithm. The public key algorithm handles key exchange and authentication. This combination of both a public key algorithm and a private key one offers the benefits of each.

How did these steps ensure that both client and server were authenticated? The client, after decrypting the data sent by the server, knows that the server was able to decrypt what the client sent, and hence knows that the server can access the server's private key. The server knows that the client has decrypted what it sent in the first step, and thus knows that the client has access to the user's private key. Both parties have authenticat-

ed the other, but no passwords have traversed the network, and no information that could be useful to an attacker has left the client or server machines.

Additionally, the server can pass the authentication through to the various application servers without resorting to insecure operating system-level trust relationships, as is often done in multi-system installations. In other words, a user might be able to leverage the public key authentication to not only the WWW-based application, but also other business applications. More details on this reduced sign-on functionality are provided in a later section.

COMPONENTS OF A PKI

The behavior described in the example above seemed very simple, but actually involved several different entities behind the scenes. As is so often the case, a lot of work must be done to make something seem simple. The entities involved here include a certificate authority, registration authorities, directory servers, various application programming interfaces and semi-custom development, third-party applications, and hardware. Some of these entities would be provided by a PKI vendor, such as the CA, RA, and a directory server, but other components would be acquired from other sources. Additionally, the policies that define the overall infrastructure and how the pieces interact with each other and the users are a central component. This section describes each component and tells why it is important to the overall desired behavior.

The basic element of a PKI is the certificate authority. One of the problems facing public key solutions is that anyone can generate a public key and claim to be anyone they like. For example, using publicly available tools, one can generate a public key belonging, supposedly, to the President of the United States. The public key will say that it belongs to the President, but it actually would belong to an imposter. It is important for a PKI to provide assurance that public keys actually belong to the person who is named in the public key. This is done via an external assurance link; to get a key pair, one demonstrates to a human that they are who they claim to be. For example, the user could, as part of the routine on the first day of employment, show his driver's license to the appropriate individual, known as a registration authority. The registration authority (RA) generates a key pair for the individual and tells the certificate authority (CA) to attest that the public key belongs to the individual. The CA does this attestation by signing the public key with the CA's private key. All users trust the CA. Because only the CA could access the CA's private key, and the private key is used to attest to the identity, all will believe that the user is in fact who the user claims to be. Thus, the CA (and associated RA) is required in order for the PKI to be useful, and any compromise of the CA's key is fatal for the entire PKI. CAs and RAs are

usually part of the basic package bought from a PKI vendor. An abridged list of PKI vendors (in alphabetical order) includes Baltimore, Entrust Technologies, RSA Security, and Verisign.

When one user (or server) wants to send an encrypted object to another, the sender must get the recipient's public key. For large organizations, there can be thousands of public keys, stored as certificates signed by the CA. It does not make sense for every user to store all other certificates, due to storage constraints. Hence, a centralized storage site (or sites) must store the certificates. These sites are databases, usually accessed via the Lightweight Directory Access Protocol (LDAP), and normally called directory servers. A directory server will provide access throughout the enterprise to the certificates when an entity requires one. There are several vendors for LDAP directories, including Netscape, ICL, Novell, and Microsoft.

There are other roles for directory servers, including escrow of users' private keys. There are several reasons why an organization might need access to users' private keys. If an organization is served by a warrant, it may be required to provide access to encrypted objects. Achieving this usually involves having a separate copy of users' private keys; this copy is called an "escrowed" key. LDAP directories are usually used for escrow purposes. Obviously, these escrow databases must be extremely tightly secured, because access to a user's private key compromises all that user's correspondence and actions. Other reasons to store users' private keys include business continuity planning and compliance monitoring.

When a sender gets a recipient's public key, the sender cannot be sure that the recipient still works for the organization, and does not know if someone has somehow compromised that key pair. Human resources, however, will know that the recipient has left the organization and the user may know that the private key has been compromised. In either case, the certificate signed by the CA — and the associated private key — must be revoked. Key revocation is the process through which a key is declared invalid. Much as it makes little sense for clients to store all certificates, it is not sensible for clients to store all revoked certificates. Rather, a centralized database — called a certificate revocation list (CRL) — should be used to store revoked certificates. The CRL holds identifiers for all revoked certificates. Whenever an entity tries to use a certificate, it must check the CRL in order to ensure that the certificate is still valid; if an entity is presented a revoked certificate, it should log the event as a possible attack on the infrastructure. CRLs are often stored in LDAP databases, in data structures accessible through Online Certificate Status Processing (OCSP), or on centralized revocation servers, as in Valicert's Certificate Revocation Tree service. Some PKIs have ability to check CRLs, such as Entrust's Entelligence client, but most rely on custom software development to handle CRL checking. Additionally, even for PKIs supporting CRL checking, the capabilities do not provide access to other

organization's CRLs — only a custom LDAP solution or a service such as, for example, Valicert's can provide this inter-organization (or inter-PKI) capability.

Off-the-shelf PKI tools are often insufficient to provide complete auditing, dual authentication, CRL checking, operating system integration, and application integration. To provide these services, custom development must be performed. Such development requires that the application and PKI both support application programming interfaces (APIs). The API is the language that the application talks and through which the application is extended. There are public APIs for directory servers, operating system authentication, CRL checking, and many more functions. It is very common for applications to support one or more APIs. Many PKI vendors have invested heavily in the creation of toolkits — notably, RSA Security, Entrust Technologies, and Baltimore.

For both performance and security reasons, hardware cryptographic support can be used as part of a PKI. The hardware support is used to generate and store keys and also to speed cryptographic operations. The CA and RAs will almost always require some sort of hardware support to generate and store keys. Potential devices include smart cards, PCMCIA cards, or external devices. An abridged list of manufactures includes Spyrus, BBN, Atalla, Schlumberger, and Rainbow. These devices can cost anywhere from a few dollars up to $5000, depending on model and functionality. They serve not only to increase the performance of CA encryption, but also to provide additional security for the CA private key, because it is difficult to extract the private key from a hardware device.

Normally, one would not employ a smart card on a CA but, if desired, user private keys can be stored on smart cards. Such smart cards may provide additional functionality, such as physical access to company premises. Employing a smart card provides higher security for the user's private key because there is (virtually) no way for the user's private key to be removed from the card, and all computations are performed on the card itself. The downside of smart cards is that each card user must be given both a card and a card reader. Note that additional readers are required anywhere a user wishes to employ the card. There are several card manufacturers, but only some cards work with some PKI selections. The card manufacturers include Spyrus, Litronic, Datakey, and GemPlus. In general, the cards cost approximately $100 per user, including both card and reader.

However, the most important element of a PKI is not a physical element at all, but rather the policies that guide design, implementation, and operation of the PKI. These policies are critical to the success of a PKI, yet are often given short shrift during implementation. The policies are called a "Certificate Practice Statement" (CPS). A CPS includes, among other things, direction about how users are to identify themselves to an RA in order to get their key pair; what the RA should do when a user los-

es his password (and hence cannot unlock his private key); and how keys should be escrowed, if at all. Additionally, the CPS covers areas such as backup policies for the directory servers, CA, and RA machines. There are several good CPS examples that serve as the starting point for an implementation. A critical element of the security of the entire system is the sanctity of the CA itself — the root key material, the software that signs certificate requests, and the OS security itself. Extremely serious attention must be paid to the operational policies — how the system is administered, background checks on the administrators, multiple-person control, etc. — of the CA server.

The technology that underpins a PKI is little different from that of other enterprisewide systems. The same concerns that would apply to, for example, a mission-critical database system should be applied to the PKI components. These concerns include business continuity planning, stress and load modeling, service-level agreements with any outsourced providers or contract support, etc. The CA software often runs either on Windows NT or one of the UNIX variants, depending on the CA vendor. The RA software is often a Windows 9x client. There are different architectures for a PKI. These architectures vary on, among other things, the number and location of CA and RA servers, the location, hierarchy, and replication settings of directory servers, and the "chain of trust" that carries from sub-CA servers (if any) back to the root CA server. Latency, load requirements, and the overall security policy should dictate the particular architecture employed by the PKI.

OTHER PKI BENEFITS: REDUCED SIGN-ON

There are other benefits of a PKI implementation — especially the promise of reduced sign-on for users. Many applications require several authentication steps. For example, a user may employ one username and password pair to log on to his local desktop, others to log on to the servers, and yet more to access the application and data itself. This creates a user interaction nightmare; how many usernames and passwords can a user remember? A common solution to this problem is to employ "trust" relationships between the servers supporting an application. This reduces the number of logins a user must perform, because logging into one trusted host provides access to all others. However, it also creates a significant security vulnerability; if an attacker can access one trusted machine, the attacker has full access to all of them. This point has been exploited many times during PricewaterhouseCoopers attack and penetration exercises. The "attackers" find a development machine, because development machines typically are less secure than production machines, and attack it. After compromising the development machine, the trust relationships allow access to the production machines. Hence, the trust relationships mean that the security of the entire system is depen-

dent not on the most secure systems — the production servers — but rather on the least secure ones.

Even using a trust relationship does not entirely solve the user interaction problem; the user still has at least one operating system username and password pair to remember and another application username and password. PKI systems offer a promising solution to this problem. The major PKI vendors have produced connecting software that replaces most operating system authentication processes with a process that is close to the PKI authentication system described above.

The operating system authentication uses access to the user's private key, which is unlocked with a password. After unlocking the private key, it can be used in the PKI authentication process described above. Once the private key is unlocked, it remains unlocked for a configurable period of time. The user would unlock the private key when first used, which would typically be when logging in to the user's desktop system. Hence, if the servers and applications use the PKI authentication mechanism, the users will not need to reenter a password — they need unlock the private key only once. Each system or application can, if it desires, engage in authentication with the user's machine, but the user need not interact, because the private key is already unlocked. From the user's perspective, this is single sign-on, but without the loss of security provided by other partial solutions (such as trust relationships).

There are other authentications involved in day-to-day business operations. For example, many of us deal with legacy systems. These legacy systems have their own, often proprietary, authentication mechanisms. Third-party products provide connections between a PKI and these legacy applications. A username and password pair is stored in a protected database. When the user attempts to access the legacy application, a "proxy" application requests PKI-based authentication. After successfully authenticating the user — which may not require reentry of the user's PKI password — the server passes the legacy application the appropriate username and password and connects the client to the legacy application. The users need not remember the username and password for the legacy application because they are stored in the database. Because the users need not remember the password, the password can be as complicated as the legacy application will accept, thus making security compromise of the legacy application more difficult while still minimizing user interaction headaches.

Finally, user keys, as mentioned above, can be stored as files or on tokens, often called smart cards. When using a smart card, the user inserts the card into a reader attached to the desktop and authenticates to the card, which unlocks the private key. From then on, the card will answer challenges sent to it and issue them in turn, taking the part of the client machine in the example above. Smart cards can contain more than sim-

ply the user keys, although this is their main function. For example, a person's picture can be printed onto the smart card, thus providing a corporate identification badge. Magnetic stripes can be put on the back of the smart card and encoded with normal magnetic information. Additionally, smart card manufacturers can build proximity transmitters into their smart card. These techniques allow the same card that authenticates the user to the systems to allow the user access to the physical premises of the office. In this model, the PKI provides not only secure access to the entity's systems and applications with single sign-on, but also to physically secured areas of the entity. Such benefits are driving the increase in the use of smart cards for cryptographic security.

PKI IN OPERATION

With the background of how a PKI works and descriptions of its components, one can now walk through an end-to-end example of how a hypothetical organization might operate its PKI.

Imagine a company, DCMEF, Inc., which has a few thousand employees located primarily in southern California. DCMEF, Inc. makes widgets used in the manufacture of automobile air bags. DCMEF uses an ERP system for manufacturing planning and scheduling as well as for its general ledger and payables. It uses a shop-floor data management system to track the manufacturing process, and has a legacy system to maintain human resource-related information. Employees are required to wear badges at all times when in the facility, and these same picture badges unlock the various secured doors at the facility near the elevators and at the entrances to the shop floor via badge readers.

DCMEF implemented its PKI in 1999, using commercial products for CA and directory services. The CA is located in a separately secured data center, with a warm standby machine locked in a disaster recovery site in the Midwest. The warm standby machine does not have keying material. The emergency backup CA key is stored in a safety deposit box that requires the presence of two corporate officers or directors to access. The CA is administered by a specially cleared operations staff member who does not have access to the logging server, which ensures that that operations person cannot ask the CA to do anything (such as create certificates) without a third person seeing the event. The RA clients are scattered through human resources, but are activated with separate keys, not the HR representatives' normal day-to-day keys.

When new employees are hired, they are first put through a two-day orientation course. At this course, the employees fill out their benefits forms, tax information, and also sign the data security policy form. After signing the form, each employee is given individual access to a machine that uses cryptographic hardware support to generate a key pair for that user. The public half of the key pair is submitted to the organization's CA

for certification by the human resources representative, who is serving as the RA, along with the new employee's role in the organization.

The CA checks to be sure that the certificate request is correctly formed and originated with the RA. Then, the CA creates and signs a certificate for the new employee, and returns the signed certificate to the human resources representative. The resulting certificate is stored on a smart card at that time, along with the private key. The private key is locked on the smart card with a PIN selected by the user (and known only to that user). DCMEF's CPS specifies a four-digit PIN, and prohibits use of common patterns like "1234" or "1111." Hence, each user selects four digits; those who select inappropriate PIN values are prompted to select again until their selection meets DCMEF policies.

A few last steps are required before the user is ready to go. First, a copy of each user's private key is encrypted with the public key of DCMEF's escrow agent and stored in the escrow database. Then, the HR representative activates the WWW-based program that stores the new employee's certificate in the directory server, along with the employee's phone number and other information, and adds the employee to the appropriate role entry in the authentication database server. After this step, other employees will be able to look up the new employee in the company electronic phone book, be able to encrypt e-mail to the new employee, and applications will be able to determine the information to which the employee should have access. After these few steps, the user is done generating key material.

The key generating machine is rebooted before the next new employee uses it. During this time, the new employee who is finished generating a key pair is taken over to a digital camera for an identification photograph. This photograph is printed onto the smart card, and the employee's identification number is stored on the magnetic strip on the back of the card to enable physical access to the appropriate parts of the building.

At this point, the new employees return to the orientation course, armed with their smart cards for building access loaded with credentials for authentication to the PKI. This entire process took less than 15 minutes per employee, with most of that spent typing in information.

The next portion of the orientation course is hands-on instruction on using the ERP modules. In a normal ERP implementation, users have to log on to their client workstation, to an ERP presentation server and, finally, to the application itself. In DCMEF, Inc., the users need only insert their smart cards into the readers attached to their workstations (via either the serial port or a USB port, in this case), and they are logged in transparently to their local machine and to every PKI-aware application — including the ERP system. When the employees insert their smart cards, they are prompted for the PIN to unlock their secret key. The re-

mainder of the authentication to the client workstation is done automatically, in roughly the manner described above. When the user starts the ERP front-end application, it expects to be given a valid certificate for authentication purposes, and expects to be able to look that certificate up in an authorization database to select which ERP data this user's role can access. Hence, after the authentication process between ERP application server and user (with the smart card providing the user's credentials) completes, the user has full access to the appropriate ERP data. The major ERP packages are PKI-enabled using vendor toolkits and internal application-level controls. However, it is not always so easy to PKI-enable a legacy application, such as DCMEF's shop-floor data manager. In this case, DCMEF could have chosen to leave the legacy application entirely alone, but that would have meant users would need to remember a different username and password pair to gain access to the shop-floor information, and corporate security would need to manage a second set of user credentials. Instead, DCMEF decided to use a gateway approach to the legacy application. All network access to the shop-floor data manager system was removed, to be replaced by a single gateway in or out. This gateway ran customized proxy software that uses certificates to authenticate users. However, the proxy issues usernames and passwords that match the user's role to the shop-floor data manager. There are fewer roles than users, so it is easier to maintain a database of role-password pairs, and the shop-floor data manager itself does not know that anything has changed. The proxy application must be carefully designed and implemented, because it is now a single point of failure for the entire application, and the gateway machine should be hardened against attack.

The user credentials issued by HR expire in 24 months — this period was selected based on the average length of employment at DCMEF, Inc. Hence, every two years, users must renew their certificates. This is done via an automatic process; users visit an intranet WWW site and ask for renewal. This request is routed to human resources, which verifies that the person is still employed and is still in the same role. If appropriate, the HR representative approves the request, and the CA issues a new certificate — with the same public key — to the employee, and adds the old certificate to DCMEF's revocation list. If an employee leaves the company, HR revokes the user's certificate (and hence their access to applications) by asking the CA to add the certificate to the public revocation list. In DCMEF's architecture, a promoted user needs no new certificate, but HR must change the permissions associated with that certificate in the authorization database.

This example is not futuristic at all — everything mentioned here is easily achievable using commercial tools. The difficult portions of this example are related to DCMEF itself. HR, manufacturing, planning, and accounting use the PKI on a day-to-day basis. Each of these departments

has its own needs and concerns that need to be addressed up-front, before implementation, and then training, user acceptance, and updates must include each department going forward. A successful PKI implementation will involve far more than corporate information security — it will involve all the stakeholders in the resulting product.

IMPLEMENTING A PKI: GETTING THERE FROM HERE

The technical component of building a PKI requires five logical steps:

1. The policies that govern the PKI, known as a Certificate Practice Statement (CPS), must be created.
2. The PKI that embodies the CPS must be initialized.
3. Users and administration staff must be trained.
4. Connections to secured systems that could circumvent the PKI must be ended.
5. Any other system integration work — such as integrating legacy applications with the PKI, using the PKI for operating system authentication, or connecting back-office systems including electronic mail or human resource systems to the PKI — must be done.

The fourth and fifth steps may not be appropriate for all organizations.

The times included here are based on the authors' experience in designing and building PKI systems, but will vary for each situation. Some of the variability comes from the size of clients; it requires more time to build a PKI for more users. Other variability derives from a lack of other standards; it is difficult to build a PKI if the organization supports neither Windows NT nor UNIX, for example. In any case, the numbers provided here offer a glimpse into the effort involved in implementing a PKI as part of an ERP implementation.

The first step is to create a CPS. Creating a CPS involves taking a commonly accepted framework, such as the National Automated Clearing House Association guidelines, PKIX-4, or the framework promulgated by Entrust Technologies, and adapting it to the needs of the particular organization. The adaptations involve modification of roles to fit organizational structures and differences in state and federal regulation. This step involves interviews and extensive study of the structure and the environment within which the organization falls. Additionally, the CPS specifies the vendor for the PKI as well as for any supporting hardware or software, such as smart cards or directories. Hence, building a CPS includes the analysis stage of the PKI selection. Building a CPS normally requires approximately three person-months, assuming that the organization has in place certain components, such as an electronic mail policy and Internet use policy, and results in a document that needs high-level approval, often including legal review.

The CPS drives the creation of the PKI, as described above. Once the CPS is complete, the selected PKI vendor and products must be acquired. This involves hardware acquisition for the CA, any RA stations, the directories, and secure logging servers, as well as any smart cards, readers, and other hardware cryptographic modules. Operating system and supporting software must be installed on all servers, along with current security-related operating system patches. The servers must all be hardened, as the security of the entire system will rely to some extent on their security. Additional traditional information security work, such as the creation of intrusion detection systems, is normally required in this phase. Many of the servers — especially the logging server — will require hardware support for the cryptographic operations they must perform; these cryptographic support modules must be installed on each server. Finally, with the pieces complete, the PKI can be installed.

Installing the PKI requires, first, generating a "root" key and using that root key to generate a CA key. This generation normally requires hardware support. The CA key is used to generate the RA keys that in turn generate all user public keys and associated private keys. The CA private key signs users' public keys, creating the certificates that are stored on the directory server. Additionally, the RA must generate certificates for each server that requires authentication. Each user and server certificate and the associated role — the user's job — must be entered into a directory server to support use of the PKI by, for example, secure electronic mail. The server keys must be installed in the hardware cryptographic support modules, where appropriate. Client-side software must be installed on each client to support use of the client-side certificates. Additionally, each client browser must be configured to accept the organization's CA key and to use the client's certificate. These steps, taken together, constitute the initialization of the PKI. The time required to initialize a PKI is largely driven by the number of certificates required. In a recent project involving 1000 certificates, ten applications, and widespread use of smart cards, the PKI initialization phase required approximately twelve person-months. Approximately two person-months of that time were spent solely on the installation of the smart cards and readers.

Training cannot be overlooked when installing large-scale systems such as a PKI. With the correct architecture, much of the PKI details are below users' awareness, which minimizes training requirements. However, the users have to be shown how to unlock their certificates, a process that replaces their login, and how to use any ancillary PKI services, such as secure e-mail and the directory. This training is usually done in groups of 15 to 30 and lasts approximately one to two hours, including hands-on time for the trainees.

After training is completed, users and system administration staff are ready to use the PKI. At this point, one can begin to employ the PKI it-

self. This involves ensuring that any applications or servers that should employ the PKI cannot be reached without using the PKI. Achieving this goal often requires employing third-party network programs that interrupt normal network processing to require the PKI. Additionally, it may require making configuration changes to routers and operating systems to block back door entry into the applications and servers. Blocking these back-doors requires finding all connections to servers and applications; this is a nontrivial analysis effort that must be included in the project planning.

Finally, an organization may want to use the PKI to secure applications and other business processes. For example, organizations, as described above, may want to employ the PKI to provide single sign-on or legacy system authentication. This involves employing traditional systems integration methodologies — and leveraged software methodologies — to mate the PKI to these other applications using various application programming interfaces. Estimating this effort requires analysis and requirements assessment.

As outlined here, a work plan for creating a PKI would include five steps. The first step is to create a CPS. Then, the PKI is initialized. Third, user and administrator training must be performed. After training, the PKI connections must be enforced by cutting off extraneous connections. Finally, other system integration work, including custom development, is performed.

CONCLUSION

Security is an enabler for electronic business; without adequate security, senior management may not feel confident moving away from more expensive and slower traditional processes to more computer-intensive ones. Security designers must find usable solutions to organizational requirements for authentication, authorization, confidentiality, and integrity. Public key infrastructures offer a promising technology to serve as the foundation for E-business security designs. The technology itself has many components — certificate authorities, registration authorities, directory servers — but, even more importantly, requires careful policy and procedure implementation.

This chapter has described some of the basics of cryptography, both secret and public key cryptography, and has highlighted the technical and procedural requirements for a PKI. The authors have presented the five high-level steps that are required to implement a PKI, and have mentioned some vendors in each of the component areas. Obviously, in a chapter this brief, it is not possible to present an entire workplan for implementing a PKI — especially since the plans vary significantly from situation to situation. However, the authors have tried to give the reader a start toward such a plan by describing the critical factors that must be ad-

dressed, and showing how they all work together to provide an adequate return on investment.

Additional Reading

1. Jonathan S. Held, Password Security, *Data Security Management,* April 2000, No. 83-01-11.
2. William S. Murray, Principles and Applications of Key Management, *Data Security Management,* June 1998, No. 83-10-50.

Douglas C. Merrill, Ph.D., is a senior manager with PricewaterhouseCoopers LLP in Los Angeles, California.

Eran Feigenbaum is a manager with PricewaterhouseCoopers LLP in Los Angeles, California.