Securing e-Medical:
Architecting Protected Health Information Systems

Brian D. Handspicker, Engineering Director, Foliage Software Systems

How do you *get* ready for HIPAA? With the HIPAA mandate for protecting health information nearly upon us, how do you know which security scenarios you must support? What security technology issues do you need to be sensitive about? How do you make tradeoffs between potential security solutions?

This paper discusses the architectural issues associated with developing, deploying, and supporting secure e-medical systems. This is the second in a series of papers from Foliage Software Systems discussing secure e-medical. It assumes that you have already read "Securing e-Medical: A Primer on Protecting Health Information".

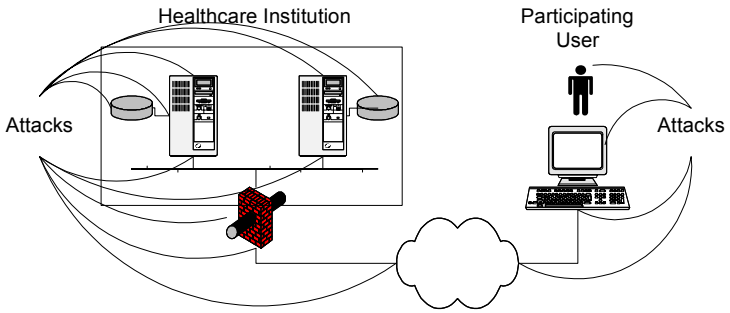
The security and privacy challenge posed by HIPAA

"The Health Insurance Portability and Accountability Act of 1996 (HIPAA) was created in response to the need for the healthcare industry to reliably and confidentially exchange patient healthcare information in support of the portability of healthcare insurance and patients between employees, insurance companies and healthcare providers." To support the confidential maintenance and exchange of electronic healthcare information, the Department of Health and Human Services has developed regulations that support and enforce HIPAA privacy and security:

- Privacy (April 14, 2001, compliance April 14, 2003)
- Security and Electronic Signatures (proposed)

These regulations prescribe care in handling paper records, as well as care in handling electronic records for protected health information.

Protected information that is stored in a distributed e-medical (electronic medical) information system is potentially subject to inappropriate access or modification, also known as an *attack.*



This paper discusses how to secure an e-medical system using widely used digital security technology and mechanisms.

Headquarters: 168 Middlesex Turnpike, Burlington, MA 01803 781.993.5500
Silicon Valley: 51 East Campbell Avenue, Campbell, CA 95008 408.321.8444
www.foliage.com

What does it mean to secure e-medical systems?

To deliver products that enable and support HIPAA, there are a set of formal security and privacy practices and technologies that need to be incorporated into the products you deliver. Specifically, HIPAA requires protecting health information through an array of security, integrity and privacy techniques, including:

- authentication of users [see Authentication],
- authorized access to protected information [see Access Control],
- accountability of changes to protected information [see Accountability],
- integrity of protected information [see Accountability],
- non-repudiation of changes to protected information [see Accountability],
- confidentiality of protected information [see Confidentiality]
- monitoring access and modification of protected information [see Monitoring].

Each of these requirements could be satisfied through a number of different technologies and mechanisms. The architectural challenge is to balance the value of the additional security functionality against the potential risk of the technology, in the context of the value of the information you are attempting to protect. There are no perfect security systems.

Digital security attempts to raise the cost of breaching security high enough that it becomes more cost-effective to simply bribe someone on the inside or execute some other social engineering ploy to get the information desired. At the same time, however, one should only put in place the security that's appropriate for the sensitivity or value of the information and the patience of its users. Security that is excessively tight relative to the perceived value of the information can result in end-users disabling security – either directly or through misuse of the information. For example:

- printing high value information out and leaving it on their desks rather than bothering with logging in to get it from the secure system
- leaving constantly changing random character passwords on a sticky note on the computer monitor.

Such scenarios result in a false sense of security, when in fact security has been compromised.

Evaluating the security requirements, analyzing the cost/benefits and selecting which technology is used to satisfy each of these requirements is done through the definition of a security architecture for your product or environment.

What is a security software architecture?

A software architecture is a set of concepts and design decisions about the structure of a software system and how the resulting parts of the system collaborate to meet the responsibilities of the system. An important aspect of documenting the concepts

and design decisions in an architecture is the identification of potential technology approaches and the tradeoff analysis that leads to selection of specific approaches for the solution architecture.

A *security* software architecture addresses the concepts and design decisions specific to the security of the software system in question, and the selection of security approaches that can best satisfy the security requirements of the product.

How do I architect a secure e-medical system?

The security architecture for an e-medical system that enables and supports HIPAA must address the following computer security services:

- Basic Cryptography
- Authentication
- Access Control
- Accountability
- Integrity
- Non-repudiation
- Confidentiality
- Monitoring

For each of these topics there are potential architectural approaches to satisfying the underlying architectural scenarios. The remainder of this paper discusses those approaches and considers the tradeoffs in selecting between those approaches.

Basic Cryptography

The computer security services that are required to protect electronic information are based on mathematics and computer technology called *cryptography*. Most of the techniques we discuss in this paper are built upon either “secret key encryption” or “public key encryption” cryptography. Cryptography uses mathematical algorithms to convert the text of protected information (“plaintext”) into what appears to be gibberish (“ciphertext”) and back to plaintext. The algorithms use a piece of data called a “key” to control exactly how the plaintext is to be scrambled into ciphertext and then unscrambled again. This process is called encryption/decryption, and is based on the requirements that:

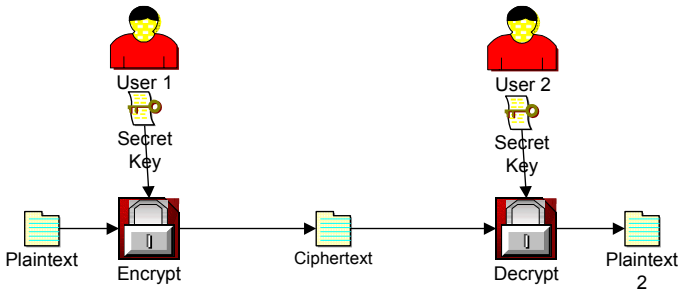
1. it must be extremely difficult to derive the plaintext from the ciphertext without prior knowledge of the key and
2. it must be extremely difficult to derive the key, even if you know both the ciphertext and the plaintext.

Encryption can be implemented using either a single key to scramble and unscramble the text (symmetric or secret key encryption) or a pair of keys, one to scramble and a different key to unscramble the text (public key encryption). Each underlying technology requires a different set of supporting concepts and mechanisms.[i]

Secret Key Encryption

Secret key encryption (a type of *symmetric* encryption) uses a single key to encrypt plaintext and decrypt the resulting ciphertext. This key must be kept secret between

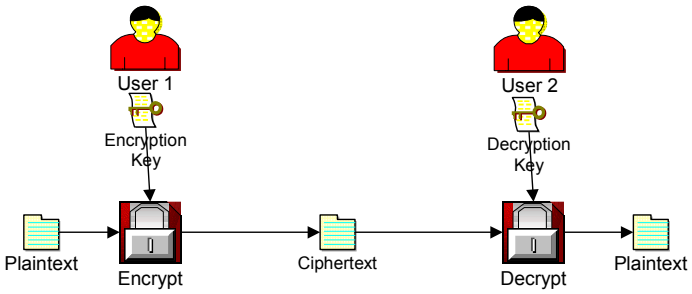
a small number of people and organizations to ensure that the information protected by the encryption remains secure. The fewer people and organizations that must know of the key the better.



Secret key encryption algorithms usually have the benefit of being very efficient and fast and are well suited to encrypting large files and large amounts of data. However, that information can only be decrypted by other people that know the secret key. So, on its own, secret key encryption is limited to small numbers of users of the encrypted information.

Public Key Encryption

Public key encryption (a type of a*symmetric* encryption) uses a pair of keys to encrypt and decrypt information. Either key can be used to encrypt, but only its opposite mate can be used to decrypt the resulting ciphertext. This feature is used in public key encryption systems by allowing one key to be publicly published (obviously the "public key") while its mate is kept secret (the "private key"). This allows *anyone* to use the public key to encrypt information that only the owner of the secret private key can decrypt – making the encrypted information very secure and only available to the intended recipient. On the other hand, the owner of the private key can encrypt information using the private key that *anyone* can decrypt with the assurance that the information could only have been encrypted (and presumably been sent) by the owner of the private key.



The features of public key encryption can be used to implement a number of very valuable security services. For example:

1. If you combine these techniques – a sender encrypting some protected information with an intended recipient's public key, and then encrypting the first resulting ciphertext with the sender's private key – the recipient is assured that the information really came from the sender, no one else, *and* only the recipient can read the information.

2. If you use a private key to encrypt protected information (or as we'll discuss later, a message digest), you can ensure that no one has modified the information since it was last modified or sent.

Tradeoffs - Encryption Schemes:

- Performance versus Number of Participants

Secret key encryption performs faster and uses fewer computing resources than public key encryption. However, the number of participants with whom symmetric encrypted protected information may be shared is limited by the number of participants you can trust to know the secret key. While public key encryption is often slower, an unlimited number of participants may securely exchange protected information.

- Degree of Security versus Breadth of Security Features

Secret key encryption does not require third party service providers (e.g. Certificate Authorities) to participate in the implementation of a secure exchange. As a consequence, if the secret participants are trusted and the secret key well protected, there is a lower potential for social engineering attacks through third parties. However, it is more difficult to implement features such as Digital Signatures or Message Authentication Codes using secret key encryption techniques.

- Simplicity versus Public Key Infrastructure

Secret key encryption techniques require little more than some simple tools, simple policies and a simple trust model. Public key encryption requires an infrastructure of tools, services and third party vendors, as well as a more sophisticated and therefore suspect trust model.

A number of sophisticated approaches combine secret key and public key techniques to get the best of both worlds. (See, for example, "Session Keys" below.) We do not advocate choosing one approach over another. Rather, you should choose the mechanisms you need to meet your specific requirements.

Key Management

Key management is one of the most critical yet challenging aspects of security. Key management provides for the:

- generation of strong keys, both secret and public/private pairs,
- secure storage of keys,
- exchange and distribution of keys to authorized users
- complete destruction of keys on expiration.

Because it is also a tempting target to individuals attempting to breach your security, implementation of key management systems is best left to experienced security consultants and specialist product companies. However, there are a number of key management features that must be considered when selecting a product (e.g. a Public Key Infrastructure) or designing a solution with a security consultant, including key generation, storage, exchange and destruction.

Key Generation – To ensure a secure encryption system, key management systems must generate keys that are random, avoid known "weak" keys and be sufficiently long to increase the difficulty in breaking the keys.

Key Storage – Public keys, private keys, secret keys and , passwords, end up stored in and on computer systems. This security information is obviously very sensitive. In particular, the secure representation and storage of private keys, secret keys and passwords must be carefully designed. Too often, software that uses private and secret keys fails to remove the keys from memory after use – creating an opportunity for clever sleuths to steal the key through inspection of memory. And often, password and key files are poorly protected on disk, and are stored on systems that are physically insecure or are archived on tapes stored in non-secure facilities.

Tip: Software applications that must directly use plaintext copies of private or secret keys should overwrite the memory associated with the storage of the key *immediately* after use. Otherwise a clever attacker can force the application to crash and inspect memory for un-wiped lists of plaintext keys.

Key Exchange – There are a number of cases in which secure, and where possible out-of-band, exchange of private and secret keys is required. When a new public/private key pair is generated for an asymmetric encryption system, the private key must be able to be transmitted to the subscriber for their use. Session-specific secret keys (see below) must be able to be transmitted to the recipient system for use with a symmetric encryption system for the secure transmission of session information. When first getting started with encryption, this can seem like a chicken and egg problem. For example, if you don't have a means of securely exchanging information, how do you securely get the first key you need to start securely exchanging information? Of course, once a public/private key pair are available, they can be used to exchange other keys – whether new public/private pairs, secret keys or one-time session keys.

Key Destruction – To preserve the security of an encryption scheme over the long haul, keys must be changed frequently and previous keys destroyed. Changing keys (and passwords) on a regular basis should be obvious – the longer a potential hacker has to work on a key, the more likely they will be able to break the key or acquire the key through social engineering techniques. The lifetime of a key should be inversely proportional to the value of the information it is protecting (the higher the value the shorter the lifetime). In addition, if the key is programmatically generated, destroying evidence of previous keys provides an added degree of protection to the generation algorithm.

Tip: If password and key files must be backed up, back them up separately from the rest of the data in the system. This way the backup tapes can be stored in the most secure manner and these most sensitive tapes can be destroyed without affecting the backup tapes of other less protected data.

Tradeoffs – Key Management

- Key length: 40-bit versus 64-bit versus 128-bit Keys

With current computer hardware and software, 40-bit keys are relatively easy to break through brute force attacks. 64-bit keys are better, but not perfect. 128-bit keys are preferable, when both the encryption computing time for the information is affordable and the export laws allow.

- Key lifetime: One-time versus Short-term versus Long-term Keys

One-time keys are the least likely to be broken, if their use is truly temporary and the algorithm for generating the keys is not predictable. They are convenient for encrypting information while it is being transferred from one system to another. However, they are ill-suited to use for identifying individuals (e.g. passwords or certificates) or for encrypted information stored long-term on a system. Keys that are used to identify individuals should live long enough to be readily managed and in the case of passwords to avoid disgruntled users. Keys that are used to encrypt data in long-term storage need to change often enough within the value lifetime of the information protected to foil attempts to breach the information, but not so often that they create an administrative overhead for re-encrypting information unnecessarily.

- Key storage: No storage versus Local storage versus Centralized storage

The most secure mechanism for the storage of keys is not to store the keys online at all. This can be accomplished either by requiring keys to be memorized (problematic for keys long enough to be useful) or stored on a hardware token or smart-card (problematic since the owner must be trusted not to lose the card). Storing keys local to either the user's login or the location of the encrypted information could distribute the opportunity for a ne'er-do-well, and thus reduce potential for attack. However, it creates more systems on which the highest physical, hardware and software security must be implemented. Centralizing storage of keys simplifies the management security and eases the problem of securing the keys. However, it creates an attractive nuisance to anyone interested in breaching security.

- Key destruction: Memory and Disk and Tape

Not a tradeoff, but the reiteration of a requirement: when a plain-text key is used, the memory in which it was stored must be zeroed after use. When a plain-text key is destroyed, it must be securely overwritten on disk *and* any backup tapes must be rewritten and destroyed.

Digital Certificates

Many of the security mechanisms we've discussed in this paper are based on the consequences of digitally "signing" protected information using either secret key encryption or public key encryption. When a piece of information is "signed" using public key encryption, the signer uses their private half of the public/private key pair to encrypt the information. Someone testing that signed information (e.g. to validate the source of the information or integrity of the information) uses the public half of the key pair to decrypt the information. The security question that is raised by this operation is how does the person testing the signed information know that the public/private key pair really belong to the asserted signing user and not someone masquerading as that user? In the absence of some means of creating a formal trust between the sender of the information and the tester of the information, it would

be easy to spoof the tester. Digital Certificates are encrypted documents generated by a Certificate Authority that assert the identity of the owner of a public/private key pair. These certificates, based on ISO/CCITT standard X.509, include the name of the authority that issued the certificate, name or identities of the subscriber of the certificate (owner of the key pair), the public key of the subscriber and the operational period of the certificate (valid from date "a" until date "b" – note the certificate operational period is not the same as the associated key lifetimes, but should never be valid longer than the expiration dates of any of the associated keys.) Other information may optionally be included in these certificates. These certificates are then digitally signed by the issuing certificate authority.

Digital Certificate Authorities

Certificate Authorities (CAs) issue digital certificates for use as an assurance of identity of the holder/presenter of the certificate. Organizations may act as their own CA, issuing certificates for their internal users. They may also issue certificates for external users (e.g. partners), assuming their partners trust the organization to act as a CA. Or, an organization may use a third party CA to issue Certificates and act as a "trusted source" for both internal and external users. Of course, you and your partners need to be able to trust the third-party CA generating the certificates. And, by adding another layer of service providers to your security system also brings with it the additional opportunity for "social engineering" to breach the security of your system. The architectural challenge is to balance the value of the additional security functionality against the potential additional risk, in the context of the value of the information you are attempting to protect.

Authentication

Authentication is the process of verifying the identity of a potential user of a system. Most authentication mechanisms are based on some combination of one or more "shared secrets" (e.g. password), security devices (e.g. access card) and/or physical characteristic (e.g. fingerprint). A common combination (sometimes called "two factor authentication") is authentication based on something you *have* (e.g. your ATM card) and something you *know* (e.g. the ATM account PIN). At its simplest, authentication is performed through the verification of an access code or a username/password pair entered by the user at the time they wish access to the system. More sophisticated authentication mechanisms include the use of "smart cards" to store encrypted credentials and even biometric analysis of fingerprints, face-scans or retina-scans. Authentication on its own does not provide control of access to any information or services provided by the system. Authentication only verifies with some level of certainty that you are who you claim to be. The more stringent the authentication requirements, the more sophisticated the authentication mechanism to achieve the level of certainty required.

When an authentication attempt fails (e.g. an incorrect password is given), the attempt to access the system or information must be blocked and the failed attempt logged for future forensics investigation. A retry policy may allow the user to reattempt authentication. However, such policies must limit the number of retries and error messages should not provide any information that might be used to improve attempts to break into a system.

Tip: Never reveal whether it was the username or the password that was incorrect during a failed login attempt. Doing so helps an attacker focus their efforts.

When a successfully authenticated session has been idle for a significant period of time (as specified by local security policy), then access to the session should be blocked, and displayed information cleared from the screen until the user has successfully re-authenticated.

Tip: The policy for idle time-out of an authenticated session should be inversely proportional to the value of the information accessible through the system and proportional to the risk of attack. For example, one should set a very short time-out (e.g. 1 minute) on a pharmaceutical ordering system that is accessible in a public area, one could set a longer time-out (e.g. 15 minutes) on a system that holds private information in a secure area, and perhaps an even longer time-out (e.g. 1 hour) on a personal PC in a locked office.

Password or Access-code Authentication

The simplest mechanism for authentication is the use of a password or access-code that allows a user to “log in” to a system. Passwords and access-codes are often used in combination with other authenticating “information”. Authenticating information could include a username, a magnetic-strip key-card, or even a door-key. When used alone, passwords or access-codes must be unique to the individual you are attempting to authenticate. When used in combination with other authenticating information, the combination of information must be unique to the individual to be able to identify the user of protected information.

Tip: Passwords, access-codes and even usernames are a form of shared secret. While usernames usually make poor shared secrets on their own – being easily guessed, even often reused as email addresses – the combination of an unknown pool of usernames and secret passwords still provides stronger security than a published list of usernames with secret passwords. So, usernames should always be considered secrets not to be widely shared outside an organization.

The strength of a password is directly associated with the length of the password in characters, the complexity of characters used (e.g. mixing upper and lower cases, numbers and letters, punctuation, etc.), the lack of personal clues and avoidance of known words. The ideal password is a random string of upper, lower, numbers, letters and punctuation marks with a length sufficient to discourage direct attacks. However, such passwords are difficult to memorize, and thus often end up written down on a piece of paper and stuck to the monitor or keyboard of the user – thus destroying the very security intended. So organizations usually create a policy that allows users to select their own password, but enforce length, mix of characters and discourage embedded personal information.

One-time Password Authentication

Whenever a password must be exchanged across a network, the password is at risk. Too often, passwords are exchanged in plaintext. Even when exchanged as encrypted text, they are subject to attack or replay. One mechanism for avoiding these problems, particularly between systems that regularly communicate, is the use of a one-time password system. With one-time password systems a user or application is given a sequence of private passwords (through a secure or non-electronic channel) that are each intended to be used only once. The system that

issues the passwords generates the sequence using an algorithm that for each password entered into the algorithm produces the next password. (Of course, the passwords must then be used in reverse order or an attacker that intercepts or cracks one of the passwords could just regenerate the list from that point in the series themselves. Typically, the issuing system keeps the 1st password in the original series to be able to regenerate the entire list, and the last password used.)

Pass-phrase/Challenge Authentication

Though not usually used for primary authentication, the use of pass-phrases (e.g. "Joe sent me") or question/answer challenges (e.g. "what is your mother's maiden name/Smith") is widely used for secondary authentication by customer service personnel to validate a user before revealing or reissuing a forgotten password.

Tip: A single challenge usually makes for weak security (unless you are asking a question so secret or obscure it rises to the level of a password). But in combinations (e.g. "what is your mother's maiden name" and "what are the last four digits of your social security number") they begin to raise the level of confidence. Careful choice of pass-phrases or questions can increase the difficulty of guessing or acquiring the answers by a third party.

Digest Authentication

Digest authentication implemented by HTTP1.1 (RFC 2617) to authenticate secure web transactions, takes the concept of challenge-response authentication one step further. The web server sends a randomly generated challenge to the browser, which computes and returns a checksum of the username, password, challenge value, requested HTTP method, and requested URI (Universal Resource Identifier). Since only the challenge and response is exchanged, rather than the password itself, it provides additional protection for the password. Unfortunately, although this technique avoids exchanging the password at the time of authentication, the password must still have been shared between server and browser at some time before use for authentication.

Digital Certificates

As described in the previous section, digital certificates can store information that can be used to electronically verify an identity:

- *if* you trust the source of the certificate, and
- *if* you trust the initial authentication of the subscriber by the source of the certificate, and
- *if* you trust the storage mechanism of the certificate, and
- *if* you trust the source system presenting the certificate, and (finally)
- *if* you trust that the source system has sufficiently strong authentication, authorization and access control mechanisms and policies to ensure that the individual *using* the certificate is really the *owner* of the certificate.

Even with all of these caveats, digital certificates have become a very popular authentication mechanism in distributed systems because they can convey a lot of authentication information in an encrypted form across the public Internet.[ii] In addition, they provide an essential component to most public key encryption systems and all of the various related security mechanisms.

Hardware Tokens and Smart Card Authentication

One mechanism that allows for more sophisticated authentication without the challenges of complex passwords is the encoding of passwords or digital certificates within either a hardware authentication "token" or a more general "smart card". Hardware that is specific only to authentication is called a token. Hardware that can support authentication, as well as other secure services (e.g. debit card transactions) is called a smart card. The information on the token or card can be as complex as necessary to achieve the desired level of security without requiring the user to remember the information themselves. In particular, authentication tokens are often synchronized with an authentication server to produce "one-time passwords". Of course, now you are dependent on the user to remember the card and protect *its* security.

Tip: One variation on this hardware approach is to include challenge questions/responses within an encoded digital certificate on the token or card that could be used to verify that the holder of the card is likely the owner of the card.

Biometric Authentication

It has become almost routine to see biometric authentication devices in action movies. Of course, at the same time they are being presented by Hollywood as the most sophisticated security mechanisms, they are inevitably being compromised in the film by some clever thief, spy or warrior. Yet, there are highly sensitive circumstances when biometric devices are appropriate. Currently available biometric authentication includes devices to read fingerprints, retinal patterns, voiceprints, and facial characteristics.

Tip: There are also significant concerns that cinematic fiction is closer to reality than some device manufacturers would like. Recent research by Tsutomu Matsumoto revealed that $10 worth of kitchen supplies could be used to create gelatin fake fingers with molded fingerprints that fool some fingerprint authentication devices 80% of the time.[iii]

Verification Services

Verification services provide a subscription service to businesses that wish to double check the identity of a user. These services are often run by credit reporting agencies which have large amounts of information about individuals. They compare information submitted by the subscriber about an individual to information in their credit databases (e.g. home address, social security numbers, credit card numbers, etc.) to generate a probability of accurate identity. This is similar to a challenge question/answer approach to authentication, but often is done without the end-user's knowledge.

Tradeoffs - Authentication:

- Robustness of key versus limits of human memory

The longer and more random an encryption key, the stronger the resulting encryption. However, the longer and more random the key, the harder it is for the key to be memorized. The result is often either humans memorizing short, simple keys that are easy to crack, or longer more random keys stored on a computer system that are easy to find. To meet high security requirements, this tradeoff leads to the use of either hardware tokens/smartcards or biometric authentication.

- Degree of electronic security versus risk of physical loss

Hardware tokens and smartcards can be used to gain higher security from longer keys with shorter key-lives. However, these cards need to be protected by the individual, and care must be taken not to let the cards fall into the hands of others – even temporarily or accidentally.

- Cost to acquire versus cost to maintain

Tokens, smartcards, biometric scanners, and other hardware based authentication solutions add capital costs to implementing a security system. However, to achieve strong, ongoing security through software based authentication solutions, a significant administrative effort is required to generate, securely store, retire, destroy and regenerate keys that are held by humans and computers.

- Use of third parties to verify or authenticate versus close control over crypto-information

Use of third-party certificate authorities and verification services can offload some of the responsibility for initial authentication and add a perception of added trust to a security system. However, third-parties also become additional points of attack on the associated identities, certificates and keys.

Access Control

Access Control is the process of determining what privileges (also known as grant or denial of "rights") that an authenticated user (human or programmatic) performing in a authorized role (e.g. "administrator", "owner") has for the access to a specific resource (e.g. application, service, file or piece of information) and what operations the user may perform on that resource (e.g. read, modify, delete, execute) based on a set of policies and/or rules.

Access control is sometimes also called "authorization", particularly when referring to the process of determining whether a user is authorized to have access to a system or application. Access to a specific piece of information can only be granted to a properly authenticated, authorized user with appropriate levels of access (e.g. a physician may have complete access to their own patient's information, yet only have access to de-identified or statistical information about another physician's patients). Obviously, care should be taken when granting access to not only preclude access to the unauthorized users but also grant access to information to a *covering* healthcare provider during appropriate periods of time and granting access to emergency personnel. Access control can be implemented with at least four levels of sophistication:

- User-based access control
- Role-based access control
- Hierarchical-based access control
- Context-based or rules-based access control

User-based Access Control

Implementing authorization policies can be as simple as maintaining and referring to a list that matches individual user identity and a specific right for a resource. These are usually implemented by a specific application on a specific system (e.g. a file system access control list, aka ACL). The challenge with user-based access control is that it requires considerable system administration effort to maintain the access control lists for all users on all systems for each piece of protected information.

Role-based Access Control

Since users often play specific roles (e.g. administrator, physician, covering physician, etc.), the administrative headache of managing a protected system can be reduced by matching a user's role to a specific right for a resource. However, this too becomes problematic when managing access rights across many different systems. In addition, individual user rights must still be granted for information owned by individuals and not accessible by other members of the user's groups.

Hierarchical Access Control

By organizing roles and groups of users hierarchically, the effort to manage access rights can be further simplified – if no specific permissions are granted for a specific user or role for a resource, the user/role inherits the default rights of its parent group. This approach works well if there are groups of users with nearly identical access rights across all individual member users. However, in healthcare environments, the rights that you have to protected information can depend on the context.

Context or Rules-based Access Control

Under normal circumstances, a user in a healthcare facility has limited access to patient information. Only those with a direct need-to-know are allowed access to detailed protected information. However, there are a number of situations in which healthcare professionals need to have access to information based on the current context:

- a covering physician must have access to patient information in a unit
- a physician receiving a referral must access some subset of a patient's protected information
- an emergency room nurse must access historical information while performing triage in a busy ER

More sophisticated approaches to implementing context-based access control policy involve the use of policy and rules services for all of the resources within a system or a site. These access control services may support both static policy (e.g. what applications can be used, files that can be accessed, directories that can be viewed, hours of access, storage quota, etc.) and dynamic rules based on real-time context (e.g. whether currently covering for another healthcare provider).

Tradeoffs – Authorization and Access Control:

- User and Role-based versus Context and Rules-based Access Control

Many applications already provide mechanisms for implementing and testing authorization. For example, file system access control mechanisms exist on all major operating systems. However, the cost and inconvenience of managing authorization policy across many applications on many machines quickly undermines diligence in maintaining this critical security information. On the other hand, context and rules-based access control products can be expensive to buy, challenging to integrate with legacy systems and frustrating to configure in their own right. You should consider the long-term consequences of choosing simple local access control versus choosing sophisticated centralized access control.

- Granularity of information protected versus cost of maintaining and testing authorization

At a minimum access to applications and files must be protected. Some environments require protecting information at the granularity of individual database records. This adds computing overhead, as well as management overhead. In rare cases, individual database fields must be independently protected. However, this dramatically increases the computation cost and implementation complexity. (And, if possible, may be better addressed through restructuring the schema for the database.)

Accountability

Users of protected health information (whether authorized or not) are accountable for all access to, modifications to, and distribution of that information. HIPAA requires that unauthorized access to protected health information be reported. Audit logs are tools for logging the authorized and unauthorized use of the applications and/or services of a system, as well as access and/or changes to protected health information. Regular inspection of audit logs is critical to protecting the security of the system, the integrity of the protected information, and the legal standing of the organization. (see Monitoring)

Protected health information must be represented, stored and distributed in such a way that any attempt to alter the information (whether authorized or not) can be identified and tracked. Typically a system-independent mechanism bound tightly to the information (e.g. checksum, CRC, or digital signature) provides corroboration of the integrity of the information.

Non-repudiation takes integrity of information one step further to provide non-refutable evidence of creation, deletion, modification or distribution of information. This ensures that even an authorized user cannot access, change, or share the information and then deny the access.

Checksums and CRC (Cyclical Redundancy Checks)

Protected information can be protected against accidental change during transmission using checksums or cyclical redundancy checks (CRCs). A checksum is generated before the information is transmitted and is attached to the transmission. On receipt of the protected information a new checksum is generated

and compared to the checksum that was transmitted. If there is a difference between the sender's and the recipient's information checksum, there has been an error in transmission of the information. Checksum algorithms are designed to catch major errors in transmission, but often catch small errors as well.

However, checksums should only be considered for protection against accidental change. Checksums provide only a weak integrity check. First, many different pieces of information can yield the same checksum value. Second, the typical checksum is usually a small value (often on the order of 8 bits). A malicious outsider with knowledge of the checksum or CRC algorithm could intercept the information, alter the contents, and/or re-compute the checksum transparently. The main value of checksums is to detect data corruption.

Message Digests

Another technique for recognizing a change in a piece of protected information is to generate and encrypt a message digest of the information and then attach the resulting MD key to the original information. A message digest is created by running the protected information through a *hash function.* Hash functions take variable length data, and based on the data generate a fixed length (128 or 160 bit) key that is unique to the original data.

If the original information remains unchanged, a regenerated message digest should match the original message digest. If someone tampers with the information, the regenerated message digest will not match the original digest. The original digest can be protected by being digitally signed by the last authorized modifier of the information.

The algorithm used by a message digest makes it extremely unlikely that two different pieces of information will end up with the same MD key. On the other hand, even very slight differences in the format of the information (say a tab substitution for spaces) will generate two different MD's.

Message Authentication/Integrity Codes

Message authentication codes (MACs), also known as message integrity codes (MICs) are a variation on message digests. However, with MACs, the last authorized modifier's secret key is hashed along with the protected information to create a digest that could only have been created by the owner of the key. MACs need not be digitally signed to assure authenticity, but they are also only verifiable by the owner of the secret key.

Digital Signatures

A digital signature of protected information is created by generating a message digest of the information and then encrypting the digest with the private key of a public/private encryption key pair. A recipient of the message can verify that the information originated from the purported sender by decrypting the message digest and comparing the original digest with a newly generated digest. This also verifies that the information has not been accidentally or maliciously changed since its last authorized modification.

Digital Timestamps

To assure non-repudiation of a change to protected information, it is necessary to include a non-forgeable timestamp with a digital signature. This ensures that in the event a private key is ever compromised (through breach, accident or even intentional publication), the original owner of the key cannot repudiate modifications that occurred before the compromise.

Tradeoffs - Accountability:

- Do we need to know if it has changed in transit accidentally?

Use CRCs or checksums. They are simple, cost-effective and widely implemented.

- Do we need to know if it has changed in storage?

Use Message Digests. They are simple, cost-effective and more accurate for long-term storage.

- Do I know no one else has changed it?

Use Message Authentication Codes. Since they use secret keys, they are very secure and provide proof to a small number of people that the information has not changed.

- Do we need other people to know who changed it?

Use Digital Signatures. Since they use private/public keys that are secure, while still testable by anyone that needs to know.

- Do we need to prove who changed it when?

HIPAA requires use of Electronic Signatures to enforce non-repudiation on changes to electronic records. HIPAA Electronic Signatures combine Digital Signatures with Digital Timestamps to provide proof that the information has not changed since a specific time at which it was encrypted by a specific person.

Confidentiality

Data in a secure system, or data being exchanged across secure distributed systems must not be viewable by unauthorized users or systems. Privacy often must extend to privacy of requests to the secure system. Unauthorized users could glean important information about data on the system simply based on "over-hearing" a request regarding the data (e.g. "copy fredsmith-hiv-test.doc.\archive").

Confidential Storage

Confidential information should never be stored in plaintext on a non-secure system. If the physical or electronic security of a system is suspect, confidential information should be stored encrypted using keys that *are not* stored on the non-secure system. Depending on the nature of the information, it may be stored in an encrypted file, stored as a set of message digest records or stored as encrypted fields within a database.

Tip: A common technique for storing login passwords is to create a file with a set of message digest records, one record per username/password pair. Since the password is not stored, just the digest for it, the original password cannot be derived. Yet, when a user logins in with a valid password, the message digest of the entered password will match the message digest in the password file.

Confidential Communications

Protecting the privacy of data that is being exchanged across a network requires encryption of the data. This encryption can happen at a number of different layers in the communications process:

1. Network layer encryption
2. Session layer encryption
3. Application layer encryption

Network layer encryption is usually implemented between secure network routers based on the IETF IP sec standard. These routers encrypt all (IP payload) traffic sent between themselves and other mutually authenticated routers. If the sending and receiving local area networks (LANs) and associated systems are secure, and the path between them is secure (e.g. configured as a virtual private network using network-level encryption), session-level and application-level encryption is not necessary.

The challenge with network-layer encryption is that information is only encrypted between the edges of the LANs of each organization. The information will be in plaintext within each LAN and on the end-systems of each organization. When you are unsure if the network between end-systems is secure, session-layer encryption can be used to secure the information being exchanged from end-system to end-system independent of whether the network-layer is encrypted. Session-layer encryption is often implemented using the secure socket layer (SSL) industry standard. SSL is used for secure transactions by web, email, and file transfer applications. SSL supports two different key-lengths – 40 bit and 128 bit. The longer the key, the more difficult it will be to break the encryption. However, support for keys larger than 64-bit (as of 5/24/02) is subject to U.S. government export control. Many companies have secured export licenses for their 128-bit SSL implementations. Care should be used when selecting technology if there is a need to allow access to information from outside the United States.

SSL is being replaced by a developing new IETF standard “Transport Layer Security” (TLS – RFC 2246). TLS is based on SSL, with some enhancements to improve security. Unfortunately, as a result of the enhancements, TLS and SSL are not compatible or interoperable with each other. The industry is still gaining experience implementing TLS, so wide-scale use may yet take some time.

Certain types of highly-confidential information (e.g. encryption keys) should rarely exist in plain-text. For these types of information, the associated application may need to control and exchange the information in continuously encrypted form. If the information is encrypted by the application-layer, it is irrelevant if the session-layer or network-layer encrypts the information.

Session Key-Based Encryption

The broad use of public key encryption techniques carries with it two challenges:

1. public key asymmetric encryption/decryption is slow compared to secret key symmetric encryption/decryption
2. the more information exchanged using a public/private key pair, the more opportunity for attack on that pair

A common technique for exchanging large amounts of information between systems is to use public key encryption to securely exchange a temporary secret key to be used for symmetric encryption during this session. The symmetric secret key encryption/decryption will execute faster for the large amounts of information to be exchanged. In addition, once the session is over the key is destroyed limiting the opportunity for someone to break the key.

Replay Protection

Even with strong measures to protect privacy and integrity of information, it is possible for a third party to intercept and record protected information in transit and retransmit it at a later time. If the duplicate transmissions are not detected, logged and discarded, significant health or financial problems can be created. For example, multiple orders for pharmaceutical treatment could result in overdoses or theft; multiple work orders for diagnostic tests could result in over-billing. Replay protection can be implemented using globally unique transaction identifiers.

Tip: Resist the urge to use shorter locally unique identifiers. Over the lifetime of the information system, the breadth of systems with which you need to interoperate is certain to grow with the prospect of new conflicts between transaction identifiers.

Data Compression and Encryption

Data compression is often used to reduce the size of information being exchanged between systems. Data compression algorithms exploit statistical properties in the original information to reduce the encoded size of the information. By their very nature, encryption algorithms destroy the statistical properties that compression algorithms use to reduce the size of the information. So compressing encrypted information will not significantly reduce its size. However, compressing a clear-text copy of the information and then encrypting the results, not only produces a smaller encrypted message, but may also improve the security by reducing redundancy in the pre-encrypted information.

Tradeoffs - Confidentiality:

- Network versus Transport versus Application layer encryption

If the sending and receiving LAN environments are known to be physically and electronically secure and an encrypted VPN exists between the systems, using the built-in network-layer encryption is transparent and centralizes management of the associated encryption keys. Centralized management of a smaller number of keys increases the likelihood that the keys will be well maintained and well protected. If, however, either LAN is suspect but the end-systems are trusted, use SSL/TLS transport-layer encryption to protect the information

exchanged. The application will need to know how to pass information through the appropriate API or protocol, and the user will need to have the appropriate Digital Certificates installed with the application. This increases the wire line security of the encrypted information, but also increases the risk of breach through potential attacks on the certificates and through the use of third party certificate authorities.

Monitoring

Each of the security mechanisms used to protect sensitive systems and information has the potential of being breached by a clever attacker. To maintain the security and integrity of the protected information, vigilance is required, not only in selection of technology and implementation of policies, but also in monitoring the systems to identify attempted breaches, suspicious access trends, and track compliance with specified policies and procedures. Monitoring includes:

- Audit Logs
- Trend Analysis
- Alarms
- Event Reporting
- Cyberforensics

Audit Logs

Each attempt to create, access, modify, transfer, or delete protected information must be tracked and recorded. This is called an audit trail or audit log. Audit logs must be reviewed on a regular basis, either manually or through an automated rules-based analysis system.

Trend Analysis

Often questionable attempts to access or modify information occurs across a set of related protected information. One failed attempt on one piece of information could be overlooked as a slip of the mouse. However, multiple attempts to gain access to the same set of information should trigger an alarm.

Alarms

Alarms are notifications of an unusual event or attempt to breach the security of the protected system. The alarms and alarm system need to be protected themselves to ensure that the functionality and information in the alarm system are not compromised to hide attempts to breach the system.

Event Reporting

When a protected health system or protected health information is compromised, federal and state laws require that the event be formally reported.

Cyberforensics

Finally, if an attempt to breach the security of the system is recognized, cyberforensic tools should be used to identify the attacker and close the security weakness being exploited.

What do I do Monday morning?

Putting this information all together, medical device manufacturers and healthcare software vendors need to go through the following steps to ensure they architect appropriate solutions for their security needs:

1. Identify security scenarios you need to support
2. Identify the risks inherent in the environment to be protected
3. Identify any sensitivities in the environment to be protected
4. Perform tradeoff analysis on potential architectural approaches
5. Integrate selected security approaches into products/environment
6. Monitor and audit environment to identify weaknesses
7. Repeat as necessary

About Foliage Software Systems

Foliage Software Systems delivers custom software, complex systems integration services, and technology strategy consulting. Since its founding in 1991, Foliage has completed more than 150 projects for clients in heathcare, semiconductors, avionics, financial services, wireless services, and e-business. More than three-quarters of Foliage's software engineers have a decade or more of experience and the average is eighteen years. The company's 95% retention rate facilitates teamwork and continuity from project to project. Foliage has been consistently profitable, is self-funded, and has annual revenues of more than $25 million. Foliage operates development centers at our headquarters in Burlington, Massachusetts and at our Silicon Valley office in Campbell, California. The company has been named to the Deloitte & Touche *Fast 50* for the last two years and Software Magazine's *Software 500*. Learn more about Foliage's track record by viewing the success stories at: http://www.foliage.com/medical.

Headquarters: 168 Middlesex Turnpike, Burlington, MA 01803 781.993.5500
Silicon Valley: 51 East Campbell Avenue, Campbell, CA 95008 408.321.8444
www.foliage.com

---

[i] Warwick Ford, Computer Communications Security: Principles, Standard Protocols and Techniques 71-75 (1994);
Bruce Schneier, Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2.1-2.8, 21-46 (2d ed. 1996)

[ii] "The Emperor's New Clothes: The Shocking Truth About Digital Signatures and Electronic Commerce", Jane K. Winn, 2001
"Ten Risks of PKI: What You're not Being Told about Public Key Infrastructure", Carl Ellicon and Bruce Schneier, Computer Security Journal, Volume XVI, Number 1, 2000
"Conventional Public Key Infrastructure: An Artifact Ill-Fitted to the Needs of the Information Society", Roger Clarke, European Conference in Information Systems, June 2001

[iii] T. Matsumoto, H. Matsumoto, K. Yamada, S. Hoshino, "Impact of Artificial Gummy Fingers on Fingerprint Systems," Proceedings of SPIE Vol. #4677, Optical Security and Counterfeit Deterrence Techniques IV, 2002.