MODULE 10 AUTHENTICATION, CRYPTOGRAPHY AND DIGITAL SIGNATURES

Authentication –

- Authentication is used by a server when the server needs to know exactly who is accessing their information or site.
- Authentication is used by a client when the client needs to know that the server is system it claims to be.
- In authentication, the user or computer has to prove its identity to the server or client.
- Usually, authentication by a server entails the use of a user name and password. Other ways to authenticate can be through cards, retina scans, voice recognition, and fingerprints.
- Authentication by a client usually involves the server giving a certificate to the client in which a trusted third party such as Verisign or Thawte states that the server belongs to the entity (such as a bank) that the client expects it to.
- Authentication does not determine what tasks the individual can do or what files the individual can see. Authentication merely identifies and verifies who the person or system is.

Digital Certificates –

Digital Certificates are part of a technology called Public Key Infrastructure or PKI. Digital certificates have been described as virtual ID cards. This is a useful analogy. There are many ways that digital certificates and ID cards really are the same.

In creating digital certificates a unique cryptographic key pair is generated. One of these keys is referred to as a public key and the other as a private key. Then the certification authority—generally on your campus—creates a digital certificate by combining information about you and the issuing organization with the public key and digitally signing the whole thing. This is very much like an organization's ID office filling out an ID card for you and then signing it to make it official.

In PKI terms, the public key for an individual is put into a digital document, along with information about that individual, and then the digital document is signed by the organization's certification authority. This signed document can be transmitted to anyone and used to identify the subject of the certificate. However, the private key of the original key pair must be securely managed and never given to anyone else. As the private key is a very large prime number, it is not something an individual memorizes; rather, the private key must be stored on some device, such as a laptop computer, PDA, or USB key ring.

What is Digital signature –?

Above we stated that the digital certificate was digitally signed. The holder of a digital certificate can also use it to digitally sign other digital documents, for example, purchase orders, grant applications, financial reports or student transcripts. A digital signature is not an image of your pen and ink signature—it is an attachment to a document that contains an encrypted version of the document created using the signer's private key. Once a document is signed, no part of that document can be changed without invalidating the signature. Thus if someone obtained a copy of your digital certificate and changed the name in it to be their own name, any application receiving that modified certificate would see immediately that the signature on it was not valid. In this sense, a digital credential is much better than a traditional ID card to prove that the holder is really the person to whom it was issued. In fact, digital signatures in general are much more useful than pen and ink signatures since anyone checking the signature also can find out something about the signer in order to know whether the signature is meaningful.

Public Key Infrastructures and Certificate Authorities

Digital certificates are one part of a set of components that make up a public key infrastructure (PKI). A PKI includes organizations called certification authorities (CAs) that issue, manage, and revoke digital certificates; organizations called relying parties who use the certificates as indicators of authentication, and clients who request, manage, and use certificates. A CA might create a separate registration authority (RA) to handle the task of identifying individuals who apply for certificates. Examples of certification authorities include VeriSign, a well-known commercial provider, and the CREN Certificate Authority that is available for higher education institutions. In addition to the organizational roles, there must be an associated database or directory, generally using a directory access protocol called LDAP, which will store information about certificate holders and their certificates. There also must be a way to make available information about revoked certificates. An application that makes use of PKI digital credentials may consult the revocation database before relying on the validity of a certificate. It may wish to consult the Subject directory as well in order to retrieve further information about the certificate Subject.

Types of Certificates

There are different types of certificates, each with different functions and this can be confusing. It helps to differentiate between at least four types of certificates. You can see samples of some of these different types of certificates in your browser.

- Root or authority certificates. These are certificates that create the base (or root) of a certification authority hierarchy, such as Thawte or CREN. These certificates are not signed by another CA—they are self-signed by the CA that created them. When a certificate is self-signed, it means that the name in the Issuer field is the same as the name in the Subject Field.

- Institutional authority certificates. These certificates are also called campus certificates. These certificates are signed by a third party verifying the authenticity of a campus certification authority. Campuses then use their "authority" to issue client certificates for faculty, staff, and students.
- Client certificates. These are also known as end-entity certificates, identity certificates, or personal certificates. The Issuer is typically the campus CA.
- Web server certificates. These certificates are used to secure communications to and from Web servers, for example when you buy something on the Web. They are called server-side certificates. The Subject name in a server certificate is the DNS name of the server.

Certificate based authentication –

Let's see how the certificate based authentication works, When presented with a certificate, an authentication server will do the following (at a minimum):

1. Has the Digital Certificate been issued/signed by a Trusted CA?
2. Is the Certificate Expired – checks both the start and end dates
3. Has the Certificate been revoked? (Could be OCSP or CRL check)
4. Has the client provided proof of possession?

Let's examine the above 4 items one at a time:

Has the Digital Certificate Been Signed by a Trusted CA?

The signing of the certificate really has two parts. The first part is the certificate must have been signed correctly (following the correct format, etc). If it is not, it will be discarded immediately. Next, the signing CA's public key must be in a Trusted Certificates store, and that certificate must be trusted for purposes of authentication.

Has the Certificate Expired?

Just like a driver's license or a passport, a certificate will have 2 dates listed in it: a date issued, and date it is valid to (when does it expire).

An authentication server does the same sort of check. Is the certificate valid for the date and time that the authentication request comes in. This is one reason why Network Time Protocol (NTP) is so important when working with certificates. Many of us have seen problems where time was out of sync. For example: a certificate was presented on January 10th 2014 at 11:11am, but its "valid-from" value started on January 10th at 11:30am. This was because of a time sync issue where the Certificate Authority thought it was 20 minutes later than the authentication server, and the brand-new certificate was not valid yet

Has the Cert Been Revoked?

You are driving down the road, and you are pulled over by a policeman. The policeman asks for your driver's license and proof of insurance. You hand the officer a driver's license, which is immediately checked for evidence of authenticity, i.e.: does it look like a valid driver's license or a forgery. Ok, it's not fake, check. Next, expiration: it is not expired. Check. Now the policeman asks you to wait there, while he goes back to his squad car.

While in the squad car, the officer will perform some authorization checks (are you a registered owner of the car you were driving, etc.). Those are not important for this conversation, though. What is important is that the policeman must make sure your VALID driver's license was not revoked by the DMV. A quick look-up on the computer into the DMV records shows that your driver's license was revoked for too many DWI's. The cold steel of the handcuffs and the rough shove into the back seat of the Squad car as you are hauled off to jail make you re-evaluate your life choices.

Certificate Authentication has the same capability (not the handcuffs, I'm talking about the look-up into the DMV to revocation status). Every certificate authority should also have a service to publish a list of certificates that have been revoked. There are 2 main ways to do this today:

- Certificate Revocation List (CRL). This is basically a signed list that the CA publishes on a website that can be read by authentication servers. The file is periodically downloaded and stored locally on the authentication server, and when a certificate is being authenticated the server examines the CRL to see if the client's cert was revoked already. CRL could be compared to the policeman having a list of suspended drivers in his squad car.
- Online Certificate Status Protocol (OCSP). This is the preferred method for revocation checks in most environments today, because it provides near real-time updates. OCSP allows the authentication server to send a real-time request (like a http web request) to the service running on the CA or another device and checking the status of the certificate right then & there. OCSP could be compared to the policeman using the computer in the squad car & doing a look-up into the DMV's database.

If the certificate has been revoked, then access is denied. Enjoy the lights and sirens on the way to jail.

Cryptography –

Cryptography is where security engineering meets mathematics. It provides us with the tools that underlie most modern security protocols. It is probably the key enabling technology for protecting distributed systems, yet it is surprisingly hard to do right.

Cryptography is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication.

Cryptographic goals –

1. Confidentiality is a service used to keep the content of information from all but those authorized to have it. Secrecy is a term synonymous with confidentiality and privacy. There are numerous approaches to providing confidentiality, ranging from physical protection to mathematical algorithms which render data unintelligible.
2. Data integrity is a service which addresses the unauthorized alteration of data. To assure data integrity, one must have the ability to detect data manipulation by unauthorized parties. Data manipulation includes such things as insertion, deletion, and substitution.
3. Authentication is a service related to identification. This function applies to both entities and information itself. Two parties entering into a communication should identify each other. Information delivered over a channel should be authenticated as to origin, date of origin, data content, time sent, etc. For these reasons this aspect of cryptography is usually subdivided into two major classes: entity authentication and data origin authentication. Data origin authentication implicitly provides data integrity.
4. Non-repudiation is a service which prevents an entity from denying previous commitments or actions. When disputes arise due to an entity denying that certain actions were taken, a means to resolve the situation is necessary. For example, one entity may authorize the purchase of property by another entity and later deny such authorization was granted. A procedure involving a trusted third party is needed to resolve the dispute.

Symmetric-key encryption

Let’s assume that Alice wants to talk to Bob. She wants to keep the message secret. Bob is the only one who should be able to read the message. The message is confidential, so Alice uses a key to encrypt the message. The original message is called a plaintext while the encrypted message is called a ciphertext. The ciphertext is sent to Bob, who knows the key and uses the same symmetric cipher (e.g., AES or 3DES). Thus Bob is able to decrypt the message.

Alice and Bob share the key, which is called symmetric. They are the only ones who know the key and no one else is able to read the encrypted message. This way, confidentiality is achieved.

Key Length vs. Security

The key space doubles when one bit is added to the key. Longer keys are better, but don’t necessarily increase security. Because people tend to use patterns for passwords, the attacker can build a dictionary of commonly used passwords and launch a dictionary attack. This way the attacker can save time, because he doesn’t have to brute force the whole key space.

Symmetric vs. Session Key

The symmetric key can be changed every time Alice communicates with Bob. Then it is called a session key (randomly generated and valid only for one session). If an attacker grabs the session

key, he can decrypt only the messages from one session. If Alice and Bob always used the same key, the attacker would be able to decrypt all messages encrypted with this key.

Scalability and Secure Key Distribution

There are a few problems with symmetric ciphers. This system is not scalable. If there are 1,000 people who want to communicate with each other, everyone needs 999 different keys to establish separate and confidential communication channels. Secure key distribution is another problem. The security of the system is broken if a man-in-the-middle can grab the key while it is being transmitted from Alice to Bob.

Public-key cryptography

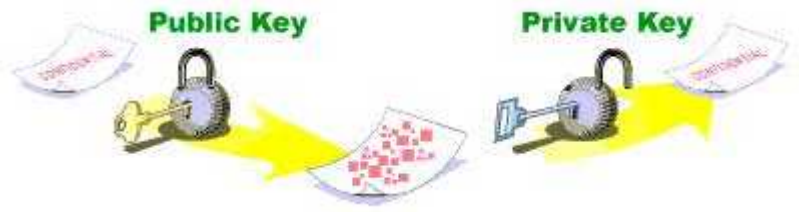
The Public and Private key pair comprise of two uniquely related cryptographic keys (basically long random numbers). Below is an example of a Public Key:

3048 0241 00C9 18FA CF8D EB2D EFD5 FD37 89B9 E069 EA97 FC20 5E35 F577 EE31 C4FB C6E4 4811 7D86 BC8F BAFA 362F 922B F01B 2F40 C744 2654 C0DD 2881 D673 CA2B 4003 C266 E2CD CB02 0301 0001

The Public Key is what its name suggests - Public. It is made available to everyone via a publicly accessible repository or directory. On the other hand, the Private Key must remain confidential to its respective owner.

Because the key pair is mathematically related, whatever is encrypted with a Public Key may only be decrypted by its corresponding Private Key and vice versa.

For example, if Bob wants to send sensitive data to Alice, and wants to be sure that only Alice may be able to read it, he will encrypt the data with Alice's Public Key. Only Alice has access to her corresponding Private Key and as a result is the only person with the capability of decrypting the encrypted data back into its original form.



As only Alice has access to her Private Key, it is possible that only Alice can decrypt the encrypted data. Even if someone else gains access to the encrypted data, it will remain confidential as they

should not have access to Alice's Private Key. Public Key Cryptography can therefore achieve Confidentiality.

Hashes

Hash functions take some data of an arbitrary length (and possibly a key or password) and generate a fixed-length hash based on this input. Hash functions used in cryptography have the property that it is easy to calculate the hash, but difficult or impossible to re-generate the original input if only the hash value is known. In addition, hash functions useful for cryptography have the property that it is difficult to craft an initial input such that the hash will match a specific desired value.

MD5 and SHA-1 are common hashing algorithms used today. These algorithms are considered weak (see below) and are likely to be replaced after a process similar to the AES selection. New applications should consider using SHA-256 instead of these weaker algorithms.

Key Exchange Algorithms

Lastly, we have key exchange algorithms (such as Diffie-Hellman for SSL). These allow use to safely exchange encryption keys with an unknown party.

Algorithm Selection

As modern cryptography relies on being computationally expensive to break, specific standards can be set for key sizes that will provide assurance that with today's technology and understanding, it will take too long to decrypt a message by attempting all possible keys.

Therefore, we need to ensure that both the algorithm and the key size are taken into account when selecting an algorithm.

How to determine if you are vulnerable

Proprietary encryption algorithms are not to be trusted as they typically rely on 'security through obscurity' and not sound mathematics. These algorithms should be avoided if possible.

Specific algorithms to avoid:

- MD5 has recently been found less secure than previously thought. While still safe for most applications such as hashes for binaries made available publicly, secure applications should now be migrating away from this algorithm.
- SHA-0 has been conclusively broken. It should no longer be used for any sensitive applications.
- SHA-1 has been reduced in strength and we encourage a migration to SHA-256, which implements a larger key size.

- DES was once the standard crypto algorithm for encryption; a normal desktop machine can now break it. AES is the current preferred symmetric algorithm.

Cryptography is a constantly changing field. As new discoveries in cryptanalysis are made, older algorithms will be found unsafe. In addition, as computing power increases the feasibility of brute force attacks will render other cryptosystems or the use of certain key lengths unsafe. Standard bodies such as NIST should be monitored for future recommendations.

Specific applications, such as banking transaction systems, may have specific requirements for algorithms and key sizes.

How to protect yourself

Assuming you have chosen an open, standard algorithm, the following recommendations should be considered when reviewing algorithms:

Symmetric:

- Key sizes of 128 bits (standard for SSL) are sufficient for most applications
- Consider 168 or 256 bits for secure systems such as large financial transactions
- Symmetric-key encryption protocols should include message authentication
- Always Encrypt then MAC

Asymmetric:

The difficulty of cracking a 2048 bit key compared to a 1024 bit key is far, far, far, more than the twice you might expect. Don’t use excessive key sizes unless you know you need them. Bruce Schneier in 2002 (see the references section) recommended the following key lengths for circa 2005 threats:

- Key sizes of 1280 bits are sufficient for most personal applications
- 1536 bits should be acceptable today for most secure applications
- 2048 bits should be considered for highly protected applications.

Hashes:

- Hash sizes of 128 bits (standard for SSL) are sufficient for most applications
- Consider 168 or 256 bits for secure systems, as many hash functions are currently being revised (see above).

NIST and other standards bodies will provide up to date guidance on suggested key sizes.

Key Storage

Crypto relies on keys to assure a user's identity, provide confidentiality and integrity as well as non-repudiation. It is vital that the keys are adequately protected. Should a key be compromised, it can no longer be trusted.

Any system that has been compromised in any way should have all its cryptographic keys replaced.

How to determine if you are vulnerable

Unless you are using hardware cryptographic devices, your keys will most likely be stored as binary files on the system providing the encryption.

Can you export the private key or certificate from the store?

- Are any private keys or certificate import files (usually in PKCS#12 format) on the file system? Can they be imported without a password?
- Keys are often stored in code. This is a bad idea, as it means you will not be able to easily replace keys should they become compromised.

How to protect yourself

- Cryptographic keys should be protected as much as is possible with file system permissions. They should be read only and only the application or user directly accessing them should have these rights.
- Private keys should be marked as not exportable when generating the certificate signing request.
- Once imported into the key store (CryptoAPI, Certificates snap-in, Java Key Store, etc.), the private certificate import file obtained from the certificate provider should be safely destroyed from front-end systems. This file should be safely stored in a safe until required (such as installing or replacing a new front end server).
- Host based intrusion systems should be deployed to monitor access of keys. At the very least, changes in keys should be monitored.
- Applications should log any changes to keys.
- Pass phrases used to protect keys should be stored in physically secure places; in some environments, it may be necessary to split the pass phrase or password into two components such that two people will be required to authorize access to the key. These physical, manual processes should be tightly monitored and controlled.

- Storage of keys within source code or binaries should be avoided. This not only has consequences if developers have access to source code, but key management will be almost impossible.
- In a typical web environment, web servers themselves will need permission to access the key. This has obvious implications that other web processes or malicious code may also have access to the key. In these cases, it is vital to minimize the functionality of the system and application requiring access to the keys.
- For interactive applications, a sufficient safeguard is to use a pass phrase or password to encrypt the key when stored on disk. This requires the user to supply a password on startup, but means the key can safely be stored in cases where other users may have greater file system privileges.

Insecure transmission of secrets
In security, we assess the level of trust we have in information. When applied to transmission of sensitive data, we need to ensure that encryption occurs before we transmit the data onto any untrusted network.

In practical terms, this means we should aim to encrypt as close to the source of the data as possible

How to determine if you are vulnerable

This can be extremely difficult without expert help. We can try to at least eliminate the most common problems:

- The encryption algorithm or protocol needs to be adequate to the task. The above discussion on weak algorithms and weak keys should be a good starting point.
- We must ensure that through all paths of the transmission we apply this level of encryption.
- Extreme care needs to be taken at the point of encryption and decryption. If your encryption library needs to use temporary files, are these adequately protected?
- Are keys stored securely? Is an unsecured file left behind after it has been encrypted?

How to protect yourself

We have the possibility to encrypt or otherwise protect data at different levels. Choosing the right place for this to occur can involve looking at both security as well as resource requirements.

Application: at this level, the actual application performs the encryption or other crypto function. This is the most desirable, but can place additional strain on resources and create unmanageable complexity. Encryption would be performed typically through an API such as the OpenSSL toolkit (www.openssl.com) or operating system provided crypto functions.

An example would be an S/MIME encrypted email, which is transmitted as encoded text within a standard email. No changes to intermediate email hosts are necessary to transmit the message because we do not require a change to the protocol itself.

Protocol: at this layer, the protocol provides the encryption service. Most commonly, this is seen in HTTPS, using SSL encryption to protect sensitive web traffic. The application no longer needs to implement secure connectivity. However, this does not mean the application has a free ride. SSL requires careful attention when used for mutual (client-side) authentication, as there are two different session keys, one for each direction. Each should be verified before transmitting sensitive data.

Attackers and penetration testers love SSL to hide malicious requests (such as injection attacks for example). Content scanners are most likely unable to decode the SSL connection, letting it pass to the vulnerable web server.

Network: below the protocol layer, we can use technologies such as Virtual Private Networks (VPN) to protect data. This has many incarnations, the most popular being IPsec (Internet Protocol v6 Security), typically implemented as a protected ‘tunnel’ between two gateway routers. Neither the application nor the protocol needs to be crypto aware – all traffic is encrypted regardless.

Possible issues at this level are computational and bandwidth overheads on network devices.