Lecture Notes for Chapter 3
System Security

Digital Signatures: A digital signature is a scheme that is used to simulate the security properties provided by a hand-written signature. It is something which only the signer should be able to produce but everybody should be able to verify.

Digital signatures are easily supported using public key cryptography. Signer encrypts the message using his private key and anybody can verify it using the signer's public key. However, since the message itself can be arbitrarily large and public key cryptography is slow, instead of encrypting the message with sender's own public key, it usually encrypts/signs only the digest of the original message, which is a much smaller summarization of the original message. The message digest serves to protect the integrity of the message. Non-repudiability is enforced by the fact that only sender has the corresponding private key for encrypting the digest. Authenticity can be enforced by providing a mechanism that binds the identities of the senders with the keys (See digital certificates). Symmetric key cryptography provides authentication and integrity but does not provide the third property of non-repudiability required by a Digital Signature scheme. Both parties have the same secret key, and hence each is exposed to fraudulent falsification of a message by the other. (It is possible to support digital signatures using symmetric key cryptography by relying on a trusted intermediary, called the arbitrator.)

Message Digests: Message digests are usually used in authentication. They are one-way hash functions such that given the digested bits, there is no way to reproduce the original message (since they map arbitrary long bit string into a constant-length bit string space). A good message digest should have the property that any small change in the original message will change the entire hash value. This unpredictable output behavior makes it difficult to modify the original message in such a way that the hash value remains unchanged.

MD5 is one of the most widely used cryptographic hash functions with 128-bit hash value. People have been able to use the Brute Force attack and Birthday attack to find collisions in MD5. For N possible values, we can expect a collision after seeing approximately sqrt(N) of them. So, for an n-bit key, after trying 2^(n/2) keys, a collision is expected.

Cryptographers recommend the use of SHA-1 which has a 180-bit hash value. In the past several years, researchers have begun to find some weaknesses in SHA-1. (These weaknesses reduce the strength of the hash by some number of bits, but do not make it "easy" in any sense of the word.) Newer algorithms that fix this weakness are being designed.

Digital Certificates: With public keys, it is possible to create a new key pair and distribute the public key, claiming that it is the public key for someone else. You could sign messages using such a private key, which can be verified using your public key by the receiver of the message. However, there would be no assurance that the message indeed originate from you, since it is possible that an attacker pretended to be you, and sent his public key to the receiver, pretending it to be yours. Digital Certificates provide a way to overcome this "man-in-the-middle" attack.

A certificate is an attestation mechanism: it provides a way for Alice to prove to Bob that the public key she is presenting to him is indeed hers. To do this, she relies on a trusted third party called "Certification Authority" (CA). In particular, it is assumed that Bob (and any one who wishes to verify Alice's public key) will trust the CA, and has the CA's public key. Now, a digital certificate is nothing but Alice's public key that is signed by the CA (i.e., Alice's public key encoded using CA's private key). When Bob receives Alice's certificate from her, he can decode it using the CA's public key (which he must already possess) to obtain Alice's public key.

In real-world deployment, there are usually multiple root CAs --- typically a small number. The public keys of these root CAs are typically hard-coded into any software that needs to verify certificates, e.g., a web browser. (As of 9/2011, Firefox seems to trust about 60 root CAs.) Those CAs may also have the

rights to sign the public keys of other CAs or individuals to extend trust to them. This forms a chain of trust starting from the root CAs.

Designing Secure Systems: Security involves people and people are fallible. As a result, 100% security is not easily achieved. Any security mechanism tries to reduce the risk by making the job of attacker difficult. In trying to identify vulnerabilities and making arrangements for them, we often make systems less usable. There is a trade-off that exists between security and usability.

Layering and Bypassing: Layering is an important principle in software design, and provides an important way to break down and manage system complexity. They allow software designers to reason about the functionality of one layer purely in terms of the layer that is immediately beneath it, while ignoring layers that are further down. But in the context of security (as well as performance), it is no longer enough to reason about one layer at a time – security mechanisms can be bypassed from any layer in most systems, so one needs to look at the entire system design and implementation.

General Principles and Concepts: It is difficult to evaluate whether a system is secure or not. It is only possible to detect vulnerabilities. Assurance in a secure system is the confidence that it is free of security holes. The simpler the system, the greater is the assurance.

Integrity and Authentication: User authentication provides a means to “bind” entities in the physical world (human users, typically) with entities in cyber space (usually, software). It can be based on secrets (e.g., passwords), or physical possessions (e.g., keys to locks, entry cards, biometrics).

Passwords: The user feeds in a password which is compared against the password stored in the password file, e.g., on a UNIX system, it is the /etc/passwd or /etc/shadow file. The theft of the password file could lead to theft of all the passwords stored in it. Even if the security of this file could somehow be ensured while it is on the computer, one needs to worry about its security when the entire system is backed up --- how can you ensure that all those backups on tapes are really secure? Hence, instead of storing the passwords in plain text, their hash values are stored. In early days, a modified version of DES that acted like a hash was used. These days, MD5 is more common. When the user provides a password to authenticate, its hash is computed and compared against the stored hash value. If they match, access is granted. If the total password space is 10^6, a naïve brute-force attack will require 10^6 attempts. But if there are 1000 users in the password, then an attacker can guess a password, hash it, and then compare it with the hash of all 1000 users. As a result, the expected number of guesses to find a password is in the range of 1000 rather than a million. To defeat this easier attack, and bring back the number of guesses to 1 million in the above example, a “salt” value is used. In simple terms, a random value is generated at the time of user account creation. This value is inserted into the password file. The salt is appended to user-provided password before it is hashed. This way, the hash function used for each user becomes different, as the salt values for distinct users are distinct. As a result, an attacker has to perform a distinct hash operation per user, thus he has to compute 10^6 hashes in the example just mentioned.

In an offline attack, an attacker has access to the encrypted (or hashed) password. The attacker is able to perform hashes on her own computer, and compare the result with the hashes to verify if she has the right password. .Hence, she is able to mount dictionary attacks that work fast --- if it takes a microsecond to perform a hash, the attacker can cycle through a million guesses in a second. To slow down attackers, we can force them into performing an online attack by preventing access to encrypted password file. For instance, these days, UNIX systems keep the encrypted password in /etc/shadow that is readable only by the super user. The salt value is also stored there. Rest of the information, such as the user name, userid, groups, etc. continue to be in /etc/passwd for backwards compatibility

Online attacks require an attacker to type the password on the target system, and make it perform the authentication. To slow down attacks, typical strategies include: (a) introducing artificial delays, e.g., ensure that each authentication attempt takes a few seconds, (b) adaptive system response/ limiting the number of attempts, e.g., locking out the terminal (or the user account) after 3 failed attempts. Alternatively, an exponential backoff algorithm could be used, e.g,, double the delay period with each failed attempt. In general, it is believed that online attack is harder since the system can react to repeated attacks. Hence, a weaker (in a relative sense) password can be used if offline attacks are not

possible. However, one should be very skeptical before ruling out offline attacks. Repeated break-ins into web sites over the past several years have shown that supposedly secure files containing passwords (in some cases, cleartext passwords) were stolen.

In a distributed environment, there are two approaches to authentication:

[1] Host Based Authentication: In Host-based authentication, the server delegates responsibility for user authentication to the client host. It is based on a trust relationship between machines. For instance, the NFS (network file system) service relies on this form of host-based authentication. The advantage of this approach is that it does not require passwords to be transmitted over the network. Its drawback is that it is very weak in today's environment where end-users often possess administrative privileges on their workstations.

[2] Direct user authentication: In this scenario, the end-user authenticates herself directly to the server. This requires passwords to be sent to the server from the client machine. In the past, services such as telnet, ftp and rsh transmitted password in plaintext. This is clearly insecure. Today, most such services have been replaced by more secure ones that don't transmit passwords in plaintext. An example is SSH, which is widely used today. SSH provided two basic ways for authentication. The first scheme simply encrypts user's password on the network. It relies on host's public/private key pairs to bootstrap this encryption. The second scheme relies on a public/private key pair for the user. The public key is stored on the server, while the private key is stored on the client. (Private keys are always encrypted using passphrases, which are basically passwords. The password is used as a secret key.) This scheme used a challenge response protocol. The server sends a challenge to Alice, say, a random number encrypted using Alice's public key. Only Alice can decrypt it with her private key and send it back to the server.

Password Problems: Users use a lot of passwords and they have to remember the passwords in their minds. Hence, people tend to use easy-to-remember passwords, which are often strings with some simple rules like their pet's name, social security number, son's name, etc. Those are vulnerable to dictionary attacks.

Dictionary attacks exploit the fact that users tend to select passwords from a much smaller set based on some rules rather than a set of all alphanumeric strings of a certain length. By enumerating this smaller set, it becomes feasible to break passwords (in a short time). For instance, the number of possible alphanumeric strings of length eight is 36^8, which is close to 3 trillion. But the number of English words is much less than a million. As a result, if a user picks an English word as a password, it can be broken relatively easily (in a manner of minutes in an offline attack.) Even if words are not chosen from English, it can be the case that the very techniques designed to promote user recall of passwords can also reduce the space of possible passwords. If those rules/techniques are captured by a password enumeration technique, cracking the passwords become easier.

Although some systems force users to choose a more "secure" password, like having a combination of alphanumeric and special characters, users may use some simple rules to generate a new password that can satisfy the requirements. For example, when it needs a number, a user may add "1" at the beginning or at the end of their own password. If this is done, then these password rules have the opposite of the intended effect: they simply make some of the characters highly predictable, effectively leading to a shorter password.

It is very common to have a password manager which helps users to manage their passwords. For example, a web browser can remember the login information for websites. Users can then pick a strong password and let the browser remember the password. All of the passwords remembered by the browser can themselves be protected using a master password. Instead of being forced to remember different passwords for different websites, users need only remember one master password. Unfortunately, some websites tend to prevent the browser from to remembering passwords, and force the user to type in the password every time they visit the site. The motivation of these sites is to prevent password information being remembered in shared-use computers, where they may be misused by a subsequent user. Unfortunately, this approach too has the opposite of the intended effect: they lead the user to pick a low-complexity, easy-to-remember password. Another factor driving these password policies is the question of liability. Financial sites want to put in certain features that would satisfy the "due diligence" requirement from a legal perspective, even if the practical effect of the

policies is to provide decreased security.

Graphical password: The motivation of using graphical password is that good text-based passwords are difficult to remember. Hence, researchers are studying the use of graphical passwords as humans have a very powerful visual perception. For example, given a picture, users can pick some points/pixels on the picture as their password. (This scheme is referred to as “passpoints.”) While such an approach can provide a very large space of possible passwords, in practice, the key space may be smaller. For instance, users may be attracted to certain “hot spots” in the picture such as brightly colored spots, facial features, etc. As a result, the actual keyspace can be much smaller.

Another technique for visual passwords is passfaces. It involves showing the user a set of images of faces and then allowing the user to select a set of faces as password. It might seem that security can be enhanced if the user is shown different set of faces which will contain the passfaces. But the set of passfaces can be determined by calculating the intersection of all the sets and the user will be at risk. So it is better to show the user same set of images everytime and let the user choose the passfaces.

However, visual passwords have not been deployed significantly due to some of their shortcomings, as well as the dearth of experience in their use. One of the shortcomings is shoulder-surfing. An onlooker can look at the screen and know which images the user is clicking and can imitate the same clicks to get access to the user’s account. The entropy or the randomness achieved may be small since users may end up choosing certain faces (or other types of pictures, or points on an image) much more often than other. Once again, some of the issues that arise with text passwords arise: people choose images that are easy to remember, and this may end up leading to low entropy passwords.

Visual prompt: Sometimes a picture is used to prove the authenticity of the site. It is believed that this can prevent phishing attacks as users will be able to verify the legitimate site by seeing their personalized image, which is not accessible to the attacker. (Phishers are obviously able to recreate the look and feel of an official site in terms of its publicly accessible web pages, but do not know the specific picture that is shown only to a specific user.) Also in order to thwart phishing attacks, apart from the sitekey as mentioned above, some sites also have security skins which are some images used as the background or watermark.

Unfortunately, while they can offer some protection, these techniques are not particularly effective. The user may not be attentive, and may not notice the absence of their image on the login screen. Or, a phisher may include a message that the user's image is not available at that time due to a system outage. As a result, the user may end up entering their passwords on the phishing web site.

Note that phishing web sites can be thwarted using SSL. While an attacker can copy the content and the feel of a legitimate web site, he does not have access to the digital certificate of that site, and hence will not be able to successfully establish a trusted SSL connection. He can try to overcome this by presenting a self-signed certificate that asserts he is the legitimate web site, hoping that a careless user will accept such a certificate. To reduce this likelihood, browsers have made it difficult for users to accept them. For instance, in Firefox, one has
to go through multiple steps to accept a self-signed certificate, and moreover, is warned repeatedly about the dangers.

Two factor authentication: Instead of relying purely on knowing a password, a system may use additional factors to authenticate the user. For example, by requiring the user to have certain device or card; or to verify certain biometrics. One of the commonly used multi-factor authentication uses a magnetic card and a PIN (e.g., ATM); or a hand-held device in conjunction with a password or PIN (e.g., RSA’s SecureId card). One could also use a smartcard, which can support stronger suthentication protocols that make use of challenge/response paradigm.

Authentication mechanisms

Authenticating people can be based on 3 factors:

1. Something the user KNOWS : e.g. – a password or PIN
2. Something the user HAS: e.g. – An ATM card, smartcard or hardware token.

3. Something the user IS : e.g. – biometrics like fingerprint, facial recognition, iris recognition or voice recognition.

Most systems use a single factor - passwords for authentication. Good passwords must have 2 properties:

They must not be easily guessable.
They must be easy to remember.

Since these two properties are conflicting, it is desirable to use shorter easy to remember passwords, coupled with more than one factor for authentication.
Multifactor authentication uses a combination of methods for authentication. A typical scenario could be usage of two factor authentication for authenticating a user to a ATM machine based on ATM card and a secret user PIN. One possible protocol that could be used by the ATM machine is as follows. The ATM card uses magnetic strip to store the information of the users account encrypted with a secret key known the ATM machines. The PIN number, along with other account information, could be stored either on the card in this encrypted form. With this approach, the ATM machine can validate it locally, without having to contact the bank. Note that online attacks are made difficult in this scenario since the ATM machine would typically capture the card after a few unsuccessful attermpt (usually, 3 attempts). Offline attacks, where that attacker tries to read the contents of the magnetic strip offline and try to crack the PIN number stored on it, is rendered difficult by the fact that the attacker does not know the key used to encrypt the information stored on the magnetic strip. However, a malicious ATM can of course steal the PIN since the user has to enter it in plaintext. Recently, attacks have proliferated where an attacker attaches a new facade to existing ATMs that can be used for this purpose --- they can capture both the contents of the magnetic strip and the PIN, and these can be replayed to steal money.

There can also protocols where the PIN verification is done by the bank. Still, if the user has to enter her PIN in plaintext on the ATM machine, attacks using a malicious ATM are possible. To avoid such attacks, the bank can rely on a challenge-response protocol. For instance, the bank can send a challenge to the user, and expect the user to send a decrypted version of this challenge, decrypted using the PIN. Such a protocol, naturally, will require a smart ATM card that can respond to this challenge, rather than expecting the user to do this.

ATM cards are magnetic cards that can be reproduced easily. Smart Cards, on the other hand, are more secure: they can response to random challenges sent from the financial institution directly. Replay attacks with messages recorded by a compromised terminal can be prevented.

One time password: The basic goal of a One Time Password scheme (OTP's) is to make it difficult for attackers to gain access to computer systems based on compromise of user passwords. OTP's are passwords that are usable just one time. Since the password is usable just once, an attack requires the attacker to intercept the password, and prevent it from reaching the intended target. Additional attempts by the user to login (if she suspects a transient network error rather than an attack) must also be intercepted and prevented, until the attacker has the ability to use the password once. At this point the attacker can login, but just once. Moreover, the legitimate user, when she is unable to login using the OTP, will start suspecting something fishy, and may take steps to track down the attacker, or at least lock him out.

From an attacker perspective, there are multiple difficulties due to OTP. They can no longer rely on passive snooping, which is much easier to carry out than an active interception attack that is required here. Second, the legitimate user gets to know right away that there is a problem, whereas without an OTP, the attacker may be able to login many times using the snooped password without leaving too many obvious clues for the legitimate user that their password has been compromised.

One of the scheme for generating one time password is to use a master password P, and keep on applying hash function H to produce a sequence of hash values $H^1(P)$, $H^2(P)=H(H(P))$, …, $H^n(P)$ and use the $H^i(P)$ for the i-th password. In this case, user only needs to remember the last password used,

and compute the next password by applying H. However, if the attacker is able to capture some of the passwords, they may be able to figure out what H is, and will be able to produce the next unused valid passwords easily. This can be prevented by using those hash values in the reverse direction, i.e. the user can compute all the n hashed values, and use $H^n(P)$ as the first password, $H^{n-1}(P)$ as the second password. Since it is computationally infeasible to compute $H^{n-1}(P)$ from $H^n(P)$ using $H^{n-1}(P) = H^{-1}(H^n(P))$, it is secure. Moreover, verification is simpler at the system side since the system only needs to check by simply applying H to the user's password and compare it with the last seen password. Noted that when using one time password, the system should always ask for a new password, regardless of the previous requested password can be received or not. This is to ensure that network interception attacks would subsequently be detected. Thus the system has to maintain just index of the password last asked and the last successfully accepted password and its index. In order to recover from system or network failure, it has to hash the received password for times equal to the difference between the index of last successful password and the index of the password asked.

Any cryptographic hash function like MD5, SHA1 etc can be used in OTP generation.

Another approach for generating OTPs is by encrypting current time with a secret key. This approach works well especially in conjunction with smartcards, e.g., a smart card may store a strong key internally, perform the encryption, and display the result that can be typed in by a user.

Password Theft: Password theft intended to steal the password without interrupting the information flow. Passwords can be stolen by spyware, or keylogger snooping on keystrokes, and the passwords will then be sent to the attackers.

Spoofing: Spoofing attack corresponds to a malicious program masquerading as another program. For example, the system may show a popup requesting user password. It may happen the popup is from a malware but not the system.

Click-jacking: Click-jacking is a technique for capturing keystrokes. In this attack, 2 webpages are overlapping on each another. The attacker's webpage is sitting on the top, and is invisible and transparent to the users. Users can only see a legitimate website, and they will think that they are interacting with the legitimate website. However, they are actually interacting with the attacker's webpage. This can allow the malicious web page to harvest the password that the user believes is being typed on a legitimate web site.

Trusted path: Trusted path is a mechanism that provides confidence that the user is communicating with what the user intended to communicate with, ensuring that attackers can't intercept or modify whatever information is being communicated. For example, the login screen requiring users' password should result in the password being sent to the operating system, but not other software. On Microsoft Windows, Ctrl-Alt-Del is a key stroke which no other applications, except the OS's login screen will be able to handle. Other applications are locked out at this point, and the subsequent key strokes can only be received by the OS. (However, this assumes that the OS, and the underlying hardwares and BIOS are trustworthy. An example of malicious hardware could be a hard disk using DMA mechanism to write some malicious code into the memory directly.) This trusted path should exist both ways: the user should be able to send information to the login program, while being assured that no other entity can snoop on it; and should be able to determine with certainty when he/she is being prompted for password that she is communicating with the trusted login program.

Biometrics: The third way is to rely on who you are, i.e., biometrics. Biometrics can be human physiology, anatomy, skill or trait. e.g., fingerprint, iris, voice, or keystroke dynamics. They are convenient and protect against poor choice of passwords. Also these can be very difficult to steal in a controlled/ supervised environment. However, the accuracy in recognition is an issue. Fraud rate (false negative) and insult rate (false positive) are two system design parameters that can be configured:

[1] False Negatives: When a system allows access to someone who should not be getting the access.
[2] False Positive: When a system denies access to a legitimate user.

There exists a trade-off between the two, i.e., given a system, you can decrease the FN by accepting a higher FP or vice-versa. So, a definition of accuracy needs to take into account both these numbers. A

commonly used metric is “equal error rate” when both FN and FP are equal. Systems are usually evaluated based on what is the minimum achievable equal error rate.

Various individual characteristics can be used for biometric authentication. The handwritten signatures on signature tablets cannot be trusted as the pattern matching is not very accurate. Most humans are not able to distinguish between a legitimate and forged signature. The face recognition technique which pattern matches the face of the user with the one in the system is computationally expensive and has high error rate. Further, privacy concerns prevent the use of such system extensively. The most extensively used biometric is the fingerprint. However, due to the possibility of fingerprints being stolen and deposited at the required location, this technique is not very secure. The most accurate system with high level of security is the Iris recognition. The iris is unique for each person, not influenced by genetics and doesn’t wear out over lifetime of the individual. Gabor filters can transform the iris image into a 256-byte code which matches 90% with any other code deduced from the same iris. However, this technique requires low camera to eye distance and can potentially be copied in unattended situations. Another biometric that can be used is voice recognition. With this metric, recording attack can be thwarted using challenge-response authentication so that the user has to actually speak the set of words that his prompted with. Another biometrics used is keystroke dynamics in which the time elapsed between every key stroke can be noted and compared with the user keystroke pattern. Once again, a challenge-response paradigm could be used to deter record/replay attacks, where the user is prompted to type a different set of words for each login.