A Gentle Introduction to Cryptography

Part 2

Asymmetric (Public Key) Cryptography

- Public key cryptography is an attempt to circumvent the key distribution problem completely.
- As it turns out, asymmetric algorithms tend to be very inefficient.
- Their main use is in solving the key exchange problem for symmetric cryptography.

Public Key Crypto (cont.)

- In asymmetric cryptography, each user has two keys: a public key and a private key.
- The public key is made public. For example, it may be published on a Web site.
- The private key must be kept secret. It is never shared with anyone.
- The security of the private key in public key crypto is as important as key security in symmetric crypto.

Public Key Crypto (cont.)

- There is no key distribution problem in public key cryptography.
- Some people have compared public key cryptography to a mailbox.   Many people can put mail into the mailbox (in effect, using the public key), but only a postal worker with the appropriate key (corresponding to the private key) can retrieve the mail from the mailbox.

Figure A-2

- Alice wants to send a message to Bob.
- Alice uses Bob's public key to encrypt the message.
- The encrypted message is sent over the insecure medium.
- Bob uses his private key to decrypt the encrypted message.
- No one but Bob knows the private key.

Public Key Crypto (cont.)

- Public key encryption and decryption algorithms tend to be incredibly slow relative to symmetric key algorithms.
- Public key algorithms tend to be about 100 times slower than DES.
- In general, encrypting large messages using public key cryptography is not considered practical.

Public Key Crypto (cont.)

- The most important use for public key cryptography is for solving the symmetric cryptography key exchange problem.
- Viega and McGraw say that using public key cryptography is a more secure choice than using key derivation algorithms.
- SSL uses this strategy: public key crypto for sharing keys and symmetric algorithms for encrypting the message.

Rivest, Shamir, and Adelman (RSA)

- RSA is the most famous public key algorithm.
- RSA starts with picking two HUMONGOUS prime numbers, p and q.   Each of these prime numbers contains hundreds to thousands of bits.
- The two prime numbers remain secret (they are the private key).
- Their product, $n = p * q$, is the public key.

RSA (cont.)

- The product (the public key) is used to encrypt the message.
- Only someone who knows the prime factors can decrypt the message in a reasonable amount of time.
- The security of RSA is based on the difficulty of factoring n into the prime factors p and q.
- At this point in history, this is seen as a difficult problem.

RSA (cont.)

- RSA is still considered secure after twenty years of use.
- The big security problem is that some implementations of RSA have been flawed and had security problems of their own.
- The software developer should use a well-tested and highly-regarded implementation of RSA.
- Never code your own!!!

RSA (cont.)

- There are huge numbers of large prime numbers.
- There are approximately 10^{151} primes of length 512 bits or less.
- One interpretation is that there are enough primes of up to 512 bits to assign every atom in the universe 10^{74} prime numbers without ever repeating one of those primes.

The Future of RSA

- The future of RSA is hard to predict.
- It depends upon what happens in prime number factoring theory.
- Not too many years ago, experts believed that no one would ever have the resources necessary to factor a 128 bit number.
- Now, an organization with adequate resources, can factor a 512 bit number in just a few months.

The Future of RSA (cont.)

- Viega and McGraw recommend that you use no less than a 2,048 bit key for data requiring long-term security (ten or more years).
- It may be that 1,024 bit numbers may be nearing the end of their usefulness even for short-term security.
- The longer the key, the longer it takes to encrypt messages using public key cryptography.

Public Key Crypto Vulnerabilities

- Public Key encryption algorithms are more susceptible to chosen plaintext attacks than symmetric algorithms.
- However, since public key is generally used to encrypt small messages (like keys), plaintext attacks are not a practical problem.
- More significant is the “man-in-the-middle” type of attack.

“Man-in-the-Middle” Attacks

- Figure A-3 depicts a “man-in-the-middle” attack.
- First, let’s consider this kind of attack in terms of Alice trying to send a message to Bob.
- In this kind of attack, Ted sends Alice his own public key, misrepresenting it as Bob’s public key.
- Ted is pretending that he is Bob when he is communicating with Alice.

Man-in-the-Middle (cont.)

- Ted sends Bob Ted’s own public key, misrepresenting it as Alice’s public key.
- Ted is pretending to be Alice when he communicates with Bob.
- Ted is intercepting all traffic between Bob and Alice.

Man-in-the-Middle (cont.)

- When Alice sends Bob a message, she encrypts it using Ted’s public key (she thinks it is Bob’s).
- When Ted receives Alice’s message, he can decrypt it.
- Ted can then send Bob a modified or entirely different message, encrypting it was Bob’s public key.
- Bob decrypts the message, thinking it came from Alice.

“Man-in-the-Middle” (cont.)

- Figure A-3 depicts the situation in terms of a client and a server.
- The client (Alice) asks for the server’s public key so she can send secure information to the server (Bob).
- But, the client is not communicating with the server. She is communicating with the attacker, who sends her his public key.

Man-in-the-Middle (cont.)

- Meanwhile, the attacker establishes his secure connection with the server.
- This gives the attacker access to any information that the client sends to the server and any information that the server sends back to the client.
- This kind of problem motivates the need for a public key infrastructure (PKI).

PKI

- The basic idea behind a Public Key Infrastructure (PKI) is that a trusted third party certifies valid keys.
- Back to Bob and Alice. In this case, Alice would receive Bob’s public key through a trusted third party, a certification authority (CA).
- The CA would say, in effect: “Alice, trust us, Bob is a dependable fellow and this is Bob’s public key.”

PKI (cont.)

- Obviously, this does not solve the matter of trust (the security problem).
- How can Alice be sure that she can trust the so-called trusted authority?
- One of the largest CA's at this point in time is Verisign.
- Verisign performs background checks on applicants before issuing them a public key for a fee.

PKI (cont.)

- Verisign’s track record is not perfect.
- Several people registered with Verisign under the name “Bill Gates”.
- In March 2001 Microsoft announced that two false keys with MS’s name on them had been issued by a CA.

PKI (cont.)

The problem of trusted identity, taken from Viega and McGraw ….

PKI (cont.)

Advice for developers from Viega and McGraw ...

Cryptographic Hashing Functions for Data Integrity

- Cryptographic hashing functions are used to ensure the integrity of data.
- Cryptographic hashing functions are sometimes called cryptographic checksums or integrity checksums.
- Hashing functions are also used for digital signatures, which we shall discuss later.

Integrity Checksums

- Since stuff happens, it is important to have some means of detecting unauthorized changes to files.
- An integrity checksum is a value that is computed from the data that is being protected.
- The integrity checksum is stored separately from the protected data.

Integrity checksums (cont.)

- The recipient of the data recomputes the checksum from the data that is received and compares that checksum to the value that was recorded separately by the provider of the data.
- If the original checksum and the recomputed checksum do not match, then the data has been changed in some way.

Desirable Qualities for Checksum Computations

- The computation must depend upon every single bit in the data, so that if even one bit is changed, that will be reflected in the checksum.
- The computation must be such that it would be rare for two messages to have exactly the same checksums.
- The checksum should not reflect the original data in any obvious way.

Desirable properties (cont.)

The third property implies that an attacker would not be able to figure out how to manipulate the data so that the cryptographic checksums for the valid data and the corrupted data would wind up being the same. Another way of stating this is that it would be difficult for the attacker to reverse engineer the checksum computation.

Hashing functions

- Checksums are computed using hashing functions.
- Hashing functions are one-way functions. This means that the ciphertext (i.e., the checksum) cannot be used to reconstruct the plaintext.
- The checksum (the ciphertext) is much smaller than the plaintext.

Hashing functions (cont.)

- Hashing functions provide a kind of digital fingerprint.
- When we take a fingerprint, we lose a lot of information about the person.
- Still fingerprints and checksums are useful despite the information that is lost.
- Checksums are sometimes called cryptographic or digital fingerprints.

Hashing functions (cont.)

- A checksum is sometimes called:
  - A message digest, or
  - A message authentication code, or MAC
- The security of the hashing function is related to the size of the resulting checksum (in bits).
- Viega and McGraw suggest using hashing functions that produce a checksum of at least 160 bits.

Checksum Systems

- SHA-1 is a federal standard for computing checksums.
- SHA-1 does not use secret keys. The checksum is computed with a public hashing function and needs to be stored in a safe way. For example:
  - On a secure medium an attacker cannot modify
  - Encrypted on a completely separate medium from the original data

Checksum systems (cont.)

- SHA-1 uses a 160 bit digest.
- SHA-1 is known to be secure.
- Newer versions of SHA may have security problems because they have not been as thoroughly tested as SHA-1.
- Tripwire is a checksum system that works with the operating system to see if files have been created, deleted, or modified in an unauthorized way.

Hashing Functions and Passwords

- Hashing functions are often used to store passwords for users who are logging onto a multi-user system.
- When the user tries to log in with his or her established password, the login program hashes it, and compares the newly hashed password with the stored hash.
- If the two are equal, the system assumes the user typed in the right password.

Telnet and other Internet Protocols

- With Telnet, the password goes over the network unhashed.
- A packet sniffer could be used to catch the password in transit.
- Telnet authentication provides a very low bar for potential attackers to clear.
- Other protocols that have a similarly weak authentication mechanism include FTP, POP3, and IMAP.

Attacks on Hashing Computations

- A brute force attack involves finding an alternative text that will yield the same hash signature. This is usually fairly difficult because the alternative text is likely to be gibberish.
- An effective attack is called a birthday attack.

A Simple Birthday Attack

Suppose Bob and Alice enter an agreement in which Alice agrees to pay Bob $5.00 per widget. This agreement is sent to Bob with a cryptographic checksum. Bob decides not to store the original document on his server, just the checksum thinking that would be adequate evidence if Alice tries to present an alternative document ...

A Simple Birthday Attack (cont.)

In fact, Alice does try to present an alternative document which states that the agreement is that she pay $1.00 per widget. Bob thinks she will fail in a legal battle because he has the cryptographic checksum. Unfortunately for Bob, Alice's new document has the same checksum as the original and Bob loses in court.

A Simple Birthday Attack (cont.)

What Alice did (what is called a birthday attack) involved taking the original document (with the known checksum) and replacing all references to $5 to $1. Then, she systematically reformats the $1 per widget document by changing spaces to tabs and so forth. Eventually she finds a $1 document that has the same checksum as the original $5 document.

This kind of attack would be difficult with checksums of 512 or even 256 bits.

Digital Signatures for Authentication

- Public key encryption enabled the development of the technology of digital signatures.
- Digital signatures are somewhat analogous to traditional handwritten signatures.
- Digital signatures are strongly bound to the document, but weakly bound to the individual.
- A digital signature is computed, in part, using the contents of the document being signed.

Main Goals of Digital Signatures

- A signature should be proof of authenticity. Its existence on a document should be able to convince people that the person whose signature appears on the document signed the document.
- A signature should be impossible to forge.   The person who signed the document should not be able to claim that the signature is not theirs (support for non-repudiation).

Main Goals (cont.)

- After the document is signed, it should be impossible to alter the document without detection. The signature is intrinsically linked to the document that is being signed.
- It should be impossible to transplant the signature to another document. Again, the digital signature is intrinsically linked to the document that is being signed.

Figure 12.1 from Denning

- This figure shows one scheme for digital signatures that uses public key cryptography and hash algorithms (the usual technology).
- As you might have guessed, Alice wants to send a sign and encrypted message to Bob.
- Here's how it works:

Figure 12.1 (cont.)

1. Alice generates a message key, K, for symmetric encryption. Alice encrypts the message M with K, getting the ciphertext message, CM.
2. Alice encrypts K with Bob's public key-encrypting key, K_{bobpub}, getting the ciphertext key, CK. This will allow Bob to retrieve the key for decrypting the ciphertext.

Figure 12.1 (cont.)

3. Alice uses a hashing function to compute a checksum for the message, M. She then encrypts the checksum (for public key encryption) using her private signature key $KS_{Alicepriv}$. The encrypted checksum is the signature, S.
4. Alice sends CK (the encrypted message key), CM (the encrypted message), and S (the digital signature) to Bob.

Figure 12.1 (cont.)

5. Bob uses his private key, $K_{bobpriv}$, to decrypt CK. This gives him the key, K, that Alice originally sent to encrypt the message, M.
6. Bob uses K to decrypt CM (the encrypted message) to get the message, M.
7. Bob uses Alice's public signature key $KS_{Alicepub}$ to validate S, the digital signature. This requires that Bob use the hashing function to hash the message M. The resulting checksum should be equal to the decrypted signature, S.

Figure 12.1 (cont.)

This technology of digital signatures uses:

- A hash function to help generate the digital signature, S.
- Symmetric (secret key) cryptography to encrypt the message, M.
- Public key cryptography to share the secret key used to encrypt and decrypt the message, M.
- Public key cryptography to encrypt and decrypt the digital signature, S.

Pretty Good Privacy (PGP)

This is how Bob and Alice would accomplish the same goals using a user-friendly e-mail encryption system called Pretty Good Privacy (PGP):

1. Alice composes an e-mail message to Bob. She clicks on a button or menu item that says “send signed and encrypted”.

Pretty Good Privacy (cont.)

2. The encryption system prompts Alice for a password. The password unlocks her private signature key, which is stored encrypted on disk or on a separate storage medium.
3. The encryption system looks up Bob‘s public key (for encrypting messages sent to Bob under public key cryptography) in Alice's address book or on her “digital key ring” which is stored in a file on her disk.

Pretty Good Privacy (cont.)

3 (cont.). The digital key ring generates K, the symmetric key that will be shared with Bob, and computes CK, the encrypted key, CM, the encrypted message, and S, the digital signature. It puts the message in the outbound queue.

4. When the message shows up in Bob's inbox, he clicks on a button to read the message.

Pretty Good Privacy (cont.)

5. Bob's encryption system prompts him for a password, which unlocks his private key. It decrypts CK to retrieve K, and then uses K to decrypt CM, to get the message M.
6. The encryption system on Bob's machine then looks up Alice's public signture key in Bob's address book or key ring. It validates her signature S. The decrypted message is displayed to Bob along with an indication as to whether the signature was valid.

Pretty Good Privacy (cont.)

Alice and Bob each have two digital key rings when they use PGP. They have a private ring that holds their private keys and a public ring that holds their own public keys and the public keys of those they are communicating with.

The key rings are implemented as files stored on the hard drive or on a diskette.

Diffie-Hellman

- Diffie-Hellman is another popular method for sharing secret keys.
- Diffie-Hellman has some similarities to the use of public key encryption to share secret keys.
- This method was developed in 1976 by Whitfield Diffie and Martin Hellman, two cryptographers at Stanford University.
- In 1997 it was revealed that British cryptographers had developed a similar idea in the 1960s and early 1970s.

Diffie-Hellman (cont.)

Here is a simple description of the Diffie-Hellman protocol that allows two parties to compute a message key for symmetric encryption without that secret ever being shared explicitly:

1. Each party independently generates a private key.
2. They each compute a public key as a mathematical function of their individual private keys.

Diffie-Hellman (cont.)

3. They exchange public keys.
4. Each party then computes a message key (the secret key) which is derived from their own private key and the other person's public key. They both arrive at the same message key.

Diffie-Hellman (cont.)

- The public keys must be computed using a one-way function (a hashing function) that makes it imposible to get back the private keys from the publicly exchanged keys.
- If an attacker has access to one party's public key and the other party's private key, the attacker could compute the message key.
- The mathematics is such that the publicly exchanged keys cannot reveal either party's private key.

Diffie-Hellman (cont.)

The mathematics is based on the following relationship:

$$y = g^x \bmod N$$

It is easy to compute y if g, x, and N are known, but it is not easy to compute x if y, g, and N are known.

The problem of finding x is called the discrete logarithm problem because x is the logarithm of y base g (mod N). For numbers that are hundreds of digits long, this is a hard problem.

Diffie-Hellman (cont.)

Here is how Diffie-Hellman works, allowing Allice and Bob to establish a secret message key. Assume that p is some prime number and g is a base number:

- Alice generates a secret key, xalice.
- Bob generates a secret key, xbob.
- Alice computes a public key yalice = g^{xalice} mod p.
- Bob generates a public key ybob = g^{xbob} mod p.

Diffie-Hellman (cont.)

- Bob and Alice exchange their public keys.
- Alice now computes the message key, K, as

$$K = ybob^{xalice} \bmod p$$

- Bob now computes the message key, K, as

$$K = yalice^{xbob} \bmod p$$

- Both Bob and Alice end up with the same key, namely:

$$K = g^{xalice * xbob} \bmod p$$

Diffie-Hellman (cont.)

In practice, very large numbers are used (several hundred DIGITS each), but here is an example using small numbers:

p = 11, g = 5, xalice = 2, xbob = 3.

- Alice computes her public key

  yalice = 5^2 mod 11 = 3.

- Bob computes his public key

  ybob = 5^3 mod 11 = 4.

Diffie-Hellman (cont.)

- Bob and Alice exchange their public keys.
- Alice computes the message key, K, as

  $K = 4^2 \bmod 11 = 5.$
- Bob computes the message key, K, as

  $K = 3^3 \bmod 11 = 5.$
- Both kend up with the same message key, namely:

  $K = 5^{2*3} \bmod 11 = 15{,}625 \bmod 11 = 5.$

Diffie-Hellman (cont.)

- Diffie-Hellman are used in several network protocols and commercial products, including PGP.
- With Diffie-Hellman, keys can be generated as needed (on the fly) and they can be discarded at the end of the conversation.

Software Developers and Cryptography

According to Viega and McGraw the most common mistakes developers make with respect to cryptography are:

- Failing to apply cryptographywhen it is needed.
- Applying cryptography in an incorrect manner when it is deployed.

Developers and Crypto (cont.)

- The most important rule is:

Never, Never implement your own cryptographic algorithms!!!

- An experienced cryptanalyst will not be deterred by the fact that an algorithm is secret (not in the public domain). Their tools do not require knowledge of the algorithm.

Developers and Crypto (cont.)

- The safest policy is to use a published, well-used algorithm that has been well-scrutinized by respected cryptographers over a period of at least a few years.

Developers and Crypto (cont.)

- Viega and McGraw note that most of the major network protocols that use encryption have been broken at least once. These include:
  - SSL (Secure Socket Layer) version 2 – this should never be used.
  - SSH (Secure Shell Protocol) version 1 – this should be avoided.
  - MS's Point-to-Point protocol used in MS's Virtual Private Network (VPN).

Developers and Crypto (cont.)

- It is not only important to use well-known encryption algorithms, but also to use well-scrutinized implementations of those algorithms, because the algorithms could be implemented incorrectly.
- Developers also need to understand the legal framework surrounding cryptography. The laws have been weakened somewhat concerning shipping strong crypto stuff overseas, but there are still laws governing this domain.

Developers and Crypto (cont.)

- The best bet, in terms of avoiding legal complications, is to use off-the-shelf, freely available cryptographic packages.
- Handout with reviews of two cryptographic libraries.