Holub Associates
www.holub.com

Crypto 101

An Introduction to Cryptography

Allen I. Holub
www.holub.com

Basic Principles

- **Secrecy ≠ Security.**
  - ***Secrecy***: You can't find the safe.
  - ***Security***: You can't open the safe, even if you know how it works.
  - **Secret systems are never secure!**
    - The best way to assure that an encryption algorithm is secure is to have thousands of knowledgeable people try to break it.
- **Security ≠ Technology**
  - Security comes from well-thought-out protocols (in the diplomatic sense).
  - Technology only gives you a means to implement a portion of the protocol.

Security is about risk and liability

- If the cost of fixing a security breach is higher than the cost of writing off the loss, businesses will take the loss.
- Security is all about lowering risk to a reasonable level, not eliminating risk.
- Ultimately, security comes from a "web of contracts" (in the legal sense) that impose liability when security is compromised.
  - E.g. Insurance is an important component of a secure eBusiness system. (SSL ≠ security).

Crypto ≠ Security

- There's no such thing as "magic crypto fairy dust."
- Crypto solves several focused problems, but it does not make an application secure.
- A hacker can attack you application over a secure channel just as easily as over an insecure one.

What Crypto Gives You

- ***Access control***:
  - Only authorized individuals can access the it.
- ***Confidentiality***:
  - Only authorized individuals can read the text.
- ***Authentication***:
  - The writers are who they say they are.
- ***Non-repudiation***:
  - The writers can't claim they didn't write it.
- **Integrity**
  - The document you received is the one I sent.

The Cast

- **Alice sends a message.**
- **Bob receives a message.**
- **Eve eavesdrops on the message.**

Definitions

Code	**"The purple zeppelins flap gently on Thursday."**
Cipher	**A mathematical transformation of the text.**
Plaintext	**The text to be obscured (is really just binary.**
Ciphertext	**The obscured text.**
Encipher/Decipher	**vs. Encrypt/Decrypt.**
Cryptographer	**Invents codes/ciphers.**
Cryptanalyst	**Breaks codes/ciphers.**
Cryptologist	**Studies codes/ciphers.**

How long will it take?

- **Not: "is it breakable?" But: "how long will it take to break it?“**
  - **Will the information have value at that time?**
- **Consider a 4-wheel combination lock. How long to try every combination?**
  - **10,000 possibilities (~13 bits), 1 every 2 seconds == 20,000 seconds (~5.5 hours)**
  - **2 people, each trying ½ the codes: 2.750 hours**
  - **4 people, each trying ¼ the codes: 1.375 hours**
  - **10,000 people, each trying 1 code: 2 seconds**

Cost of a Brute-Force Attack

- Breaking a cipher is a function of:
  - number of possible keys (10,000 possibilities = ~13 bits)
  - cost of the hardware (number of processors)
  - time
- Given enough time or enough money, you can crack anything.
  - Will the value of the text outlive the time required to break the encryption?

In the Beginning

- There was the Captain Midnight Mystery-Dial Code-O-Graph.

Caesar Cipher

- To encode, add a constant value to each letter:
  'A' + 2 == 'C'
  'B' + 2 == 'D'
  ...
- To decode, subtract the same constant value.
- UNIX Rot13.
- Trivial to break, even if a more complex transposition is used (a'la Captain Midnight).

A More-Secure Cipher

- Assign values to Letters:
  01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 ...
  a b c d e f g h ij k l m n o p q r s t u v w ...
- Pick a page in a book as a key (the Bible is traditional).
- (Plain text + key text) MOD 25 == Cipher text.
  (Cipher text – key text) MOD 25 == Plain text
- "I am Allen" enciphered with "Call me Ishmael" as a key:

I= 9	A=1	M=12	A= 1	L=11	L=11	E=5	N=13
+C= 3	+A=1	+L=11	+L=11	+M=12	+E= 5	+I=9	+S=18
12=M	2=B	23=X	12=M	23=X	16=Q	14=O	31%25=6=F

Ciphertext is MBXMX QOFZG.

The One-Time Pad

- The only known unbreakable cipher, still.
- Effectively obscures the plaintext in noise.
- Create a book, each page of which contains a random sequence of letters.
  - Blindfolds and Bingo cages.
- Both Alice and Bob must have the book.
- Use the algorithm from the previous slide.
- Send the page number along with the encoded message.
- Destroy the page after using it.
- BUT, how do you keep the book safe?

Eliminate the Pad

- Strategy: Generate the one-time pad on the fly.
- Alice and Bob own the same random-number generator.
- Rather than send the book, send the seed (the *cryptographic key*).
- Create a one-time pad from your random sequence.
- *Symmetric* (secret key) encryption:
  - Same key used for both enciphering and deciphering.

Problems with Symmetric Encryption

- Doesn't solve the Eve problem:
  - You still have to send the key somehow.
  - But at least the key is smaller than the book.
- The longer the key the longer it takes for the random sequence to repeat, and the more secure the message.
  - Cryptanalysts break codes by looking for patterns.

Secret-Key Algorithms (1)

- DES
  - developed by IBM, with vetting by NSA.
    - Not fully trusted.
  - Short (56-bit) key length.
    - Easy to break.
- 3-DES (DESede, TrippleDES, DES3)
  - Use DES three times:
    - Encrypt with one key, encrypt result with a second key, and again with a third key.
  - Effective 168-bit key is secure.

Secret-Key Algorithms (2)

- AES (replaces DES)
  - A Dutch algorithm (Rijndael), politically correct.
  - Solves many classes of problems quickly.
  - 256-bit key
- Blowfish
  - 64-bit block cipher
  - Faster than AES
  - Can be a memory Hog
  - Key length up to 448 bits

Asymmetric (Public-Key) Cryptography

- Developed in the public sector.
- RSA: Rivest, Shamir, and Adelman.
- Encipher and Decipher operations use different keys.
- The *public* key is known to everyone.
- The *private* key is known only to yourself.
- Messages enciphered with one key must be deciphered with the other.
- Vastly slower than symmetric algorithms.
  - private-key operations slower than public key operations (by orders of magnitude).
- Popular public-domain algorithms (RSA/Diffie Helman) require large keys (~2048 bits) to be secure.
  - "Eliptical Curve" (Certicom patent) requires shorter key (~150 bits).

Sending a Message

- Alice looks up Bob's public key in a central *registry* (i.e. phone book).
- Alice encrypts the message with Bob's public key.
- Alice sends the message to Bob.
- Eve can't read it because she doesn't have Bob's private key.
- Bob decrypts the message with his private key.
- This mechanism only works if the registry can be trusted.
  - Alice needs assurance that Bob's public key is indeed Bob's key, not Eve's.
- But how can Bob know that the message actually came from Alice?

Authentication and Non-repudiation Only

1. Alice encrypts the message with her private key.
2. Alice sends the message to Bob.
   - Eve can read it, but she can't modify it because she doesn't have Alice's private key.
3. Bob decrypts the message with Alice's public key, proving that the message came from Alice.
   - The registry must be trustworthy!

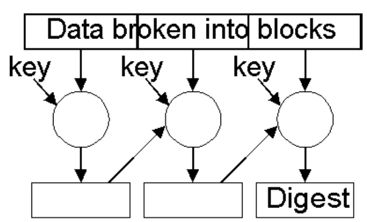
Authentication and Confidentiality

1. Alice first encrypts the message with her private key.
2. Alice then encrypts the encrypted message again with Bob's public key.
3. Alice sends the message to Bob.
   – Eve can't read it because she doesn't have Bob's private key
4. Bob decrypts the message with his private key to make it readable.
5. Bob then decrypts again, with Alice's public key to prove it came from her.
   – The registry must be trustworthy!

But Double Encryption Is Too Slow.

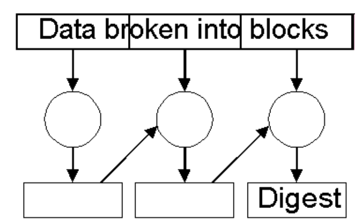
- Use a *message digest* , created using a *secure (one-way) hash function* or a *MAC (Message Authentication Code)*.
- A message digest is a fingerprint .
  – A small number (e.g. 128 bits) that uniquely identifies a large message (like a CRC).
  – Changing one bit in the input typically changes half the bits in the output.
  – Odds of two different files generating the same hash are astronomical and don't matter.
- Can't reconstruct input from the hash.
- SHA-1, MD4, MD5, etc. (algorithms are fast).

Structure of a MAC



- Uses a secret key
  – key must be known by both Alice and Bob
  – Only Alice and Bob can authenticate the hash.

Structure of a Secure Hash



- No key required
  – anybody can validate the hash
  – susceptible to “man in the middle” attack
    - Eve replaces text with her own, and creates a valid hash for that text.
    - Digital signing solves the problem.

Digital Signature (Signed Hash)

1. Alice runs her message through a secure hash algorithm to get a fingerprint.
2. Alice encrypts the fingerprint with her private key, creating a ***digital signature*** .
3. Alice sends the (unencrypted) message along with the digital signature to Bob.
4. Bob decrypts the signature with her public key, thereby proving that it came from Alice.
5. Bob runs the message through the same hash algorithm that Alice used.
6. If the resulting hash value is the same as the decrypted signature, then Bob has proven that this was the message that Alice sent.

The Whole Ball of Wax: Sending

1. Alice gets a cert from Bob, authenticates it against the CA's public key, and extracts Bob's public key.
2. Alice digitally signs the document.
3. Alice creates a symmetric ***session key*** (used once) for an algorithm supported by Bob.
4. Alice encrypts the session key with Bob's public key
5. Alice encrypts her message with the *session* key, using the symmetric algorithm that Bob likes.
6. Alice sends Bob:
   1. her cert
   2. the digital signature
   3. the encrypted message
   4. the encrypted session key.

The Whole Ball of Wax: Receiving

1. Bob uses his private key to decrypt get the session key.
2. Bob uses the session key to decrypt the message.
3. Bob authenticates the cert against the CA's public key, then extracts Alice's public key.
4. Bob verifies that the message came from Alice by decrypting the signature with Alice's public key and comparing hashes.

Digital Certificates (Certs)

- A central registry is impractical
- Digitally sign an ASCII file containing:
  - A public key (encoded as a base-64 number).
  - The DN (Distinguished Name) of the party that holds the associated private key.
  - Optional expiration dates, etc.
- Signed by a "trusted third party" called a ***CA (Certificate Authority).***
  - Think "notary."
    - VeriSign
    - Thawte (cheaper than verisign, more choices)
- Cert = the original file + the digital signature.
- Conforms to ANSI X.509 standard.

Using Digital Certificates

- Alice sends Bob her digital certificate (along with the message and/or signature).
- Bob verifies that the public key in the certificate is indeed Alice's by successfully decrypting the signature with the CA's public key.
- Bob now has Alice's public key, so can proceed as before.

Getting a Digital Certificate

- CA typically provides an applet or equivalent that does the following:
  - Alice creates a public-key/private-key pair.
  - Alice creates an ASCII file containing her *public* key only (and other X.509 data).
    - the private key is kept private, in Alice's machine.
  - Alice sends the ASCII file to the CA.
  - The CA generates a random number called a *nonce* and sends it back to Alice.
  - Alice signs the nonce, with her private key, and sends it back to the CA.
  - The CA decrypts the nonce with Alice's public key, proving that she has the private key associated with the public key in the cert.
  - The CA signs the ASCII file, and sends it back to Alice. That's the cert.

Certificate Classes

- All of the protocols discussed above require a trustworthy certificate.
  - Class 1 certs are worthless.

	Required info.	Encryption	Used for
1	Email address.	optional software	web browsing, casual email.
2	1 + successful (automated) address check.	hardware	business email, subscriptions, password replacement.
3	2 + personal presence and valid ID.	hardware	banking, content integrity (jar signing), software validation, strong encryption.

Compromised Certificates

- Once a certificate is compromised, it must be revoked.
  - The private key could have been stolen.
  - The certificate may have been obtained fraudulently.
    - Somebody inveigled three bogus Microsoft certificates from Verisign in March, 2001. They were revoked 2 (!) days later. They're still out there, somewhere.
- Revoked certs are put on a CRL (certificate revocation list).
  - pronounced "crill."
- CRL's are only good if they're used—browsers don't bother.
  - Because the CRL isn't checked, you can't really trust signed code that claims to come from Microsoft.

Certificate Chaining

Self signed "Root Certificate." Trusted Implicitly	Signed By: VeriSign Issued To: VeriSign
Authenticated using public key in root certificate	Signed By: VeriSign Issued To: ACME Inc.
Authenticated using public key in ACME certificate	Signed By: ACME Inc. Issued To: Joe Schmoe Expires: 1/1/2010

Passwords (Shared Secrets)

- **Are reasonable ways to do authentication without certificates.**
- **A password should be as secure as a cryptographic key:**
  1. **English has 1.5 bits of randomness per character.**
  2. **A 256-bit key should be protected by a 170-character password:**

     Four score and seven years ago, our fathers brought forth upon this continent a new nation: Conceived in liberty, and dedicated to the proposition that all men are created equal.
  3. **Ideally, it should have non-alphanumeric characters as well.**

Key Stores

- **Keys should be placed in a password-protected encrypted *key store* .**
- **This way, passwords can be revoked without having to revoke the key.**
- **The signing should be done by software that:**
  - **Extracts the key from the store.**
  - **Decrypts the key into memory, using a user-supplied password.**
  - **Signs some block of data.**
  - **Destroys the decrypted key by overwriting.**
- **The plaintext key should exist only for a short time, and only in physical memory**

PKI

- Keys need to be managed by a “public key infrastructure” (PKI).
- Key management for a large organization is nontrivial.
- Many security issues:
  - Access control
  - Permissions (who can modify which keys)
  - Generation (who can make certificates)

Smart Cards

- Provides an alternative to a key store.
- The private key is stored on the smart card, which is tamperproof.
- Data is submitted to the smart card for signing/encryption.
  - The key never leaves the smart card.
  - The key is inaccessible.
- The smart card can prompt for a password to provide additional security.
  - The key store is effectively on the smartcard

Some Examples

Signed Jars
SSL

© 2008, Allen I. Holub <http://www.holub.com>

Java Signed JAR

- Proves that an executable module (.class file or equivalent) you receive over the net:
  - Comes from where you expect it to be coming from.
  - Is the same code that was sent.
- Only authentication is required.

© 2008, Allen I. Holub <http://www.holub.com>

Inside a signed JAR

- A signed JAR is a .zip file that contains:
  1. A set of executable modules (.class files).
  2. A digital certificate identifying the sending party.
  3. A "manifest" that lists, for each executable module:
     1. The path, within the archive, to the module.
     2. The algorithm used to create a message digest
     3. The message digest for that module.
  4. A "signature" file containing a digital signature of the manifest file (a hash of the manifest file, encrypted with the private key associated with the public key in the certificate).

© 2008, Allen I. Holub <http://www.holub.com>

Verifying a JAR file

- The class loader:
  1. Authenticates the cert against the CA's public key.
  2. Validates the signature (proving that the file came from the expected source and that the manifest has not been tampered with) bu:
     a. decrypting the signature file with the public key in the certificate to get a hash value.
     b. rehashing the manifest using the same algorithm that the JAR-signer utility used
     c. comparing the generated hash with the one garnered from the signature.
  3. Runs each module through the message-digest algorithm specified in the manifest and compares the resulting value with the value in the manifest. If they match, then the module has not been tampered with.

SSL (Secure Socket Layer)

- Is protected by a Netscape patent (never enforced).
- Creates a secure pipe between client and server by automatically encrypting all data.
- Through a negotiation process known as a *handshake*, an SSL client authenticates the server and exchanges session keys.
  - This process is essentially the "Whole Ball of Wax," discussed earlier.
- All (expensive) private-key operations occur on the server side.
  - "SSL Accelerator" hardware offloads the work from the server itself. (e.g. F5 router.)

The Dark Underbelly of SSL

- Key strength relatively weak (40, 56, or 128-bit session key; 512 or 1024-bit RSA key.)
  - A 40-bit key is easily cracked in < 24 hours using brute force.
- No implementation that I know of authenticates the parties on the both ends of the pipe.
  - SSL supports the exchange of client and server-side certificates in order to do authentication. (*Certificate exchange*.)
  - The client-side certificates are usually ignored.
    - The Server doesn't know who it's talking to.

SSL Handshake (1)

Initiation by client (*ClientHello*)

- The client always initiates the SSL connection and handshake
- The client sends:
  - a random number
  - cipher suites it supports, in order by preference

SSL Handshake (2)

Reply by Server (*ServerHello*)

- The server sends
  - a random number
  - the cipher suite it prefers among those listed in the *ClientHello*

SSL Handshake (3)

Authentication of identify (*Certificate*)

- The server presents its X.509 certificate (which contains its public key) to the client
- The server may, but usually does not, ask the client for its certificate (*CertificateRequest*)
- The server sends a *ServerHelloDone*

Client authenticates the server

- The client verifies that the certificate is in order and has been issued by an acceptable CA.

SSL Handshake (4)

Selection of encryption algorithm

- In its *ClientHello*, the client informed the server of the cipher suites (algorithms) it supports
- In its *ServerHello*, the server indicated which of these it would like to use
- The client either accepts that suite or proposes an alternate (*ChangeCipherSpec*)
- If the client and server agree on a suite, they continue

SSL Handshake (5)

Optional Diffie-Hellman Symmetric-key exchange (*ClientKeyExchange*)

- Using the random values from the client and server, the client generates a pre-master secret, encrypts it using the server's public key, and sends it to the server.
- Each independently derives a symmetric key and a MAC key

SSL Handshake (5a)

If Diffie-Hellman isn't used, then the Client:

- creates a session key for the previously negotiated algorithm,
- encrypts it with the Server's public key,
- and sends it to the server.

SSL Handshake (6)

- Client sends a *ChangeCipherSpec* to indicate that it is ready to proceed using the negotiated cipher suite and keys.
- Client sends a *Finished*, which is the first message encrypted according to the negotiated cipher suite and keys.
  - Contains a digest of all messages exchanged so far, the derived master secret, and some other stuff.
  - Proves no tampering with the handshake.
  - This step takes a while.

© 2008, Allen I. Holub <http://www.holub.com>

SSL Handshake (7)

- Server sends a *ChangeCipherSpec* to indicate that it is ready.
- Server sends a *Finished*
  - Like the client *Finished*, contains a digest of all messages, the master secret, and some other stuff
- Data is now exchanged, encrypted using the session key and verified with a MAC.

© 2008, Allen I. Holub <http://www.holub.com>

The Kerberos Protocol

M12.1 HERAKLES, KERBEROS

© 2008, Allen I. Holub <http://www.holub.com>

Two Basic Concepts

- KDC (Key Distribution Center)
  - A trusted server
  - IETF Standard for communicating with the KDC.
    - www.ietf.org/rfc/rfc1510.txt
- Principals
  - Two entities who want to communicate
    - Alice and Bob
  - They both are clients of the KDC

Step 0: Initialization

- Alice and Bob give their passwords (and associated identites) to the KDC.
  - Must be done through a secure, out-of-band channel.
  - Might use “split channel” or “dual control”
    - Split the key into two parts, and send them independantly to the KDC using two different “couriers” (channels).

Step 1: Start the Protocol

- Alice wants to talk to Bob.
- Alice sends a plaintext message to the KDC, identifying herself and asking to communicate with Bob.
- The KDC produces a “**ticket**” for Bob holding:
  - The ticket requester (Alice)
  - The recipient (Bob)
  - A Timestamp
  - Duration (period valid after timestamp)
  - Session key (a cryptographic key good for a single communication, only).
- The KDC encrypts the ticket using *Bob’s* password as the cryptographic key.

Step 1, continued.

- Create a Ticket for Bob, holding:
  - The ticket requester (Alice)
  - The end recipient (Bob)
  - Time Stamp
  - Duration
  - Session key
  - **The recipient’s ticket** (created in previous step)
- Encrypt this second ticket with the requester’s (Alice’s) password.

We now have two tickets

- Bob's ticket
  - Created by Alice
  - Encrypted using Bob's key
- Alice's ticket
  - Holds Bob's ticket
  - Encrypted using Alice's key
- Both tickets contain the (encrypted) session key.
- The KDC now send's Alice's ticket to her via an insecure channel.
  - Eve can't see the session key or Bob's ticket. They're both encrypted using Alice's key.

Alice Sends a Message to Bob

- Alice now decrypts her ticket (she has her own password) and uses the session key to encrypt the plaintext message destined for Bob.
- She sends Bob the message and his own Ticket.
- The timestamp guarantees that the key is "fresh"
  - It can't be used at some later date by someone pretending to be Alice or Bob.

Bob Receives the Message

- Bob receives the message and his ticket.
- Bob decrypts his ticket (which was encrypted with his password by the KDC), and uses the enclosed session key to decrypt the message.
- Bob now replys to Alice
  - Thereby telling her that he
    - Successfully received he session key
    - That he's indeed Bob (otherwise he could not have decrypted the ticket).
- Bob and Alice have now established a secure session.

Single Sign On

1) Client requests ticket from KDC, supplying user name
2) KDC sends "session key," encrypted using client password.
3) Client uses that key to request tickets to a specific server.
4) Additional tickets can be requested by client using original session key, thereby eliminating need to reauthenticate the client.

Use HTTPS Everywhere!

- All links on an https:// page must be https:
- A stolen session key let's someone who hasn't logged on access your site.
  - Sessions keys are encrypted when you use https:
  - But you must use it everywhere: <img src="images.someFile">

Further Reading

- Neal Stephenson, *Cryptonomicon* (ISBN: 0380788624).
- Jon Graff, *Cryptography and E-Commerce* (ISBN: 0471405744).
- H.X.Mel and Doris baker, *Cryptography Decrypted*, (ISBN: 020161647N5)
- Bruce Schneier, *Applied Cryptography, 2nd Ed.* (ISBN: 0471117099).
- Simon Singh, *The Code Book: The Science of Cyptography from Ancient Egypt to Quantum* (ISBN 0385495323).
- Steven Levy, *Crypto: How the Code Rebels Beat the Government, Saving Privacy in the Digital Age* (ISBN: 0140244328)
- Leo Marks, *Between Silk and Cyanide: A Codemaker's War, 1941-1945* (ISBN: 068486780X)

- These slides are available at *http://www.holub.com/publications/notes_and_slides*