An Intro to Network Crypto

Jim Binkley- jrb@cs.pdx.edu
Crypto outline

- overview
- symmetric crypto (DES ... encryption)
- hash/MAC/message digest algorithms
- asymmetric crypto (public key technology)
- DH - how to start a secure connection
- KDC - a little bit on shared secret servers
- signatures - authentication with asymmetric keys
- certificates - signed public keys
- a little bit on authentication
overview

- there are MANY crypto algorithms and MANY academic network secure protocols
- how they are used in network protocols is another matter
- traditional IETF RFC said under security considerations (at end of doc)
  - “not considered here” (another F. Flub)
- new IETF POV: must consider here
symmetric encryption

- both sides know OUT OF BAND shared secret (password, bit string)
- encrypt(key, plaintext) -> cybercrud(encrypted)
- decrypt(key, cybercrud) -> plaintext
- encode/decode use same key (symmetric)
  - shared secret on both sides
- algorithms include: DES, 3DES, IDEA(128), BLOWFISH, SKIPJACK, new AES
- ssh uses 128 bit keyed IDEA
- DES key 56 bits - 0xdeadbeefdeadbeef
DES-CBC

- Cipher Block Chaining Mode
- DES processes one 64 bit block of data at a time (key is 64 bits (8 bits not important))
- CBC is used so that e.g., two 64 bit blocks in a row that are the same, do NOT produce two 64 encrypted blocks that are the same
- \( C(n) = E_k [C(n-1) \text{ xor Plaintext}(n)] \)
- requires known IV or initialization vector
pros/cons

◆ pros
  – faster than asymmetric

◆ cons
  – shared secrets do not scale in general to many users
    » more people know secret, less of a secret
  – secrets hard to distribute
  – export laws have blocked encryption software
  – DES key length too short?! (RSA challenge)
    » 2-3 bits a year used up by Moore’s law
mention briefcase man Jim

- how do get the shared secret from spot A to B
- no we do not publish it on the web
- “out of band”
- this is a GENERAL problem in secret distribution
  - is is $N^2$ for symmetric keys, but can be made linear with a server Or with public-key crypto and a server
- in general the need for briefcase man is always there else you are susceptible to a MITM attack
challenge-response with DES

- assume client/server & shared secret

client:                                        server:

------------ send ID (bob) -->

<---- send random challenge X

compute \( E = f(X, \text{DES key}) \)

------- send \( E \) to server -->

\[ \text{decode}(E, \text{key}) \]

\[ == X \]

- authentication mechanism (shared secret)
cryptanalysis means what?

- decoding the cybercrud or finding weaknesses in algorithms
- hardest if you don’t know any plaintext
- easier if you know some: known plaintext attack
- easiest if you can suggest the plaintext: proposed plaintext attack

point: end-end crypto is more secure, why?
why so?

e.g., we have crypto (like IPSEC) between a host and a border router, but not to the server.

Bart the evil one can inject data to host A and see how it is encrypted. ...
media digest algorithms aka

- hash functions OR
- one-way functions OR
- message authentication codes (if used with a shared-secret key)

- e.g., MD5, SHA, HMAC-MD5, HMAC-SHA
- MD5 - RFC 1321 (Ron Rivest)
- Secure Hash Alg. - NIST, FIPS PUB 180-1
media digest functions

- take a message, and produce a non-reproducible bit string (hash or media digest for a file)
  - \( MD(\text{msg}) \rightarrow \text{bit string} \) (128 bits with MD5)
- \( MD(\text{msg}, \text{shared secret}) \rightarrow \text{authenticator} \)
- may be used for password mechanisms
  - longer strings better, FreeBSD 128 byte passwd length
- used with signatures for efficiency reasons
  - \( MD \) algorithm faster than public-key
  - we hash the msg, then sign it
examples of MD functions

- download X and MD
  - compare X to MD to make sure you didn’t have bit rot
  - does this prevent a hacker from changing X?
- virus game1.exe upload to cwsandbox
  - get media digest - may determine they have seen it before
- nasty porno file has MD
  - police agencies have database of MDs
- used in ID system like tripwire
  - file X hasn’t changed recently
when we talk about signatures?

- we are vague …

- 1. pattern matching algorithm used to id virus bits in a file or bits on network
  - what are counter-measures on the black side?

- 2. MD signature used to type files

- 3. digital signature - public key crypto used to sign document (actually sign MD)
one time pad with MD algorithm

◆ a one-time pad is in theory:
  – an inexhaustible set of random bits of which there are 2 copies
  – briefcase man has gotten it from Alice to Bob
  – we take the msg of N bits and take the next N bits from our inexhaustible store and xor them together to encrypt
  – xor again to decrypt

◆ or we might use it just for secret key generation
  – every time we need a key, we take a phrase from a shared book and hash it with MD5 to make some bits
how about like this?

- Alice and Bob have a shared secret $K_{ab}$
- Alice computes $MD(K_{ab})$
- she xors the message with the hash,
- for the next message block she does
- $MD(MD(K_{ab}))$
- Alice needs an IV too, but never mind
- we also need a message integrity check
examples

- **MD5** - media digest 5, 128 bit string (key)
  - used with RSA signatures
- **SHA** - 160 bits,
  - used with DSS public key crypto scheme
- **MD5** has “flaws” - we may need better MD algorithms or different ideas
- **whirlpool** - new hash (512 bit digest)
ssdeep - different take on MD algorithm

- fuzzy hashing - Jesse Kornblum
- basic idea: hash can show that \( f_1 \) and \( f_2 \) are similar (or totally different)
- ssdeep is a tool he developed
- shadowserver has used it to show similarities in various malwares produced by RBN
HMAC - MD5 (or SHA)

- felt that MD5 and its like needed to be made more secure with attention to MAC function, not media digest function
- also of course, no export control ...
- HMAC - hash message auth. code, RFC2104
- roughly: $f[(K \text{xor} C1)||f[K \text{xor} C2] || \text{msg}]$
  - essentially two rounds of mac function ($f$) with cybercrud worked in as appropriate
an example - Mobile-IP

Home Agent

Mobile-IP authenticated UDP packet

registration reply

registration request

Mobile Node
Mobile-IP

- Home Agent keeps key-list of (mobile node IP addresses, per MN 128bit MD5 key)
- MN and HA share 128 bit MD5 shared secret
- compute $f(key, msg)$ and store hash in Mobile-IP registration message
- routing not setup if authentication fails
- note authentication is per IP address as ID
### Mobile-IP auth. header encapsulation

<table>
<thead>
<tr>
<th>IP</th>
<th>UDP</th>
<th>Mobile-IP</th>
<th>128-bit hash</th>
</tr>
</thead>
</table>

- authenticated part
- mobile-ip message part (app layer) includes both
  1. time bits (nonce)
  2. MN/HA ip addresses (ids)
Diffie-Hellman algorithm

- guess who invented it
- public key but doesn’t do signatures/encryption
- allows two entities that share two public numbers to arrive at a shared secret that can be used for encryption of further messages
- basis of many “session key” algorithms
- share secure channel and periodically change key (use DH to start, DES for bulk work)
DH might go like this

- Alice/Bob a priori agree on two public numbers:
  - $p$, a large ($\geq 512$ bits) prime
  - $g$, where $g < p$

- **pre-compute:**
  - Alice $S(a) = f($random$)$ $S(b) = f($random$)$ $512$ bits
  - Bob $T(a) = g^{**S(a)} \mod p$ $T(b) = g^{**S(b)} \mod p$

- Alice sends $T(a)$ to Bob; Bob sends $T(b)$ to Alice
cont:

- **post-compute of shared secret key material**
  - Alice
  - Bob
  - \( S(\text{secret}) = T(b)^{S(a)} \mod p \)
    - \( S(\text{secret}) = T(a)^{S(b)} \mod p \)

- \( S(\text{secret}) \) is the shared secret key usable for encryption/authentication and is the same because

- \( T(b) \times S(a) = T(a) \times S(b) \) as

- \( T(b)^{S(a)} = (g^{S(b)})^{S(a)} = (g^{S(a)})^{S(b)} = T(a)^{S(b)} \)
cont:

- hard to compute $S(a)$ given only $T(a)$, $g$, $p$ (hard to compute discrete log)
- may periodically recompute $S$(secret) based on use of key
  - used for time $T$ then recompute
  - used for data amount Bytes then recompute
questions re DH

◆ is unauthenticated DH subject to any active attacks?
  – if so, how?
◆ how can said attacks be fixed with what you know so far?
◆ why can’t Black Bart intercept Alice’s first packet and passively compute the shared secret?
paradigm for secure algorithm

- use asymmetric crypto to secure DH messages (e.g., RSA) or even HMAC-MD5
- use DH and handshake to setup session keys and agreement on which crypto algorithms to use for encryption/authentication
- send bulk messages with session-key derived encrypted or authenticated packets—using MD5/DES, SHA/IDEA, whatever
secure protocol paradigm then:

1. $\xleftrightarrow{}$ handshake
   DH(authenticated) gives shared secrets

2. $\xleftrightarrow{}$ handshake
   let’s use encryption X, auth. Y
   (e.g., idea/HMAC-SHA)

3. $\xrightarrow{}$ bulk per pkt data
   encrypted/authenticated data (with security header)
Perfect Forward Secrecy

**PFS defined as:**
- 1. attacker can record entire crypto session
- 2. attacker can break in and steal keys (public or private)
- 3. attacker still can’t figure out the next session

**would Alice encrypting Bob’s email with Bob’s public key have this feature?**
DH with PFS

- Alice sends Bob: [Alice, g(a) mod p] Alice (sig)
- Bob sends Alice: [Bob, g(b) mod p] Bob
- Alice sends hash(g(ab) mod p)
- Bob sends hash(1, g(ab) mod p)
- Both know the hash, and because
  - 1. DH gives private material based on initial random #
  - 2. hash is 1-way
  - we have unknowable secrets
key-escrow “foilage”

- is something a PFS protocol gives us
- doesn’t matter if big-brother knows all your keys, public/private
- protocol is unbreakable
consider this protocol

- generate key via shared-secret and hash
  - and what other properties?
- both sides do \texttt{sha(shared-secret, other?)}
- use that hash as key for privacy
- periodically hash the hash at a certain time
  - time past or bytes sent
- does this give us PFS? if not, what can we do to fix it?
KDC idea

- DH is one-way to establish shared “session keys”
- This is also about the idea of using a protocol to establish keys on both sides
- another old idea is the idea of a symmetric key-server
- you use a key-exchange protocol to get new keys from it
- KDC - key distribution center
  - traditional idea: exchange of symmetric keys
  - for indirect authentication
KDC picture

1. alice logins to KDC

2. kdc sends session-keys

3. alice sends session-keys to server bob

4. ACK or NAK
algorithm underpinnings

1. a-priori shared secret between KDC and Alice/Bob
   - 2 master keys
2. Alice gets from KDC two session keys
   - 1. one encrypted for Alice with Alice’s master
   - 2. one encrypted for Bob with Bob’s master
   - 3. this is a new Alice/Bob session key
3. Alice send’s Bob Bob’s key, and Bob decrypts with Bob’s master key
symmetric session-key system

- think of the new session keys as a 2-way base secret between Bob and Alice
- this can be used for an authentication algorithm between the two now
- this algorithm has problems (MITM and Replay)
  - also single point of failure
  - shared symmetric secrets outside a domain?
- a family of improvements include
  - Needham-Schroder (78)
- Kerberos v4 and v5
asymmetric or public-key crypto

- key generation produces (public, private) key pairs
- can give Public key away, secure private key (somehow ... and hard ...)
- two important services:
  - **signature** - append bit string that proves you signed a message, uses private key (authenticate)
  - **confidentiality** - uses public key (encrypt)
algorithms include:

- **RSA** - company and algorithm
  - invented by Rivest, Shamir, Adleman
  - key lengths, e.g., 512/1024 or inbetween
  - block size is smaller than key length
  - output will be length of key

- **DSS** - US govt. competition for RSA

- **Diffie - Hellman** (older than RSA)
  - doesn’t allow signatures/encryption
signatures

- can “sign” a message
- \textbf{sign}(M, \textit{private key})
  - but actually
  - use Media Digest algorithm to compute hash
  - say MD5 \rightarrow 128\text{ bits} (\text{hash}(M) \rightarrow \text{bit string})
  - then run private key over bit string to get signature
  - send (Msg, signature) to receiver
- receiver uses sender public key to verify
confidentiality

- you send me secure email
- obtain my public key SOMEHOW
- encrypt(Msg, public) -> encrypted message
- OK, the message has to be ASCII ...
- I decrypt with my private key
- ? how did you get my public key
- ? what if Joe spoofed me with his public key and you sent him a msg for me
big news (well maybe not)

- public, private keys are cybercrud
- one must make sure public key is somehow truly associated with party X
- and not party Y spoofing party X
- known as “man in the middle” attack if that happens
- various schemes exist for acquiring public keys (ssh/ssl/pgp, including “certificates”)
so note four operations

- **sign** (mac hash) with SELF private key
- **verify** (mac hash) with OTHER public key
- **encrypt** with SELF | OTHER public key
- **decrypt** with SELF private key
  - definitely not OTHER, else bad news
- RSA can do all 4. DSS can do sign/verify
Certificate Authorities

- it is presumed that one way to solve the problem of public key distribution
- is to get a signed public key from a trusted 3rd party
- call that node a CA - certificate authority
- nodes need the CA’s public key to start with
- can verify “certificate” signed by CA
- certificate = Joe Bob’s public key, CA sig
certificate then roughly

- your public key
- your name
- a possible timestamp (it expires at some point)
- signature over all of the above
- you need signer’s public key to verify
  - who signed signer’s certificate?
certs, cont.

- certificate can be stored anywhere
  - only CA can generate them
- CA doesn’t have to be accessible
  - but would be if network database of course
- so why don’t we have CAs?
  - netscape supports certificates and there are a few (verisign)
  - “cross-certificate” as opposed to hierarchical cert. may not be possible in some cases
X509 certificate

- version
- serialNumber - with CA’s name, ids cert.
- signature - (not the signature), names algorithm used to compute signature
- issuer - name of CA
- validity - how long it lasts
- subject - name of user
X 509 cont.

- subjectPublicKeyInfo - contains algorithm identifier AND public key
- ETC.
- encrypted - (the signature)
certificate formats - > 1 kind

- a few kinds out there at the moment
- X509 (e.g., netscape/web)
  - may be quite large
- RSA may be available in DNS
  - call ‘em DNS certificates
    - sign user name/IP/DNS names
- PGP has its own kind
bottom line:

- a certificate is basically a signed public key
- (public key, name, timestamp, signature)
- what good are they?
- authentication mechanism
- if widely deployed, could replace passwords
- ask how they are stored?
  - if stored on computer, and computer crashes ...?
  - and where is your private key stored too?
principles of authentication

- something you know
  - a password/passphrase/PIN number
  - “abracadabra”
- something you have
  - an object, a VISA card, a “dongle”, a smart card, a physical key
- something you are
  - your fingerprint/retina pattern
- combining these usually improves security
  - Pin # and VISA card
passwords - words while passing thru

- password mechanisms include:
  - 1. passwords used as authentication;
    - e.g., with DES on UNIX (prove you know shared-secret)
  - 2. authentication done as plaintext over network
    - telnet/ftp/pop/http basic authentication
  - 3. advanced password algorithms:
    - one time password or variations on that theme
    - challenge-response with a
    - hw token (counter or timestamp)
passwords

- classic password algorithm:
  - type in a string (blank the screen)
  - convert the string via DES/MD algorithm to a hash
  - compare the hash to a saved hash in a file
  - better: hash a fixed known thing that is somehow unique to user (userid ...)
  - this helps rule out on-line brute force comparison that can match > 1 user in a password file
password problems

- the password is weak
  - force the user to type in a stronger password
- the user writes the password down
  - on a card and then launders it!
  - or puts it on a yellow stickie on his monitor
- dictionary attacks on passwords
- brute-force attacks because password file is easily available
  - exploit gets it or multi-user system makes it easy to get to
password problems cont.

◆ variation on weakness
  – the password set of characters is too limited
  – too short
    » uppercase only
    » a 4-digit PIN number
  – mathematically not terribly random
    » 256-bit space with ASCII means you lost half your space (7 out of 8 bits)

◆ a random # is best, expressed in hex
password attacks, cont.

◆ someone sees you type it in
  – PIN number in a public place ...
  – of course, in that case, there are 2 authentication mechanisms

◆ if attacker can obtain password file
  – they can take their time guessing to see if they can match Alice/Bob/other users hash
  – off-line attack
  – sometimes on-line attack may be done
  – this is why you get 4 tries and then the bank machine eats your card (or login slows down)
password meta-problems

- user has many passwords
  - different for every computer
  - hard to remember
- which is why security is:
  - usually not helpful in terms of “ease of use”
  - consider the W98 “hit ESC to get around the password”
- not a good system in several variations
  - they make you change your password every 30 days
  - you vary between “hi” and “there”
  - what is your Mother’s Maiden Name?
ARE THERE BETTER SCHEMES?
- yes, but they are uncommon
- combine > 1 of the basic authentication ideas
- one-time passwords/hardware tokens
- why certificates are better, aren’t they?
password case #1 - ssh guessing

- password-based protocols suffer from RANDOM machine-based distributed password guessing attacks.
- ssh/windows login (fileshare)/sql login
  - probably not telnet so much anymore … (still there though)
- what are e.g., ssh counter-measures?
- what are botnets doing about it?
trust relationships are

- fundamental to distributed secure systems
- understand the trust relationship 1st
- then design the system
- risk alleviation systems may be able to takeover when trust relationships are too hard
  - bank card is stolen - only out $50
- trust relationships consist of
  - us (or us1 plus us2) versus them
- e.g., every computer cannot trust every other computer on the Internet by definition
- interior lines are important
study questions

◆ given an encryption algorithm like DES, could you design a key establishment protocol that computes a new shared secret between Alice and Bob?

◆ how do you protect a private-key on-line, on a multi-user o.s.?

◆ what issues can you think of with storing keys on a computer?

◆ cryptanalysis is made easier by doing what? where possible.
one more question:

◆ your bank has just deployed a new wonderful eye-ball scan authentication technique
  – scan eyeball and store in computer file like so:
    » (name, eyeball-scan-bits)
  – user at ATM has eye-ball scanned, compared with bits on computer over network to authenticate

◆ how many ways can you think of to attack this system?

◆ what problem previously mentioned does this sound like? is it the same problem?
modest homework request

- get a partner in class
- exchange email addresses
- install GNUPG (pgp in modern guise)
- now send each other a SEKRAT MESSAGE
  - that is signed
  - and encrypted
- be the 1st on the block to become a GNUPG user