Cryptography 101, Part 2

9/7/2011
Outline

- Homework Solutions
- Crypto Dos and Don'ts
- Block Ciphers
  - Modes of Operation
  - CTF Problems
- A CTF Warm-up
- More homework!
Homework Solutions

Gold stars for everyone!
def gogo():
    from PIL import Image
    import hashlib
    im = Image.open("SNight.png")
    im2 = Image.open("SNightOrig.png")
    s=""
    for i in range(0,1024):
        for j in range(0,768):
            t1 = im.getpixel((i,j))
            t2 = im2.getpixel((i,j))
            if t1 != t2:
                s+=chr(im.getpixel((i,j))[3])
    print s
    sha = hashlib.sha1()
    sha.update(s)
    print sha.hexdigest()

>>> gogo()
HASH ME: These are the droids you are looking for.
87f0a993d63a468317740d788f08d1802aaca868
Homework Solution 2

Cdokw www tssmpfov y tk kwh hitq zqjez sx guxvato ty dij guwvwx wf tdi tozo, cfj wx hwzabs rqlnqfg ps vc: arew uz lwegw gtj jsj xweliv wzq lnm toko zsd wkkzmj www jsmhkfm, jmt ex zop rq hoklunik cd gqfbbmjswxaczw kf ob, 'snz azof mu lnm rsa sx o nsqc,' zpgucll Oxmew 'cqlhkyl dugvmxmk on ggbhitkbgaoj?'

...
DO: Obey export controls (http://www.bis.doc.gov/encryption/)
DO: Follow FIPS standards (http://www.itl.nist.gov/fipspubs/)

DON'T: Write cryptographic code
DON'T: Use the wrong tools
DON'T: Use tools in the wrong way
DON'T: Be a victim of stupidity, stay informed and up-to-date
DO: Use SHA-256, SHA-512
DO: Switch to SHA-3 soon (est. NIST approval 2012)
DO: Use a hash function when you can securely distribute $H(x)$ then receive value $x'$ insecurely and want to verify $x'$ equal to $x$
DO: Salt hashes when appropriate

DON'T: Use MD2, MD4, MD5, RIPEMD, SHA-0, SHA-1
DON'T: Use a hash function as a symmetric signature
DON'T: Reuse salt values
Things they don't tell you...MAC Edition

**DO:** Use HMAC-SHA256, HMAC-SHA512
**DO:** Always use a MAC to authenticate encrypted data
**DO:** Verify authenticity of encrypted data before decryption
**DO:** Guarantee two different messages cannot result in the same input to the MAC function.

**DON'T:** Use CBC-MAC, HMAC-MD5/SHA1, DAA, Poly1305
**DON'T:** Allow side-channel timing attacks when verifying MAC
**DON'T:** Perform partial verification of MAC
Things they don't tell you...Block Edition

DO: Use AES-256
DO: Always pad plaintext input to the appropriate block size
DO: Use CTR, XTS mode
DO: Use the appropriate cryptographically random IV

DON'T: Use AES-128, AES-192, Blowfish
DON'T: Even think about using DES or Triple-DES
DON'T: Ever use a block cipher in RAW mode
DON'T: Even think about using ECB mode
DON'T: Use modes that provide both authentication and encryption (i.e. CBC and CFB mode)
DON'T: Ever reuse IV's
DO: Use a 2048-bit RSA key with public exponent 65537 and SHA256
DO: Apply symmetric encryption separately to the random key and your message
DO: Use PKCS 2.1 padding

DON'T: Use DSA or Elliptic Curve Signature Schemes
DON'T: Ever use the same RSA key for authentication and encryption
DON'T: Use PKCS 1.5 padding
DON'T: Ever use RSA without appropriate padding
Things they don't tell you...SSL/TSL Edition

DO: Understand SSL is currently our only complex attacker friendly option.
DO: Use SSL 3.0, TSL 1.2
DO: Include hash of asymmetric signature key with client side software
DO: Use SSL/TSL to secure network/internet communication

DON'T: Use SSL 1.0, SSL 2.0, TSL 1.0, TSL 1.1
DON'T: Blindly trust every certificate authority
DO: Consult a professional cryptographer if you are dealing with hardware with physical access
DO: Consult a professional cryptographer if you require energy efficiency
DO: Consult a professional cryptographer if you need maximum throughput rates
DO: Consult a professional cryptographer if you need to transmit minimal information
DON'T: Be afraid to ask for help
DON'T: Ignore any of the above advice
Block Ciphers

● Ideal Block cipher = totally random permutation from n-bit strings to n-bit strings.
  ○ Infeasible in practice:
    ■ Number of permutations of N bits = $2^N$
    ■ Number of mappings from N bit strings to N bit strings = $(2^N)!$
    ■ Number of bits require to specify a key for such a mapping = $\lg((2^N)!) > (n-1)2^{n-1}$
    ■ Exponential key size :(
● Modern block ciphers use an Iterated Cipher
Block Ciphers

- **Iterated Ciphers**
  - Require invertible function $g$
  - Key schedule $K$
  - Number of rounds $N$

- **Feistel Cipher**
  - Uses specific round function
  - Does not have to be invertible

- **List of Feistel Ciphers:**
  - Blowfish, Camellia, CAST-128, DES, FEAL, ICE, KASUMI, LOKI97, Lucifer, MARS, MAGENTA, MISTY1, RC5, TEA, Triple DES, Twofish, XTEA, ...
AES Block Encryption

- KeyExpansion—round keys are derived from the cipher key
- Initial Round
  1. AddRoundKey—each byte of the state is combined with the round key using bitwise xor
- Rounds
  1. SubBytes—a non-linear substitution step where each byte is replaced with another according to a lookup table.
  2. ShiftRows—a transposition step where each row of the state is shifted cyclically a certain number of steps.
  3. MixColumns—a mixing operation which operates on the columns of the state, combining the four bytes in each column.
  4. AddRoundKey
- Final Round (no MixColumns)
  1. SubBytes
  2. ShiftRows
  3. AddRoundKey
Modes of encryption

- There are multiple ways that you can implement your block cipher:
  - Electronic Codebook (ECB),
  - Cipher-Block Chaining (CBC),
  - Cipher Feedback (CFB),
  - Output Feedback (OFB),
  - Counter (CTR)
- Also different padding schemes
ECB: Electronic Codebook

- Simply divide plaintext into blocks and feed each block into the block cipher encryption algorithm.
- Terrible security implications:
  - Information leaked - repeated blocks in plaintext result in repeated blocks in ciphertext.
  - Vulnerable to block swapping.
    - Forge a ciphertext:
      - Given P = AABA, receive C = ZZYZ.
      - Forge C = YZZZ for P = BAAA.
For large files containing many blocks, the patterns found in the ciphertext can be very obvious.
CTF Problem - CSAW 2010, Crypto Bonus

- Users allowed to log into system with only their username
  - Root and Admin are not allowed!
- Upon authentication, they are presented with an authentication token (an encryption of the timestamp, username, and puzzle name)
- Each auth-token only lasts 5 minutes!
- Goal: Construct a correct authentication token for Root
CTF Problem - CSAW 2010, Crypto Bonus

EBC Block Swapping!

● Attempting to authenticated as "AAAAAAAA" (8 A's) results in a token:
  ○ ... A1 B2 C3 D4 A1 B2 C3 D4 ...

● As "AAAAAAAAAAAAAAAA" (12 A's)
  ○ ... A1 B2 C3 D4 A1 B2 C3 D4 A1 B2 C3 D4 ...

● How about "AAAAAAAAadmin"?
  ○ ... A1 B2 C3 D4 A1 B2 C3 D4 F1 2E BF 11 12 3D F6 18 ...

● Construct our token:
  ○ [Encrypted prefix] F1 2E BF 11 12 3D F6 18 [Encrypted suffix]

● Win.
CBC: Cipher-Block Chaining

- Encryption depends on previous block.
- No repetition like in ECB (due to IV and feed-forward)
- Can decode in parallel (since you will have access to the previous block's ciphertext and the key)
- Vulnerable to bit-flipping!
Bit-flipping in CBC mode

- Identify target bits in plaintext.
- Find corresponding bits in previous block's ciphertext.
- $\text{Ciphertext}_0 \ XOR \ \text{Decrypt(} \text{Ciphertext}_1, \text{key}) = \text{Plaintext}_1$
- If you can modify $\text{Ciphertext}_0$, you can modify $\text{Plaintext}_1$.
- Flipping a bit in $\text{Ciphertext}_0$ will flip the corresponding bit in $\text{Plaintext}_1$.
- $\text{Plaintext}_0$ is corrupted in the process (OK if no vital information was stored there!)

http://blog.gdssecurity.com/labs/tag/ctf
Users are presented with an auth token
Token is AES encryption of (Username, Teamname, Puzzle Name, Access level).
The access level was set to 5, and teams were told they needed to obtain access level 0.
Solution:
○ Use bit-flipping to change the decrypted value from 5 to 0 without actually decrypting the ciphertext
Goal: Make 0x05 become 0x00 in decrypted text
Solution: 0xa8 XOR 0x05 = 0xad

Resulting plaintext will have role=0
RSA

- Based on the factoring problem for large numbers
  - Choose $n = p \cdot q$ for primes $p$ and $q$
  - Compute $\phi(n) = (p-1)(q-1)$
  - Choose $e$ in $(0, \phi(n))$ s.t. $\gcd(e, \phi(n)) = 1$
  - Calculate $d = e^{-1} \mod \phi(n)$
- Results:
  - $(m^e)^d \mod n = m$
- Why?
  - Fermat's Little Theorem
Time for a CTF Problem!

Taken from DEFCON 18
Decrypt Please.

http://www.mediafire.com/?ibrrtnibr4pspuu

http://tinyurl.com/3bua759

Ocmln. up.'g.bjf abanfoco odrgne er yd. ypcjt d.p. /,,nnw urp yd. mroy lapy=v Ydco y.qy ,ao ,pcyy.b gocbi a ol.jcan t.fxrapew br bry .pirbrmcjw frg aoodayv WdcbyV Yd. t.f frg ap. nrrtcbi urp co yd. bam.oat. ru ydco t.fxrapev WzdcbyV
Hint 1: Frequency Table

= 39 (16.81%)
' = 1 (0.43%)
, = 3 (1.29%)
. = 20 (8.62%)
/ = 1 (0.43%)
= = 1 (0.43%)
O = 1 (0.43%)
V = 2 (0.86%)
W = 2 (0.86%)
Y = 2 (0.86%)
a = 13 (5.60%)
b = 11 (4.74%)
c = 13 (5.60%)
d = 11 (4.74%)
e = 39 (16.81%)
f = 7 (3.02%)
g = 5 (2.16%)
i = 3 (1.29%)
j = 4 (1.72%)
l = 3 (1.29%)
m = 4 (1.72%)
n = 7 (3.02%)
o = 13 (5.60%)
p = 11 (4.74%)
q = 1 (0.43%)
r = 16 (6.90%)
t = 6 (2.59%)
u = 4 (1.72%)
v = 3 (1.29%)
w = 3 (1.29%)
x = 2 (0.86%)
y = 15 (6.47%)
z = 1 (0.43%)
Hint 2: Common word?

Note that "yd." appears multiple times in the ciphertext.
Solution
This cipher was created by typing on a DVORAK keyboard using QWERTY key positioning. The translated text is:

Simple frequency analysis should do the trick here [well, for the most part]. This text was written using a special keyboard, not ergonomic, you ass. <hint> The key you are looking for is the namesake of this keyboard. </hint>

The final solution to the problem was "DVORAK"
Homework Problem 1

How could you simultaneously ensure privacy and authenticity using public key crypto?

Give your answer using some or all of the following:

- Functions: Encrypt(msg,key), Decrypt(msg,key)
- Strings: Pub_A, Priv_A, Pub_B, Priv_B, Msg
Homework Problem 2

(Taken from interview question):

Consider an authentication scheme where multiple users need to authenticate to the server prior to communication. There is a shared key that only the users and server know. Authentication occurs as follows:

1) User sends Challenge to server

2) Server proves it's identity by replying Encrypt(Challenge, SharedSecret)

3) User verifies server by encrypting Challenge and comparing with the received message

4) Server sends Challenge2 to user

5) User authenticates by replying Encrypt(Challenge2, SharedSecret)

4) Server verifies user by encrypting Challenge2 and comparing with received message

The encryption system itself is unbreakable (think OTP). However, the implementation is broken - point out the flaw that would allow anyone to authenticate without actually knowing the SharedSecret. Can you think of a simple way to fix it?
Announcements!

CSAW CTF 2011 Qualification Round Friday, September 23, 2011 20:00 EDT (-4) - Sunday, September 25, 2011 20:00 EDT (-4)