Cryptography Review

Andrew Zonenberg
4/8/2010

Basic cryptographic primitives

- Randomness etc
- Encryption / decryption
- Hashing
- Key agreement
- Signing / verification

Randomness

- Keys etc should be unpredictable
- Pseudorandom vs random
- rand() == bad (remember red bomb wire)
- /dev/random is good, radioactive decay better...
- Debian OpenSSL bug etc

Avalanche effect

- Flipping one input bit (message or key) should ideally flip a random half of the output bits
- Mixing is critical - all output bits should depend on all inputs
- But mixing alone isn't enough!

```
for(i=0; i<128; i++)
     xor odd numbered bits of block with bit i of key
for(i=128; i<256; i++)
     xor even numbered bits of block with bit i of key
```

Encryption

- Symmetric - same key used both ways
- Asymmetric - separate public / private keys

Symmetric ciphers

- Block ciphers (AES, DES, Blowfish...)
- Stream ciphers (RC4...)

Block ciphers

- Maps fixed length input to same-sized output
- Chop message into blocks and pad as needed
- Multiple modes of operation

Round based ciphers

- Designing an entire cryptosystem in one massive step is very difficult
- Define a simple operation (“round function”) and apply it many times
- Each round typically uses a separate key derived from the original key by a function called the “key schedule”

ECB mode

- Electronic Code Book
- Each input block always maps to the same output
- Problems?

ECB mode - flaws (1)

- Block-level replay attacks
- Content guessing possible
- Can discern high level structure

ECB mode - flaws (2)

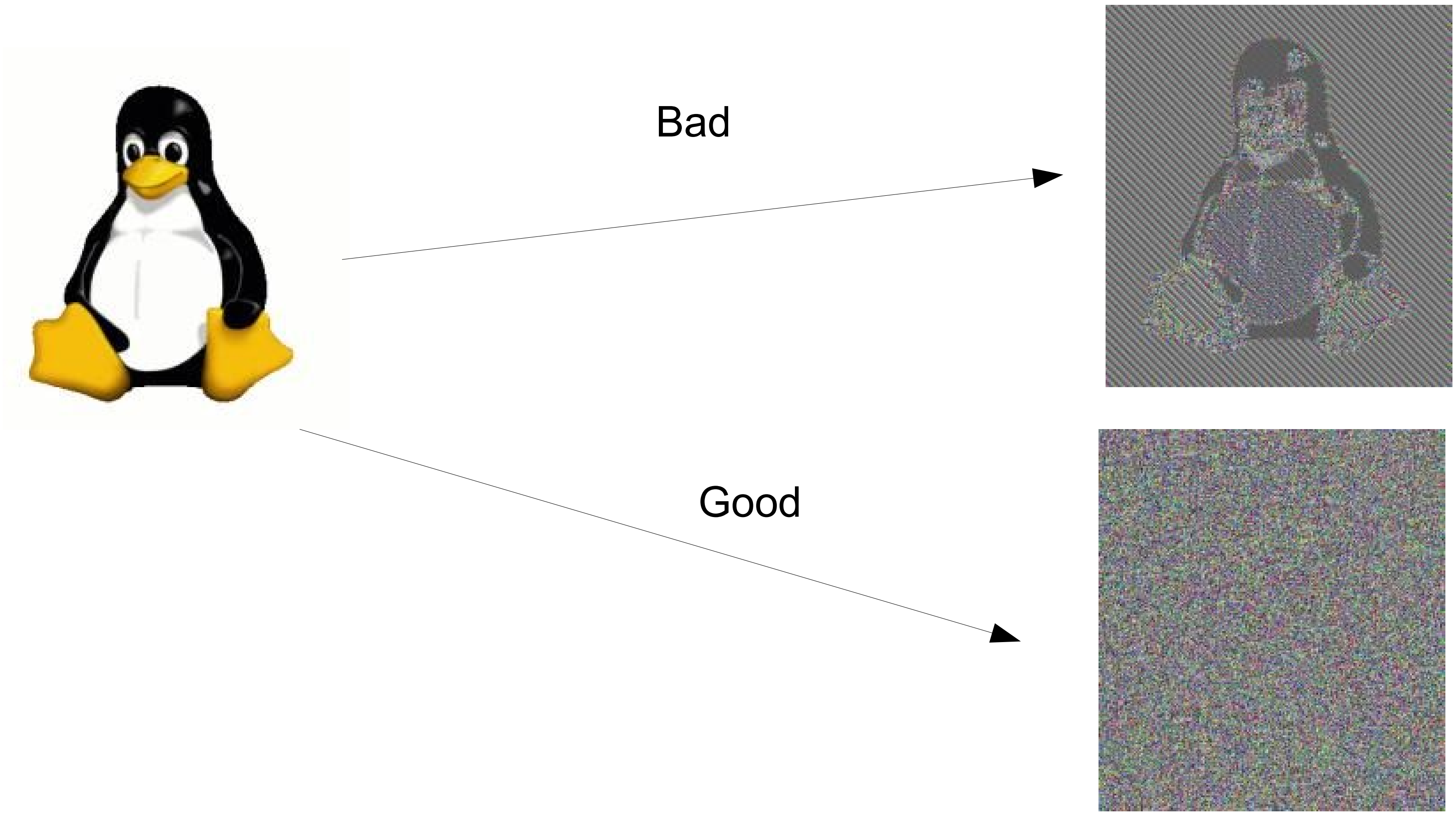
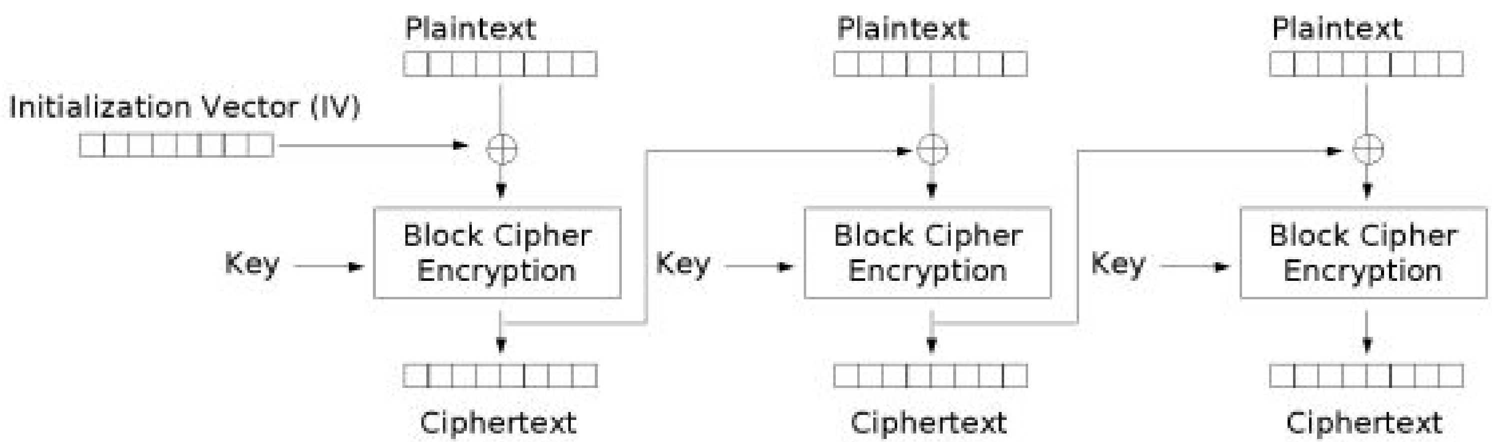


Image credit: Larry Ewing<lewing@isc.tamu.edu>, The GIMP

CBC mode

- XOR each block with previous ciphertext
- Needs unique (not secret) initialization vector for each message



Cipher Block Chaining (CBC) mode encryption

CTR mode

- aka Counter
- Turns a block cipher into a stream cipher
- Encrypt a “counter” value with our key
- XOR plaintext with resulting ciphertext
- Bump counter and repeat after N bytes
- Counter cannot ever be re-used!

A closer look at AES

- 128 bit block size, 128/192/256 bit key
- 16 byte state, 4x4 matrix
- AddRoundKey - mix subkey for this round
- SubBytes - adds nonlinearity via substitution
- ShiftRows - add diffusion by circular shifts
- MixColumns - adds diffusion by binary field mixing operation

Stream ciphers

- Generate a stream of pseudorandom data derived from our key
- XOR keystream with message
- Needs an initialization vector of some sort
- IV sharing is bad

A closer look at RC4

- Common stream cipher used in SSL etc
- Permutation S of bytes 00-FF
- Index pointers i and j
- Key can be 1 to 256 bytes, typically 40-128 bits (5 - 16 bytes)

RC4 key schedule

```
void rc4_keyschedule(
   unsigned char *key, unsigned int key_length)
{
    for (i = 0; i < 256; i++)
        S[i] = i;

    for (i = j = 0; i < 256; i++)
    {
        j = (j + key[i % key_length] + S[i]) & 255;
        swap(S, i, j);
    }

    i = j = 0;
}
```

Credit: Wikipedia article on RC4

RC4 PRNG

```
unsigned char rc4_prng()
{
    i = (i + 1) & 255;
    j = (j + S[i]) & 255;

    swap(S, i, j);

    return S[(S[i] + S[j]) & 255];
}
```

Credit: Wikipedia article on RC4

Stream cipher problems

- If IV is reused, suffers from same replay issues as ECB mode in block ciphers
- XORing two ciphertexts using the same IV gives XOR of the plaintexts
- Can flip arbitrary bits in the message easily

Stream cipher attack

- Ciphertext 1 = `0x 34069fca7fe70cf5`
- Ciphertext 2 = `0x 1b2aeb9e38a236c4`
- One of the messages is the start of an HTTP request
- The other is from an IM conversation
- Same RC4 key used for both
- Find both plaintexts and the secret keystream

Public key cryptography

- Different keys used for encryption and decryption
- Computing public key from private is easy
- Computing private from public is hard
- Messages encrypted with one can be decrypted with the other
- Very slow, huge keys

Public key encryption

- A encrypts message to B with B's public key
- Can be decrypted by B's private key only

Public key signature

- A encrypts hash of message with A's private key
- Anyone with A's public key can read it
- But only A could have produced it

A closer look at RSA

- Generate two primes P, Q
- Public modulus N = PQ
- $\Phi(pq) = (p-1)(q-1)$
- Public exponent e: $1 < e < \Phi(pq)$
  e, $\Phi(pq)$ are relatively prime
- Private exponent d: $de = 1 \bmod \Phi(pq)$

More RSA

- Encryption: $c = m^e \bmod n$
- Decryption: $m = c^d \bmod n$
- Message has size limit, typically is a symmetric session key
- Message must be padded to prevent chosen plaintext attacks etc

Flawed SSL-like protocol

- Client contacts server
- Server sends RSA public key
- Client encrypts RC4 session key to server
- All traffic is now RC4 encrypted
- Separate IV for transmit and receive traffic
- Spot the problems!

Key exchange

- Derive a shared secret between two users
- Typically cannot be used to encrypt an arbitrary message - exceptions apply ;)

Diffie-Hellman key exchange

- Select shared modulus p and base g
- A chooses secret integer a, sends $A=g^a \bmod p$
- B chooses secret integer b, sends $B = g^b \bmod p$
- A computes $B^a \bmod p$
- B computes $A^b \bmod p$
- $(g^a)^b = (g^b)^a$
- Breaking requires solving discrete logarithm problem

Exercise

- Find as many covert channels as possible in this scheme

Cryptographic hash functions

- AKA Message Digest
- Takes arbitrary length input
  (sometimes restricted to large but finite, 2^{64} etc)
- Returns fixed size output

A closer look at MD4

- 32 bit words
- Append a 1 bit, then 0s until len = 56 (mod 64)
- Append length as a 64 bit little endian integer
- Initialize state A, B, C, D
- Divide into blocks of 16 words (64 bytes)
- Process each block in sequence

Round functions

- $F = (X \& Y) | (\sim X \& Z)$
- $G = (X \& Y) | (X \& Z) | (Y \& Z)$
- $H = X \wedge Y \wedge Z$

MD4 block processing

- Save old A, B, C, D
- Round operation [abcd k s]
  a = (a + F(b,c,d) + X[k]) <<< s
- Example (round 3 of 3)

```
[ABCD  0  3]  [DABC  8  9]  [CDAB  4 11]  [BCDA 12 15]
[ABCD  2  3]  [DABC 10  9]  [CDAB  6 11]  [BCDA 14 15]
[ABCD  1  3]  [DABC  9  9]  [CDAB  5 11]  [BCDA 13 15]
[ABCD  3  3]  [DABC 11  9]  [CDAB  7 11]  [BCDA 15 15]
```

- Add old A, B, C, D to current
- Problems?

MD4 exercise

- 8e793b925ad32db390091141f6b6a11b
- Reverse the state as far as possible
- Input is 7 ASCII characters