

# Encryption Issues

# Rise and Fall of encryption

---

- Encryption provides communications confidentiality
  - as long as the encryption schema is secure
  - as long as encryption keys remain secret
- Encryption provides communications confidentiality
  - For legitimate transactions
  - For illegitimate transactions
- Encryption provides communications confidentiality
  - Not communication stealth
  - Not endpoint security

# Encryption security concerns

---

- The algorithm
  - Relative strength
  - Integration in the encryption schema
- Implementation
  - Traps and backdoors
  - Coding issues
- Use of encryption
  - Key storage weaknesses
  - Users, the usual weak link
  - Diverted use of encryption

# Encryption Algorithm Security

# Algorithm strength

---

- Time to find a key by brute force
  - 40 bits RC4
    - cracked in 32 hours in 1995 (2nd HAL challenge)
    - 3 ½ hours in 1997 (RSA Challenge)
  - RC5
    - 48 bits, cracked in 13 days by up to 7.000 machines in 1997
    - 56 bits, cracked in 270 days by tens of 1.000 machines in 1997
    - 64 bits, cracked in 1757 days by up to 330.000 machines in 2002
  - DES 56 bits
    - 1997 (RSA DES) : 90 days
    - 1998 (RSA DES Challenge II) : 39 days
    - 1998 (RSA DES Challenge II-2) : 56 hours
    - 1999 (RSA DES Challenge III) : less than 23 hours (luckily...)

RSA DES Challenge III in 1999 was successfully cracked in 23 hours while only ¼ of the key space was searched.

# Algorithm Weaknesses

---

- The RC4 example
  - Designed in 1984
  - Class of weak keys
    - Effective key size down by 5 bits [Roos95]
  - In some cases vulnerable to a related key attack
    - Invariance and known IV weaknesses [FMS2001]
    - Large keys [GW00]
  - And more ...
    - Statistical anomalies [Golic97] [FMcG2000]
    - Permutation state recovery [MT98]
    - ...

KSA = Key Scheduling Algorithm

### **Class of Weak keys**

For at least 1 out of every 256 possible keys the initial byte of the pseudo-random stream generated by RC4 is strongly correlated with only a few bytes of the key, which effectively reduces the work required to exhaustively search RC4 key spaces.

### **Invariance and IV weaknesses**

The invariance weakness is the existence of specific patterns, which are invariant with respect to the KSA. In other words, when these patterns appear in RC4 keys, they are probable to appear also in the generated permutation. Furthermore, these patterns propagate into the generated stream and consequently their occurrences (in the secret key) can be easily isolated by simple analysis of the stream (sometimes the ciphertext itself suffice). The IV weakness, is related to a popular mode of operation of stream ciphers, where in order to avoid reusing the key, it is combined with a known vector (denoted the initialization vector or the IV) and this combination is used as the seed to the key stream generator. This paper describes a practical ciphertext only attack on this mode of operation of RC4, when the combination of the key and the IV is made in a simple method such as concatenation. An interesting point is that this mode of operation is commonly used in the WEP (wired equivalent protocol) which is a part of the 802.11 standard, and this attack is applicable not only on the current (at that time) version of WEP, but also on the enhanced version WEP2.

### **Large Keys**

When the secret key is very long, RC4 is vulnerable to a related-key attack.

*Related key attack : In [cryptography](#), a **related-key attack** is any form of [cryptanalysis](#) where the attacker can observe the operation of a [cipher](#) under several different [keys](#) whose values are initially unknown, but where some mathematical relationship connecting the keys is known to the attacker. For example, the attacker might know that the last 80 bits of the keys are always the same, even though she doesn't know, at first, what the bits are. This appears, at first glance, to be an unrealistic model; it would certainly be unlikely that an attacker could persuade a human cryptographer to encrypt plaintexts under numerous secret keys related in some way. However, modern cryptography is implemented using complex computer protocols, often not vetted by [cryptographers](#), and in some cases a related-key attack is made very feasible.*

### **Statistical anomalies**

Some statistical anomalies makes it possible to distinguish RC4 stream from a random stream. It maybe relationship between bits of the cipher stream (Golic) or bias in specific RC4 states (Fluhrer and McGrew).

### **Permutation state recovery**

RSA creates a keystream used to encrypt data stream. This is done to make sure that the same key is not used for each and every data transmitted and to simulate a "random" key encryption each time. The keystream is however initialized thanks to 2 n-bits counters. Therefore the number of possible states is finite and it is possible to launch an attack to "track" the current state. In this case with a 5-bits RC4 cipher (standard is 8 however), the keyspace of  $2^{160}$  has been reduced to  $2^{42}$ .

# Algorithm Weaknesses

---

- RSA weaknesses with repeated encryptions
  - Hastad attack [Hastad85]
    - Aka Broadcast Attacks
    - Based on low encryption exponents for performance reasons
    - Efficient when a message is sent to multiple recipients who share the same public encryption exponent.
  - Simmons attack [Simmons83]
    - Aka Common Modulus Attack
    - Based on Bezout theorem
    - Efficient if Alice, Bob & Charlie use the same modulus (but different exponents of course...)

## **RSA Basics**

$N = p q$  ( $p$  and  $q$  are 2 large primes of the same size  $= n/2$ ).

$e$  and  $d$  are 2 integers satisfying  $ed = 1 \pmod{\phi(N)}$  with  $\phi(N) = (p-1)(q-1)$

$N$  is the modulus

$e$  and  $d$  are respectively the encryption and decryption exponents

$(N,e)$  and  $(N,d)$  pairs are respectively public and private keys

Encryption ( $M = \text{message}$ ,  $C = \text{cipher}$ ) :  $C = M^e \pmod{N}$

Decryption :  $C^d = M^{ed} = M \pmod{N}$  easy if you know  $d$ .

# Algorithm Weaknesses

---

- More RSA exponent based attacks
  - Wiener attack [Wiener90]
    - Makes it possible to deduce private key from public key
    - Efficient if private key exponent is "small enough".
  - Algebraic attacks via LLL algorithm [Durfee02]
    - Needs "small" encryption exponent
- And so on...
  - The "YKLM-Scheme" [YKLM01] [YKLM03]
    - Improves performance
    - Create new classes of weak keys

## **Weiner Attack**

Small exponents are interesting to improve calculation performances. However, Wiener proved that if  $d < 1/3(N^{1/4})$ , then  $d$  can easily be deduced from the public key  $(N, e)$ .

## **Algebraic attacks**

More general approach of the Wiener attack, using the LLL lattice basis reduction algorithm.

## **YKLM Scheme**

New key generation algorithm, basically design to protect against DFA (differential fault attack). However it has been proved that it creates a class of key that is vulnerable to an extension of the Wiener attack.

# The encryption scheme

---

- Using weak encryption in protocols
  - SSLv2 and RC4
- Defining weak protocols
  - WEP : the very case study
    - All possible mistakes in a single protocol
    - Potentially, very weak shared key
    - Possible weak IVs
    - No key renegotiation
    - Use of a linear checksum calculation function (CRC-32)
    - Low seed entropy makes cleartext difference attack possible
    - Authentication phase makes known cleartext attack possible

# **Encryption Implementation**

# Randomness issues & tricks

---

- Reminder : strong random generation
  - Bit generation uniformity
  - Bit generation independence
- Usual weakness
  - Impossible to generate real-time randomness today
    - Performance issues
  - Use of a random (as much as possible) seed
  - Use of mathematic function to generate pseudo-random
  - Weaknesses in both will lower the security of encrypted data.
    - Use of `rand()`, `random()` and `drand48()` to be prohibited
    - Use `CryptGenRandom` with your own seed
    - `/dev/urandom` initialization is not guaranteed by OS, therefore there may not be enough entropy
    - Etc.

# Servers verbosity

---

- Error management to guess part of secrets
- The swiss attack against SSL [Asselborn03]
  - Based on Vaudenay's padding oracle theory
  - Several conditions :
    - CBC encryption
    - Padding bytes can be calculated
  - Proved to work with Stunnel
- Bleichenbacher attack [Bleichenbacher01]
  - Generation of specific data accepted by recipient as PKCS#1 format
  - Needs real-time error generation from server
  - Proved to be efficient on most SSL implementations

## **Swiss Attack against SSL**

In CBC (Cipher Block Chaining) mode encryption, 2 blocks are sent over the network. The first one (made of  $n$  bytes  $r_1, r_2 \dots r_n$ ) is considered as an Initialization Vector and the second ( $y_1 \dots y_n$ ) is the encrypted data. If we manipulate  $r$  to obtain no "padding error" it means that once decrypted ( $y_i \Rightarrow x_i$ ),  $x_n \text{ XOR } r_n$  provided a result that matches the padding algorithm. If the padding algorithm is known and predictable (and therefore we know  $p_n$ ),  $x_n \text{ XOR } r_n = p_n \Rightarrow x_n = p_n \text{ XOR } r_n$ , we guess the last cleartext byte.

## **Bleichenbacher Attack**

Exploits old PKCS#1 format weakness to make it possible to guess the cleartext message. This is also done thanks to the padding mechanism.

# Trapdoors in key generation

---

- Trapdoors in private keys
  - Reduce size of possible keys universe
  - Used to lower encryption strength to a crackable result
- Trapdoors in public keys
  - Use of algorithm weaknesses
    - RSA weak keys and Wiener's attack
  - Makes it possible to deduce private key from public key

## **Trapdoors in private keys**

Some key generators can be coded in such a way that they are not able to generate any key of the key space, but only a subgroup of it. It may be possible, as an example, to have an engine that is capable of generating "only"  $2^{24}$  different keys instead of the  $2^{56}$  one for RSA. When you are aware of such trap it becomes easier to crack keys by brute force.

## **Trapdoors in public keys**

In specific conditions it is possible to deduce the private key from the public key. It may either be because keys belong to a class of "weak" keys or because they have specific characteristics (small private exponent). Once again it is possible for the creator of the key generation program to force the generation of weak keys.

# Backdoor block ciphers

---

- Malicious crypto implementations
  - Leaks key information to software designer
  - Several proof of concept
    - Monkey (aimed at skipjack)
    - Black Rugose (aimed at Rijnael)
  - Goes along with subliminal channels to send the key to the malicious designer

Another way for the software designer to perform malicious actions. Block ciphers algorithm may be vulnerable to some attacks "from the inside" meaning that the designer can have private key information leak. Therefore the designer can determine part or all bits of the key.

The second problem is the capability to discretely send the key to the designer of the program. This is where covert channels are useful.

# Bogus implementation

---

- Coding tricks
  - Usual issues
    - Input validation
    - Memory management
    - Best practice
  - A neverending story
    - Frequent OpenSSL bugs
    - The mbuf issues in BSD implementations of IPSEC
- Functional tricks
  - SSLv2 fallback to weak RC4
  - Encryption algorithm enforcement
    - Example : CBC for the Swiss attack

**Using Encryption**

# Key "recovery"

---

- RSA "square and multiply" makes it possible
  - With a stop watch
    - Timing attack [Kosher97]
  - With an oscilloscope
    - Power consumption attack
- RSA CRT optimization makes it possible
  - With a hammer
    - Differential Fault Attack (DFA)

## **Square and multiply**

Operations made to perform decryption involves square calculation for key bits = 0 and square + multiply for key bits = 1. Therefore it will take longer and more electric power to decrypt with key bits = 1. From there it is possible to create an attack against key containers.

## **RSA CRT**

The Chinese Remainder Theorem is used to accelerate decryption and signature operations. If an error occurs while a signature is generated it will provide a way to factor N.

# Users issues

---

- Neverending story
  - Users don't use encryption with appropriate care
    - Don't pay attention to their key containers
    - Don't pay attention to security alerts or information
  - Don't even use encryption
    - Don't check if they really communicate over SSL on the web
    - Still use insecure protocols
      - ftp, telnet, rsh !

# Malicious use

---

- Encryption of egress traffic
  - To create covert channels
  - To make information leak possible
- Encryption of ingress traffic
  - To confuse IDS/IPS and let attacks undetected
    - HTTPS is the best example