Security and Cryptography 2022

Mirosław Kutyłowski

VII. DISK ENCRYPTION

Problems

- random access – decryption of each page independently
- sector size (512, 4096 bytes versus blocksize of encryption)
- some mode to be used:
  – ECB obviously wrong
  – CBC and other modes require Initial Vector , but there is no extra space for storing IV!
- different IV's or so-called "tweaking" inside each sector
- no extra space in a sector, encryption "in place"

Malleability

CBC: if the plaintext is known, then one can change every second block to a desired value (every second block would be junk):

- recall that $C_i = E_K(C_{i-1} \oplus P_i)$
- **replace** C_{i-1} with $C_{i-1} \oplus P_i \oplus P'$
- **effect:** then the ith block decrypts to P' (while block P_{i-1} will become junk):

$$\mathrm{New}(P_i) = \mathrm{new}(C_{i-1}) \oplus \mathrm{Dec}_K(C_i) = (C_{i-1} \oplus P_i \oplus P') \oplus (C_{i-1} \oplus P_i) = P'$$

creating IV vectors (should not repeat!): Encrypted salt-sector initialization vector (ESSIV)

- $\mathrm{IV}_n = \mathrm{Enc}_K(n)$ where $K = \mathrm{Hash}(K_0)$ and n is the sector number

Encryption algorithms

- attempts to use sector- size block ciphers - unpopular
- tweaking traditional block ciphers (tweakable narrow-block encryption)

LRW Liskov, Rivest, and Wagner

– $C = \mathrm{Enc}_K(P \oplus X) \oplus X$ ($\oplus$ denotes addition in the field)

where $X = F \otimes I$ ($\otimes$ denotes multiplication in the field)

F is the additional key, I is the index of the block

– **the issue of "red herrings":** encrypting the block $F||0^n$:

$$C_0 = \mathrm{Enc}_K(F \oplus F \otimes 0) \oplus (F \otimes 0) = \mathrm{Enc}_K(F)$$

$$C_1 = \mathrm{Enc}_K(0 \oplus F \otimes 1) \oplus F \otimes 1 = \mathrm{Enc}_K(F) \oplus F$$

so F will be revealed

Xor–encrypt–xor (XEX)

– $X_J = \mathrm{Enc}_K(I) \otimes \alpha^J$

– $C_J = \mathrm{Enc}_K(P \oplus X_J) \oplus X_J$

– I is the sector number, J is the block numer in the sector and α is a generator

XEX-based tweaked-codebook mode with ciphertext stealing (XTS)

– IEEE 1619 Standard Architecture for Encrypted Shared Storage Media

– different key for IV than for encryption ("through misunderstanding XEX specification")

– deals with the sector size not divisible by the block size

– for the last block (a problem due to fixed size – one cannot use paddings!)

  i. expands the k byte plaintext with the last bytes of the ciphertext of the previous block,

  ii. the resulting ciphertext stores in place of the ciphertext of the previous block

  iii. the ciphertext from the previous block truncated to k bytes and stored as the last ciphertext

  for decryption: the missing $n - k$ bytes are recovered from decryption of the ciphertext of the last (originally) block

– **problem:** no MAC, one can manipulate blocks, something will be recovered!

Generating key for disk encryption from the password

Password-Based Key Derivation Function 2 (PBDKF2)

– $\mathrm{DerivedKey} = \mathrm{PBDKF2}(\mathrm{PRF}, \mathrm{Password}, \mathrm{Salt}, c, \mathrm{dkLen})$

– c is the number of iterations requested

– $\mathrm{DerivedKey} = T_1 || T_2 || \ldots || T_{\mathrm{dklen/hlen}}$

– $T_i = F(\mathrm{Password}, \mathrm{Salt}, c, i)$

– $F(\mathrm{Password}, \mathrm{Salt}, c, i) = U_1 \otimes U_2 \otimes \ldots \otimes U_c$ where $\otimes$ stands for xor

– $U_1 = \mathrm{PRF}(\mathrm{Password}, \mathrm{Salt}, i)$

– $U_j = \mathrm{PRF}(\mathrm{Password}, U_{j-1})$ for $1 < j < c$

slight problem: if password too long, then first processed by hash, then some trivial collisions

Password hashing competition

- organized by a group of people
- Argon2 winner
- some controversies
- design goals:
  - strictly sequential computation
  - fills memory fast
  - tradeoff resilience (smaller area results in higher time - but potentially compensated by ASIC)
  - scalability of parameters
  - number of threads can be high
  - GPU/FPGA/ASIC unfriendly
  - optimized for current processors

Argon2 key derivation function

inputs:

- message P (up to $2^{31}-1$ bytes)
- nonce S (up to $2^{31}-1$ bytes)
- **parameters**: degree of parallelism p, tag length τ, memory size m from $8p$ to $2^{32}-1$ kB, number of iterations t, version v, secret K (up to 32 bytes), associated data X (up to $2^{32}-1$ bytes)

extract-then-expand

- **extract a 64 byte value**: $H_0 = \mathrm{Hash}(p, \tau, m, t, v, y, <P>, P, <S>, S, <K>, K, <X>, X)$ where $<A>$ denotes the length of A,
- **expand** using a variable length hash H':
  - initialize blocks $B[i,j]$ with p rows ($i = 0, ..., p-1$) and $q = \lfloor \frac{m}{4p} \rfloor \cdot 4$ columns, each

$B[i,j]$ of 1kB

- $B[i,0] = H'(H_0||0000||i)$
- $B[i,1] = H'(H_0||1111||i)$
- $B[i,j] = G(B[i,j-1]||\,B[i',j']))$ where i', j' depend on the version, G is compression

- **t iterations:**
  - $B[i,0] = G(B[i,q-1]||B[i',j'])$
  - $B[i,j] = G(B[i,j-1]||B[i',j'])$
- **final**
  - $B_{\text{final}} = B[0,q-1] \oplus B[1,q-1] \oplus .... \oplus B[p-1,q-1]$
  - $\text{Tag} = H'(B_{\text{final}})$

variable length hashing

– H_x a hash function with output of length x

– if $\tau \leq 64$, then $H'(X) = H_\tau(\tau || X)$

– if $\tau > 64$:

$$r = \lceil \tau/32 \rceil - 2$$

$$V_1 = H_{64}(\tau || X)$$

$$V_2 = H_{64}(V_1)$$

...

$$V_r = H_{64}(V_{r-1})$$

$$V_{r+1} = H_{\tau - 32r}(V_{r-1})$$

$$H'(X) = A_1 || A_2 || \ldots || A_r || V_{r+1}$$

where A_i are the first 32 bits of V_i

compression function G

- Blake2b round function P used
- $G(X, Y)$ on 1kB blocks X, Y:
  - $R = X \oplus Y$, R treated as 8×8 matrix of 16-byte registers $R_0, \ldots., R_{63}$
  - $(Q_0, \ldots., Q_7) = P(R_0, \ldots., R_7)$

    $(Q_8, \ldots., Q_{15}) = P(R_8, \ldots., R_{15})$

...

$(Q_{56}, ..., Q_{63}) = P(R_{56}, ..., R_{63})$

– $(Z_0, Z_8, ..., Z_{56}) = P(Q_0, Q_8, ..., Q_{56})$

$(Z_1, Z_9...., Z_{57}) = P(Q_1, Q_9, ..., Q_{57})$

...

$(Z_7, Z_{15}, ..., Z_{63}) = P(Q_7, Q_{15}, ..., Q_{63})$

– finally output

$Z \oplus R$

Format preserving encryption

disk encryption is one of the cases of Format Preserving Encryption:

the size of the output must be exactly the same as the size of the plaintext

example: encrypting credit card numbers in a database

challenge: redesign of block ciphers to small blocks is hardly possible

Generic methods

Random walks

- a sequence of simple transformations determined by the (long) key, each transformation is a permutation
- concept based on a random walk in a (relatively small) graph
- based on concept of rapid mixing of Markov chains and approaching the uniform distribution

Cycling

example: having a block encryption scheme Enc with blocks of length k create a FPE for block length $k-1$:

- append input x with a zero: $x' := x\|0$
- $c' := \mathrm{Enc}_K(x')$
- if $c' = c\|0$, then output c
- else $c'' := \mathrm{Enc}_K(c')$
- if $c'' = c\|0$ then output c
- else continue in the same way until getting a ciphertext of the form $c\|0$

decryption:

- $c' := c\|0$
- decrypt c' repeatedly until you get a plaintext of the form $p\|0$. Then output p

Problem: this approach does not work as FPE for really short data

Feistel constructions - example

Algorithm 1: The Encryption Algorithm of FF3

Input : Message P of domain of size $M \times N$, Key K, Tweak $T = T_L || T_R$
Output: Ciphertext C of domain size $M \times N$

$(L, R) \leftarrow P$;
for $i \leftarrow 0$ **to** 7 **do**
  if $i \mod 2 = 0$ **then**
    $L \leftarrow L \boxplus AES_K(Encode96(R)||T_R \oplus i) \mod M$;
  else
    $R \leftarrow R \boxplus AES_K(Encode96(L)||T_L \oplus i) \mod N$;
return $C \leftarrow L||R$;

(remark: the pict. from Amon et al seems to contain some minor misprints)

FF3 is one of two algorithms recommended by NIST as FPE

ATTACKS on FF3

the attacks are generally of high complexity but for small plaintext size they may be still dangerous

example: message recovery attack

- an unknown plaintext can be encrypted with chosen tweaks (important!)
- idea: characteristics and differential cryptanalysis:
  - → difference only in L: $X=(L,R)$, $X'=(L',R)$
  - → after the first round difference not changed (say $(\Delta,0)$)
  - → in the second round the output of the round function =0 with probability $1/2^{\text{length of } L}$
  - → ... so with this probability the difference remains $(\Delta,0)$
  - → final ciphertext difference $(\Delta,0)$ with a fair pbb
- known L from (L,R), other input (L',R) where L',R are unknown, goal: learn L'
- collect ciphertexts with many different tweaks:
  - → outputs (C,D) and (C',D') with difference $(\Delta,0)$ yield a candidate $L'=L\otimes\Delta$