

Security and Cryptography 2021

Mirosław Kutylowski

VI. 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 IV, 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 i th block decrypts to P' (while block P_{i-1} will become junk):

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

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

- $IV_n = \text{Enc}_K(n)$ where $K = \text{Hash}(K_0)$ 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

– $C = \text{Enc}_K(P \oplus X) \oplus X$ (\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 $F || 0^n$:

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

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

so F will be revealed

Xor-encrypt-xor (XEX)

- $X_J = \text{Enc}_K(I) \otimes \alpha^J$
- $C_J = \text{Enc}_K(P \oplus X_J) \oplus X_J$
- I is the sector number, J is the block number 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)

- $\text{DerivedKey} = \text{PBDKF2}(\text{PRF}, \text{Password}, \text{Salt}, c, \text{dkLen})$
- c is the number of iterations requested
- $\text{DerivedKey} = T_1 || T_2 || \dots || T_{\text{dklen}/\text{hlen}}$
- $T_i = F(\text{Password}, \text{Salt}, c, i)$
- $F(\text{Password}, \text{Salt}, c, i) = U_1 \otimes U_2 \otimes \dots \otimes U_c$ where \otimes stands for xor
- $U_1 = \text{PRF}(\text{Password}, \text{Salt}, i)$
- $U_j = \text{PRF}(\text{Password}, U_{j-1})$ 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:** $H_0 = \text{Hash}(p, \tau, m, t, v, y, \langle P \rangle, P, \langle S \rangle, S, \langle K \rangle, K, \langle X \rangle, X)$ where $\langle A \rangle$ denotes the length of A ,
- **expand** using a variable length hash H' :
 - initialize blocks $B[i, j]$ with p rows ($i = 0, \dots, p - 1$) 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 - compression function
- **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**
 - $B_{\text{final}} = B[0, q - 1] \oplus B[1, q - 1] \oplus \dots \oplus B[p - 1, q - 1]$
 - $\text{Tag} = H'(B_{\text{final}})$

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 $\tau > 64$:

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

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

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

...

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

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

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

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, \dots, R_{63}
 - $(Q_0, \dots, Q_7) = P(R_0, \dots, R_7)$
 - $(Q_8, \dots, Q_{15}) = P(R_8, \dots, R_{15})$
 - ...
 - $(Q_{56}, \dots, Q_{63}) = P(R_{56}, \dots, R_{63})$
 - $(Z_0, Z_8, \dots, Z_{56}) = P(Q_0, Q_8, \dots, Q_{56})$
 - $(Z_1, Z_9, \dots, Z_{57}) = P(Q_1, Q_9, \dots, Q_{57})$
 - ...
 - $(Z_7, Z_{15}, \dots, Z_{63}) = P(Q_7, Q_{15}, \dots, Q_{63})$
- finally output
$$Z \oplus R$$