bcrypt Algorithm

Introduction

The bcrypt functions as a password-hashing function based on the Blowfish cipher. This function involves random data (called salt) to use as an additional input into a one-way function that hashes data, a password or a passphrase to provide protection against pre-computed tables meant to catch the output of cryptographic hash functions, usually meant to crack password hashes and is quite adaptive to improve its resistance against brute-force search attacks.

The Blowfish Cipher

Blowfish is a symmetric-key block cipher included in many cipher suites and encryption products. A symmetric-key algorithm uses the same cryptographic keys for both encryption of plaintext and decryption of ciphertext. The keys maybe identical or there may be a simple transformation to go between the keys. A block cipher is a deterministic-algorithm operating on fixed-length groups of bits, called blocks. They are specified elementary components in the design of many cryptographic protocols and are widely used to implement the encryption of large amounts of data, including data exchange protocols.

Blowfish has a 64-bit block size and a variable key length from 32 bits up to 448 bits.[3] It is a 16-round Feistel cipher and uses large key-dependent S-boxes. In structure it resembles CAST-128, which uses fixed S-boxes. The adjacent diagram shows Blowfish's encryption routine. Each line represents 32 bits. There are five subkey-arrays: one 18-entry P-array (denoted as K in the diagram, to avoid confusion with the Plaintext) and four 256-entry S-boxes (S0, S1, S2 and S3).

Every round r consists of 4 actions:

1. XOR the left half (L) of the data with the r^{th} P-array entry

2. Use the XORed data as input for Blowfish's F-function

3. XOR the F-function's output with the right half (R) of the data

4. Swap L and R

The F-function splits the 32-bit input into four eight-bit quarters, and uses the quarters as input to the S-boxes. The S-boxes accept 8-bit input and produce 32-bit output. The outputs are added modulo 232 and XORed to produce the final 32-bit output (see image in the upper right corner).[4]

After the 16th round, undo the last swap, and XOR L with K18 and R with K17 (output whitening).

Decryption is exactly the same as encryption, except that P1, P2, ..., P18 are used in the reverse order. This is not so obvious because xor is commutative and associative. A common misconception is to use inverse order of encryption as decryption algorithm (i.e. first XORing P17 and P18 to the ciphertext block, then using the P-entries in reverse order).

The bcrypt Algorithm

The bcrypt algorithm works by applying the Blowfish algorithm 64 times on a string. In bcrypt the usual Blowfish key setup function is replaced with an expensive key setup function. The **expensive key setup** runs as follows:

Input:

1. password: array of bytes (1 .. 72 bytes) *(UTF-8 encoded password)*

2. salt: array of bytes (16 bytes) *(random salt)*

3. cost: Number (4 .. 31) $(log_2(Iterations))$

Output:

1. P: array of UInt32 *(array of 18 per-round subkeys)*

2. S_1 .. S_4: array of UInt32 *(array of four SBoxes; each SBox is 256 UInt32)*

Initialize P (Subkeys), and S (Substitution boxes) with the hex digits of pi

```
**repeat** $(2^{\text{cost}})$
P, S $\leftarrow ExpandKey(P, S, 0, password)$
$P, S \leftarrow ExpandKey(P, S, 0, salt)$
**return**$P, S$
```

The ExpandKey(state, 0, key) is the same as regular Blowfish key schedule since all XORs with the all-zero salt value are ineffectual.ExpandKey(state, 0, salt) is similar, but uses the salt as a 128-bit key.