

Blowfish Encryption Algorithm

The data transformation process for PocketBrief uses the *Blowfish Algorithm* for Encryption and Decryption, respectively. The details and working of the algorithm are given below.

Blowfish is a symmetric block cipher that can be effectively used for encryption and safeguarding of data. It takes a variable-length key, from 32 bits to 448 bits, making it ideal for securing data. *Blowfish* was designed in 1993 by **Bruce Schneier** as a fast, free alternative to existing encryption algorithms. *Blowfish* is unpatented and license-free, and is available free for all uses.

Blowfish Algorithm is a **Feistel Network**, iterating a simple encryption function 16 times. The block size is 64 bits, and the key can be any length up to 448 bits. Although there is a complex initialization phase required before any encryption can take place, the actual encryption of data is very efficient on large microprocessors.

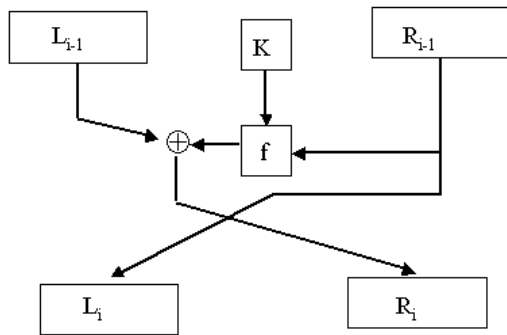
Blowfish is a variable-length key block cipher. It is suitable for applications where the key does not change often, like a communications link or an automatic file encryptor. It is significantly faster than most encryption algorithms when implemented on 32-bit microprocessors with large data caches.

Feistel Networks

A Feistel network is a general method of transforming any function (usually called an F-function) into a permutation. It was invented by Horst Feistel and has been used in many block cipher designs. The working of a Feistel Network is given below:

- Split each block into halves
- Right half becomes new left half
- New right half is the final result when the left half is XOR'd with the result of applying f to the right half and the key.
- Note that previous rounds can be derived even if the function f is not invertible.

Feistel Network



The *Blowfish* Algorithm:

- ✓ Manipulates data in large blocks
- ✓ Has a 64-bit block size.
- ✓ Has a scalable key, from 32 bits to at least 256 bits.
- ✓ Uses simple operations that are efficient on microprocessors.

e.g., exclusive-or, addition, table lookup, modular- multiplication. It does not use variable-length shifts or bit-wise permutations, or conditional jumps.

- ✓ Employs precomputable subkeys.

On large-memory systems, these subkeys can be precomputed for faster operation. Not precomputing the subkeys will result in slower operation, but it should still be possible to encrypt data without any precomputations.

- ✓ Consists of a variable number of iterations.

For applications with a small key size, the trade-off between the complexity of a brute-force attack and a differential attack make a large number of iterations superfluous. Hence, it should be possible to reduce the number of iterations with no loss of security (beyond that of the reduced key size).

- ✓ Uses subkeys that are a one-way hash of the key.

This allows the use of long passphrases for the key without compromising security.

- ✓ Has no linear structures that reduce the complexity of exhaustive search.
- ✓ Uses a design that is simple to understand. This facilitates analysis and increase the confidence in the algorithm. In practice, this means that the algorithm will be a Feistel iterated block cipher.

DESCRIPTION OF THE ALGORITHM

Blowfish is a variable-length key, 64-bit block cipher. The algorithm consists of two parts: a key-expansion part and a data- encryption part. Key expansion converts a key of at most 448 bits into several subkey arrays totaling 4168 bytes.

Data encryption occurs via a 16-round Feistel network. Each round consists of a key-dependent permutation, and a key- and data-dependent substitution. All operations are XORs and additions on 32-bit words. The only additional operations are four indexed array data lookups per round.

Subkeys

Blowfish uses a large number of subkeys. These keys must be precomputed before any data encryption or decryption.

- The P-array consists of 18 32-bit subkeys:

P1, P2, ..., P18.

- There are four 32-bit S-boxes with 256 entries each:

S1,0, S1,1, ..., S1,255;

S2,0, S2,1, ..., S2,255;

S3,0, S3,1, ..., S3,255;

S4,0, S4,1, ..., S4,255.

Encryption

Blowfish has 16 rounds.

The input is a 64-bit data element, x .

Divide x into two 32-bit halves: x_L , x_R .

Then, for $i = 1$ to 16:

$$x_L = x_L \text{ XOR } P_i$$

$$x_R = F(x_L) \text{ XOR } x_R$$

Swap x_L and x_R

After the sixteenth round, swap x_L and x_R again to undo the last swap.

Then, $x_R = x_R \text{ XOR } P_{17}$ and $x_L = x_L \text{ XOR } P_{18}$.

Finally, recombine x_L and x_R to get the ciphertext.

Decryption is exactly the same as encryption, except that P1, P2,..., P18 are used in the reverse order.

Implementations of *Blowfish* that require the fastest speeds should unroll the loop and ensure that all subkeys are stored in cache.

Generating the Subkeys

The subkeys are calculated using the *Blowfish* algorithm:

1. Initialize first the P-array and then the four S-boxes, in order, with a fixed string. This string consists of the hexadecimal digits of pi (less the initial 3): P1 = 0x243f6a88, P2 = 0x85a308d3, P3 = 0x13198a2e, P4 = 0x03707344, etc.
2. XOR P1 with the first 32 bits of the key, XOR P2 with the second 32-bits of the key, and so on for all bits of the key (possibly up to P14). Repeatedly cycle through the key bits until the entire P-array has been XORed with key bits. (For every short key, there is at least one equivalent longer key; for example, if A is a 64-bit key, then AA, AAA, etc., are equivalent keys.)
3. Encrypt the all-zero string with the *Blowfish* algorithm, using the subkeys described in steps (1) and (2).
4. Replace P1 and P2 with the output of step (3).
5. Encrypt the output of step (3) using the *Blowfish* algorithm with the modified subkeys.
6. Replace P3 and P4 with the output of step (5).
7. Continue the process, replacing all entries of the P array, and then all four S-boxes in order, with the output of the continuously changing *Blowfish* algorithm.

In total, 521 iterations are required to generate all required subkeys. Applications can store the subkeys rather than execute this derivation process multiple times.

DESIGN DECISIONS

A 64-bit block size yields a 32-bit word size, and maintains block-size compatibility with existing algorithms. *Blowfish* is easy to scale up to a 128-bit block, and down to smaller block sizes.

The fundamental operations were chosen with speed in mind. XOR, ADD, and MOV from a cache are efficient on both Intel and Motorola architectures. All subkeys fit in the cache of a 80486, 68040, Pentium, and PowerPC.

The **Feistel Network** that makes up the body of *Blowfish* is designed to be as simple as possible, while still retaining the desirable cryptographic properties of the structure.

In algorithm design, there are two basic ways to ensure that the key is long enough to ensure a particular security level. One is to carefully design the algorithm so that the

entire entropy of the key is preserved, so there is no better way to cryptanalyze the algorithm other than brute force. The other is to design the algorithm with so many key bits that attacks that reduce the effective key length by several bits are irrelevant. Since *Blowfish* is designed for large microprocessors with large amounts of memory, the latter has been chosen. But it works equally well on Handheld systems with a decent microprocessor.

The subkey generation process is designed to preserve the entire entropy of the key and to distribute that entropy uniformly throughout the subkeys. It is also designed to distribute the set of allowed subkeys randomly throughout the domain of possible subkeys. The digits of pi were chosen as the initial subkey table for two reasons: because it is a random sequence not related to the algorithm, and because it could either be stored as part of the algorithm or derived when needed. But if the initial string is non-random in any way (for example, ASCII text with the high bit of every byte a 0), this non-randomness will propagate throughout the algorithm.

In the subkey generation process, the subkeys change slightly with every pair of subkeys generated. This is primarily to protect against any attacked of the subkey generation process that exploit the fixed and known subkeys. It also reduces storage requirements. The 448 limit on the key size ensures that the every bit of every subkey depends on every bit of the key.

The key bits are repeatedly XORed with the digits of pi in the initial P-array to prevent the following potential attack: Assume that the key bits are not repeated, but instead padded with zeros to extend it to the length of the P-array. An attacker might find two keys that differ only in the 64-bit value XORed with P1 and P2 that, using the initial known subkeys, produce the same encrypted value. If so, he can find two keys that produce all the same subkeys. This is a highly tempting attack for a malicious key generator. To prevent this same type of attack, the initial plaintext value in the subkey-generation process is fixed.

The subkey-generation algorithm does not assume that the key bits are random. Even highly correlated key bits, such as an alphanumeric ASCII string with the bit of every byte set to 0, will produce random subkeys. However, to produce subkeys with the same entropy, a longer alphanumeric key is required.

The time-consuming subkey-generation process adds considerable complexity for a brute-force attack. The subkeys are too long to be stored on a massive tape, so they would have to be generated by a brute-force cracking machine as required. A total of 522 iterations of the encryption algorithm are required to test a single key, effectively adding 29 steps to any brute-force attack.

The most efficient way to break *Blowfish* is through exhaustive search of the keyspace.

References

- B. Schneier, **Applied Cryptography**,
John Wiley & Sons, New York, 1994.

- B. Schneier, **Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)**
Fast Software Encryption, Cambridge Security Workshop Proceedings (December 1993), Springer-Verlag, 1994, pp. 191-204.