```
package use_case_controller;

//Copyright (c) 2006 Damien Miller <djm@mindrot.org>

import java.io.UnsupportedEncodingException;

public class BCrypt {

    // BCrypt parameters
    private static final int GENSALT_DEFAULT_LOG2_ROUNDS = 10;
    private static final int BCRYPT_SALT_LEN = 16;

    // Blowfish parameters
    private static final int BLOWFISH_NUM_ROUNDS = 16;

    // Initial contents of key schedule
    private static final int P_orig[] = { 0x243f6a88, 0x85a308d3, 0x13198a2e, 0x03707344, 0xa4093822, 0x299f31d0,
            0x082efa98, 0xec4e6c89, 0x452821e6, 0x38d01377, 0xbe5466cf, 0x34e90c6c, 0xc0ac29b7, 0xc97c50dd, 0x3f84d5b5,
            0xb5470917, 0x9216d5d9, 0x8979fb1b };
    private static final int S_orig[] = { 0xd1310ba6, 0x98dfb5ac, 0x2ffd72db, 0xd01adfb7, 0xb8e1afed, 0x6a267e96,
            0xba7c9045, 0xf12c7f99, 0x24a19947, 0xb3916cf7, 0x0801f2e2, 0x858efc16, 0x636920d8, 0x71574e69, 0xa458fea3,
            0xf4933d7e, 0x0d95748f, 0x728eb658, 0x718bcd58, 0x82154aee, 0x7b54a41d, 0xc25a59b5, 0x9c30d539, 0x2af26013,
            0xc5d1b023, 0x286085f0, 0xca417918, 0xb8db38ef, 0x8e79dcb0, 0x603a180e, 0x6c9e0e8b, 0xb01e8a3e, 0xd71577c1,
            0xbd314b27, 0x78af2fda, 0x55605c60, 0xe65525f3, 0xaa55ab94, 0x57489862, 0x63e81440, 0x55ca396a, 0x2aab10b6,
            0xb4cc5c34, 0x1141e8ce, 0xa15486af, 0x7c72e993, 0xb3ee1411, 0x636fbc2a, 0x2ba9c55d, 0x741831f6, 0xce5c3e16,
            0x9b87931e, 0xafd6ba33, 0x6c24cf5c, 0x7a325381, 0x28958677, 0x3b8f4898, 0x6b4bb9af, 0xc4bfe81b, 0x66282193,
            0x61d809cc, 0xfb21a991, 0x487cac60, 0x5dec8032, 0xef845d5d, 0xe98575b1, 0xdc262302, 0xeb651b88, 0x23893e81,
            0xd396acc5, 0x0f6d6ff3, 0x83f44239, 0x2e0b4482, 0xa4842004, 0x69c8f04a, 0x9e1f9b5e, 0x21c66842, 0xf6e96c9a,
            0x670c9c61, 0xabd388f0, 0x6a51a0d2, 0xd8542f68, 0x960fa728, 0xab5133a3, 0x6eef0b6c, 0x137a3be4, 0xba3bf050,
            0x7efb2a98, 0xa1f1651d, 0x39af0176, 0x66ca593e, 0x82430e88, 0x8cee8619, 0x456f9fb4, 0x7d84a5c3, 0x3b8b5ebe,
            0xe06f75d8, 0x85c12073, 0x401a449f, 0x56c16aa6, 0x4ed3aa62, 0x363f7706, 0x1bfedf72, 0x429b023d, 0x37d0d724,
            0xd00a1248, 0xdb0fead3, 0x49f1c09b, 0x075372c9, 0x80991b7b, 0x25d479d8, 0xf6e8def7, 0xe3fe501a, 0xb6794c3b,
            0x976ce0bd, 0x04c006ba, 0xc1a94fb6, 0x409f60c4, 0x5e5c9ec2, 0x196a2463, 0x68fb6faf, 0x3e6c53b5, 0x1339b2eb,
            0x3b52ec6f, 0x6dfc511f, 0x9b30952c, 0xcc814544, 0xaf5ebd09, 0xbee3d004, 0xde334afd, 0x660f2807, 0x192e4bb3,
            0xc0cba857, 0x45c8740f, 0xd20b5f39, 0xb9d3fbdb, 0x5579c0bd, 0x1a60320a, 0xd6a100c6, 0x402c7279, 0x679f25fe,
            0xfb1fa3cc, 0x8ea5e9f8, 0xdb3222f8, 0x3c7516df, 0xfd616b15, 0x2f501ec8, 0xad0552ab, 0x323db5fa, 0xfd238760,
            0x53317b48, 0x3e00df82, 0x9e5c57bb, 0xca6f8ca0, 0x1a87562e, 0xdf1769db, 0xd542a8f6, 0x287effc3, 0xac6732c6,
            0x8c4f5573, 0x695b27b0, 0xbbca58c8, 0xe1ffa35d, 0xb8f011a0, 0x10fa3d98, 0xfd2183b8, 0x4afcb56c, 0x2dd1d35b,
            0x9a53e479, 0xb6f84565, 0xd28e49bc, 0x4bfb9790, 0xe1ddf2da, 0xa4cb7e33, 0x62fb1341, 0xcee4c6e8, 0xef20cada,
            0x36774c01, 0xd07e9efe, 0x2bf11fb4, 0x95dbda4d, 0xae909198, 0xeaad8e71, 0x6b93d5a0, 0xd08ed1d0, 0xafc725e0,
            0x8e3c5b2f, 0x8e7594b7, 0x8ff6e2fb, 0xf2122b64, 0x8888b812, 0x900df01c, 0x4fad5ea0, 0x688fc31c, 0xd1cff191,
            0xb3a8c1ad, 0x2f2f2218, 0xbe0e1777, 0xea752dfe, 0x8b021fa1, 0xe5a0cc0f, 0xb56f74e8, 0x18acf3d6, 0xce89e299,
            0xb4a84fe0, 0xfd13e0b7, 0x7cc43b81, 0xd2ada8d9, 0x165fa266, 0x80957705, 0x93cc7314, 0x211a1477, 0xe6ad2065,
            0x77b5fa86, 0xc75442f5, 0xfb9d35cf, 0xebcdaf0c, 0x7b3e89a0, 0xd6411bd3, 0xae1e7e49, 0x00250e2d, 0x2071b35e,
            0x226800bb, 0x57b8e0af, 0x2464369b, 0xf009b91e, 0x5563911d, 0x59dfa6aa, 0x78c14389, 0xd95a537f, 0x207d5ba2,
            0x02e5b9c5, 0x83260376, 0x6295cfa9, 0x11c81968, 0x4e734a41, 0xb3472dca, 0x7b14a94a, 0x1b510052, 0x9a532915,
            0xd60f573f, 0xbc9bc6e4, 0x2b60a476, 0x81e67400, 0x08ba6fb5, 0x571be91f, 0xf296ec6b, 0x2a0dd915, 0xb6636521,
            0xe7b9f9b6, 0xff34052e, 0xc5855664, 0x53b02d5d, 0xa99f8fa1, 0x08ba4799, 0x6e85076a, 0x4b7a70e9, 0xb5b32944,
            0xdb75092e, 0xc4192623, 0xad6ea6b0, 0x49a7df7d, 0x9cee60b8, 0x8fedb266, 0xecaa8c71, 0x699a17ff, 0x5664526c,
            0xc2b19ee1, 0x193602a5, 0x75094c29, 0xa0591340, 0xe4183a3e, 0x3f54989a, 0x5b429d65, 0x6b8fe4d6, 0x99f73fd6,
            0xa1d29c07, 0xefe830f5, 0x4d2d38e6, 0xf0255dc1, 0x4cdd2086, 0x8470eb26, 0x6382e9c6, 0x021ecc5e, 0x09686b3f,
            0x3ebaefc9, 0x3c971814, 0x6b6a70a1, 0x687f3584, 0x52a0e286, 0xb79c5305, 0xaa500737, 0x3e07841c, 0x7fdeae5c,
            0x8e7d44ec, 0x5716f2b8, 0xb03ada37, 0xf0500c0d, 0xf01c1f04, 0x0200b3ff, 0xae0cf51a, 0x3cb574b2, 0x25837a58,
            0xdc0921bd, 0xd19113f9, 0x7ca92ff6, 0x94324773, 0x22f54701, 0x3ae5e581, 0x37c2dadc, 0xc8b57634, 0x9af3dda7,
        0xa9446146, 0x0fd0030e, 0xecc8c73e, 0xa4751e41, 0xe238cd99, 0x3bea0e2f, 0x3280bba1, 0x183eb331, 0x4e548b38,
        0x4f6db908, 0x6f420d03, 0xf60a04bf, 0x2cb81290, 0x24977c79, 0x5679b072, 0xbcaf89af, 0xde9a771f, 0xd9930810,
        0xb38bae12, 0xdccf3f2e, 0x5512721f, 0x2e6b7124, 0x501adde6, 0x9f84cd87, 0x7a584718, 0x7408da17, 0xbc9f9abc,
        0xe94b7d8c, 0xec7aec3a, 0xdb851dfa, 0x63094366, 0xc464c3d2, 0xef1c1847, 0x3215d908, 0xdd433b37, 0x24c2ba16,
        0x12a14d43, 0x2a65c451, 0x50940002, 0x133ae4dd, 0x71dff89e, 0x10314e55, 0x81ac77d6, 0x5f11199b, 0x043556f1,
        0xd7a3c76b, 0x3c11183b, 0x5924a509, 0xf28fe6ed, 0x97f1fbfa, 0x9ebabf2c, 0x1e153c6e, 0x86e34570, 0xeae96fb1,
        0x860e5e0a, 0x5a3e2ab3, 0x771fe71c, 0x4e3d06fa, 0x2965dcb9, 0x99e71d0f, 0x803e89d6, 0x5266c825, 0x2e4cc978,
        0x9c10b36a, 0xc6150eba, 0x94e2ea78, 0xa5fc3c53, 0x1e0a2df4, 0xf2f74ea7, 0x361d2b3d, 0x1939260f, 0x19c27960,
        0x5223a708, 0xf71312b6, 0xebadfe6e, 0xeac31f66, 0xe3bc4595, 0xa67bc883, 0xb17f37d1, 0x018cff28, 0xc332ddef,
        0xbe6c5aa5, 0x65582185, 0x68ab9802, 0xeecea50f, 0xdb2f953b, 0x2aef7dad, 0x5b6e2f84, 0x1521b628, 0x29076170,
        0xecdd4775, 0x619f1510, 0x13cca830, 0xeb61bd96, 0x0334fe1e, 0xaa0363cf, 0xb5735c90, 0x4c70a239, 0xd59e9e0b,
        0xcbaade14, 0xeecc86bc, 0x60622ca7, 0x9cab5cab, 0xb2f3846e, 0x648b1eaf, 0x19bdf0ca, 0xa02369b9, 0x655abb50,
        0x40685a32, 0x3c2ab4b3, 0x319ee9d5, 0xc021b8f7, 0x9b540b19, 0x875fa099, 0x95f7997e, 0x623d7da8, 0xf837889a,
        0x97e32d77, 0x11ed935f, 0x16681281, 0x0e358829, 0xc7e61fd6, 0x96dedfa1, 0x7858ba99, 0x57f584a5, 0x1b227263,
        0x9b83c3ff, 0x1ac24696, 0xcdb30aeb, 0x532e3054, 0x8fd948e4, 0x6dbc3128, 0x58ebf2ef, 0x34c6ffea, 0xfe28ed61,
        0xee7c3c73, 0x5d4a14d9, 0xe864b7e3, 0x42105d14, 0x203e13e0, 0x45eee2b6, 0xa3aaabea, 0xdb6c4f15, 0xfacb4fd0,
        0xc742f442, 0xef6abbb5, 0x654f3b1d, 0x41cd2105, 0xd81e799e, 0x86854dc7, 0xe44b476a, 0x3d816250, 0xcf62a1f2,
        0x5b8d2646, 0xfc8883a0, 0xc1c7b6a3, 0x7f1524c3, 0x69cb7492, 0x47848a0b, 0x5692b285, 0x095bbf00, 0xad19489d,
        0x1462b174, 0x23820e00, 0x58428d2a, 0x0c55f5ea, 0x1dadf43e, 0x233f7061, 0x3372f092, 0x8d937e41, 0xd65fecf1,
        0x6c223bdb, 0x7cde3759, 0xcbee7460, 0x4085f2a7, 0xce77326e, 0xa6078084, 0x19f8509e, 0xe8efd855, 0x61d99735,
        0xa969a7aa, 0xc50c06c2, 0x5a04abfc, 0x800bcadc, 0x9e447a2e, 0xc3453484, 0xfdd56705, 0x0e1e9ec9, 0xdb73dbd3,
        0x105588cd, 0x675fda79, 0xe3674340, 0xc5c43465, 0x713e38d8, 0x3d28f89e, 0xf16dff20, 0x153e21e7, 0x8fb03d4a,
        0xe6e39f2b, 0xdb83adf7, 0xe93d5a68, 0x948140f7, 0xf64c261c, 0x94692934, 0x411520f7, 0x7602d4f7, 0xbcf46b2e,
        0xd4a20068, 0xd4082471, 0x3320f46a, 0x43b7d4b7, 0x500061af, 0x1e39f62e, 0x97244546, 0x14214f74, 0xbf8b8840,
        0x4d95fc1d, 0x96b591af, 0x70f4ddd3, 0x66a02f45, 0xbfbc09ec, 0x03bd9785, 0x7fac6dd0, 0x31cb8504, 0x96eb27b3,
        0x55fd3941, 0xda2547e6, 0xabca0a9a, 0x28507825, 0x530429f4, 0x0a2c86da, 0xe9b66dfb, 0x68dc1462, 0xd7486900,
        0x680ec0a4, 0x27a18dee, 0x4f3ffea2, 0xe887ad8c, 0xb58ce006, 0x7af4d6b6, 0xaace1e7c, 0xd3375fec, 0xce78a399,
        0x406b2a42, 0x20fe9e35, 0xd9f385b9, 0xee39d7ab, 0x3b124e8b, 0x1dc9faf7, 0x4b6d1856, 0x26a36631, 0xeae397b2,
        0x3a6efa74, 0xdd5b4332, 0x6841e7f7, 0xca7820fb, 0xfb0af54e, 0xd8feb397, 0x454056ac, 0xba489527, 0x55533a3a,
        0x20838d87, 0xfe6ba9b7, 0xd096954b, 0x55a867bc, 0xa1159a58, 0xcca92963, 0x99e1db33, 0xa62a4a56, 0x3f3125f9,
        0x5ef47e1c, 0x9029317c, 0xfdf8e802, 0x04272f70, 0x80bb155c, 0x05282ce3, 0x95c11548, 0xe4c66d22, 0x48c1133f,
        0xc70f86dc, 0x07f9c9ee, 0x41041f0f, 0x404779a4, 0x5d886e17, 0x325f51eb, 0xd59bc0d1, 0xf2bcc18f, 0x41113564,
        0x257b7834, 0x602a9c60, 0xdff8e8a3, 0x1f636c1b, 0x0e12b4c2, 0x02e1329e, 0xaf664fd1, 0xcad18115, 0x6b2395e0,
        0x333e92e1, 0x3b240b62, 0xeebeb922, 0x85b2a20e, 0xe6ba0d99, 0xde720c8c, 0x2da2f728, 0xd0127845, 0x95b794fd,
        0x647d0862, 0xe7ccf5f0, 0x5449a36f, 0x877d48fa, 0xc39dfd27, 0xf33e8d1e, 0x0a476341, 0x992eff74, 0x3a6f6eab,
        0xf4f8fd37, 0xa812dc60, 0xa1ebddf8, 0x991be14c, 0xdb6e6b0d, 0xc67b5510, 0x6d672c37, 0x2765d43b, 0xdcd0e804,
        0xf1290dc7, 0xcc00ffa3, 0xb5390f92, 0x690fed0b, 0x667b9ffb, 0xcedb7d9c, 0xa091cf0b, 0xd9155ea3, 0xbb132f88,
        0x515bad24, 0x7b9479bf, 0x763bd6eb, 0x37392eb3, 0xcc115979, 0x8026e297, 0xf42e312d, 0x6842ada7, 0xc66a2b3b,
        0x12754ccc, 0x782ef11c, 0x6a124237, 0xb79251e7, 0x06a1bbe6, 0x4bfb6350, 0x1a6b1018, 0x11caedfa, 0x3d25bdd8,
        0xe2e1c3c9, 0x44421659, 0x0a121386, 0xd90cec6e, 0xd5abea2a, 0x64af674e, 0xda86a85f, 0xbebfe988, 0x64e4c3fe,
        0x9dbc8057, 0xf0f7c086, 0x60787bf8, 0x6003604d, 0xd1fd8346, 0xf6381fb0, 0x7745ae04, 0xd736fccc, 0x83426b33,
        0xf01eab71, 0xb0804187, 0x3c005e5f, 0x77a057be, 0xbde8ae24, 0x55464299, 0xbf582e61, 0x4e58f48f, 0xf2ddfda2,
        0xf474ef38, 0x8789bdc2, 0x5366f9c3, 0xc8b38e74, 0xb475f255, 0x46fcd9b9, 0x7aeb2661, 0x8b1ddf84, 0x846a0e79,
        0x915f95e2, 0x466e598e, 0x20b45770, 0x8cd55591, 0xc902de4c, 0xb90bace1, 0xbb8205d0, 0x11a86248, 0x7574a99e,
        0xb77f19b6, 0xe0a9dc09, 0x662d09a1, 0xc4324633, 0xe85a1f02, 0x09f0be8c, 0x4a99a025, 0x1d6efe10, 0x1ab93d1d,
        0x0ba5a4df, 0xa186f20f, 0x2868f169, 0xdcb7da83, 0x573906fe, 0xa1e2ce9b, 0x4fcd7f52, 0x50115e01, 0xa70683fa,
        0xa002b5c4, 0x0de6d027, 0x9af88c27, 0x773f8641, 0xc3604c06, 0x61a806b5, 0xf0177a28, 0xc0f586e0, 0x006058aa,
        0x30dc7d62, 0x11e69ed7, 0x2338ea63, 0x53c2dd94, 0xc2c21634, 0xbbcbee56, 0x90bcb6de, 0xebfc7da1, 0xce591d76,
        0x6f05e409, 0x4b7c0188, 0x39720a3d, 0x7c927c24, 0x86e3725f, 0x724d9db9, 0x1ac15bb4, 0xd39eb8fc, 0xed545578,
        0x08fca5b5, 0xd83d7cd3, 0x4dad0fc4, 0x1e50ef5e, 0xb161e6f8, 0xa28514d9, 0x6c51133c, 0x6fd5c7e7, 0x56e14ec4,
        0x362abfce, 0xddc6c837, 0xd79a3234, 0x92638212, 0x670efa8e, 0x406000e0, 0x3a39ce37, 0xd3faf5cf, 0xabc27737,
        0x5ac52d1b, 0x5cb0679e, 0x4fa33742, 0xd3822740, 0x99bc9bbe, 0xd5118e9d, 0xbf0f7315, 0xd62d1c7e, 0xc700c47b,
        0xb78c1b6b, 0x21a19045, 0xb26eb1be, 0x6a366eb4, 0x5748ab2f, 0xbc946e79, 0xc6a376d2, 0x6549c2c8, 0x530ff8ee,
        0x468dde7d, 0xd5730a1d, 0x4cd04dc6, 0x2939bbdb, 0xa9ba4650, 0xac9526e8, 0xbe5ee304, 0xa1fad5f0, 0x6a2d519a,
```

```java
            0x63ef8ce2, 0x9a86ee22, 0xc089c2b8, 0x43242ef6, 0xa51e03aa, 0x9cf2d0a4, 0x83c061ba, 0x9be96a4d, 0x8fe51550,
            0xba645bd6, 0x2826a2f9, 0xa73a3ae1, 0x4ba99586, 0xef5562e9, 0xc72fefd3, 0xf752f7da, 0x3f046f69, 0x77fa0a59,
            0x80e4a915, 0x87b08601, 0x9b09e6ad, 0x3b3ee593, 0xe990fd5a, 0x9e34d797, 0x2cf0b7d9, 0x022b8b51, 0x96d5ac3a,
            0x017da67d, 0xd1cf3ed6, 0x7c7d2d28, 0x1f9f25cf, 0xadf2b89b, 0x5ad6b472, 0x5a88f54c, 0xe029ac71, 0xe019a5e6,
            0x47b0acfd, 0xed93fa9b, 0xe8d3c48d, 0x283b57cc, 0xf8d56629, 0x79132e28, 0x785f0191, 0xed756055, 0xf7960e44,
            0xe3d35e8c, 0x15056dd4, 0x88f46dba, 0x03a16125, 0x0564f0bd, 0xc3eb9e15, 0x3c9057a2, 0x97271aec, 0xa93a072a,
            0x1b3f6d9b, 0x1e6321f5, 0xf59c66fb, 0x26dcf319, 0x7533d928, 0xb155fdf5, 0x03563482, 0x8aba3cbb, 0x28517711,
            0xc20ad9f8, 0xabcc5167, 0xccad925f, 0x4de81751, 0x3830dc8e, 0x379d5862, 0x9320f991, 0xea7a90c2, 0xfb3e7bce,
            0x5121ce64, 0x774fbe32, 0xa8b6e37e, 0xc3293d46, 0x48de5369, 0x6413e680, 0xa2ae0810, 0xdd6db224, 0x69852dfd,
            0x09072166, 0xb39a460a, 0x6445c0dd, 0x586cdecf, 0x1c20c8ae, 0x5bbef7dd, 0x1b588d40, 0xccd2017f, 0x6bb4e3bb,
            0xdda26a7e, 0x3a59ff45, 0x3e350a44, 0xbcb4cdd5, 0x72eacea8, 0xfa6484bb, 0x8d6612ae, 0xbf3c6f47, 0xd29be463,
            0x542f5d9e, 0xaec2771b, 0xf64e6370, 0x740e0d8d, 0xe75b1357, 0xf8721671, 0xaf537d5d, 0x4040cb08, 0x4eb4e2cc,
            0x34d2466a, 0x0115af84, 0xe1b00428, 0x95983a1d, 0x06b89fb4, 0xce6ea048, 0x6f3f3b82, 0x3520ab82, 0x011a1d4b,
            0x277227f8, 0x611560b1, 0xe7933fdc, 0xbb3a792b, 0x344525bd, 0xa08839e1, 0x51ce794b, 0x2f32c9b7, 0xa01fbac9,
            0xe01cc87e, 0xbcc7d1f6, 0xcf0111c3, 0xa1e8aac7, 0x1a908749, 0xd44fbd9a, 0xd0dadecb, 0xd50ada38, 0x0339c32a,
            0xc6913667, 0x8df9317c, 0xe0b12b4f, 0xf79e59b7, 0x43f5bb3a, 0xf2d519ff, 0x27d9459c, 0xbf97222c, 0x15e6fc2a,
            0x0f91fc71, 0x9b941525, 0xfae59361, 0xceb69ceb, 0xc2a86459, 0x12baa8d1, 0xb6c1075e, 0xe3056a0c, 0x10d25065,
            0xcb03a442, 0xe0ec6e0e, 0x1698db3b, 0x4c98a0be, 0x3278e964, 0x9f1f9532, 0xe0d392df, 0xd3a0342b, 0x8971f21e,
            0x1b0a7441, 0x4ba3348c, 0xc5be7120, 0xc37632d8, 0xdf359f8d, 0x9b992f2e, 0xe60b6f47, 0x0fe3f11d, 0xe54cda54,
            0x1edad891, 0xce6279cf, 0xcd3e7e6f, 0x1618b166, 0xfd2c1d05, 0x848fd2c5, 0xf6fb2299, 0xf523f357, 0xa6327623,
            0x93a83531, 0x56cccd02, 0xacf08162, 0x5a75ebb5, 0x6e163697, 0x88d273cc, 0xde966292, 0x81b949d0, 0x4c50901b,
            0x71c65614, 0xe6c6c7bd, 0x327a140a, 0x45e1d006, 0xc3f27b9a, 0xc9aa53fd, 0x62a80f00, 0xbb25bfe2, 0x35bdd2f6,
            0x71126905, 0xb2040222, 0xb6cbcf7c, 0xcd769c2b, 0x53113ec0, 0x1640e3d3, 0x38abbd60, 0x2547adf0, 0xba38209c,
            0xf746ce76, 0x77afa1c5, 0x20756060, 0x85cbfe4e, 0x8ae88dd8, 0x7aaaf9b0, 0x4cf9aa7e, 0x1948c25c, 0x02fb8a8c,
            0x01c36ae4, 0xd6ebe1f9, 0x90d4f869, 0xa65cdea0, 0x3f09252d, 0xc208e69f, 0xb74e6132, 0xce77e25b, 0x578fdfe3,
            0x3ac372e6 };

    // bcrypt IV: "OrpheanBeholderScryDoubt"
    static private final int bf_crypt_ciphertext[] = { 0x4f727068, 0x65616e42, 0x65686f6c, 0x64657253, 0x63727944,
            0x6f756274 };

    // Table for Base64 encoding
    static private final char base64_code[] = { '.', '/', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',
            'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', 'a', 'b', 'c', 'd', 'e', 'f', 'g',
            'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '0', '1',
            '2', '3', '4', '5', '6', '7', '8', '9' };

    // Table for Base64 decoding
    static private final byte index_64[] = { -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
            -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
            0, 1, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, -1, -1, -1, -1, -1, -1, -1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
            12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, -1, -1, -1, -1, -1, -1, 28, 29, 30, 31, 32,
            33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, -1, -1, -1, -1, -1 };

    // Expanded Blowfish key
    private int P[];
    private int S[];

    /**
     * Encode a byte array using bcrypt's slightly-modified base64 encoding scheme.
     * Note that this is *not* compatible with the standard MIME-base64 encoding.
     *
     * @param d   the byte array to encode
     * @param len the number of bytes to encode
     * @return base64-encoded string
     * @exception IllegalArgumentException if the length is invalid
     */
    private static String encode_base64(byte d[], int len) throws IllegalArgumentException {
        int off = 0;
        StringBuffer rs = new StringBuffer();
        int c1, c2;

        if (len <= 0 || len > d.length)
            throw new IllegalArgumentException("Invalid len");

        while (off < len) {
            c1 = d[off++] & 0xff;
            rs.append(base64_code[(c1 >> 2) & 0x3f]);
            c1 = (c1 & 0x03) << 4;
            if (off >= len) {
                rs.append(base64_code[c1 & 0x3f]);
                break;
            }
            c2 = d[off++] & 0xff;
            c1 |= (c2 >> 4) & 0x0f;
            rs.append(base64_code[c1 & 0x3f]);
            c1 = (c2 & 0x0f) << 2;
            if (off >= len) {
                rs.append(base64_code[c1 & 0x3f]);
                break;
            }
            c2 = d[off++] & 0xff;
            c1 |= (c2 >> 6) & 0x03;
            rs.append(base64_code[c1 & 0x3f]);
            rs.append(base64_code[c2 & 0x3f]);
        }
        return rs.toString();
    }

    /**
     * Look up the 3 bits base64-encoded by the specified character, range-checking
     * againt conversion table
     *
     * @param x the base64-encoded value
     * @return the decoded value of x
     */
    private static byte char64(char x) {
        if ((int) x < 0 || (int) x > index_64.length)
            return -1;
        return index_64[(int) x];
    }

    /**
     * Decode a string encoded using bcrypt's base64 scheme to a byte array. Note
     * that this is *not* compatible with the standard MIME-base64 encoding.
     *
     * @param s       the string to decode
     * @param maxolen the maximum number of bytes to decode
     * @return an array containing the decoded bytes
     * @throws IllegalArgumentException if maxolen is invalid
     */
    private static byte[] decode_base64(String s, int maxolen) throws IllegalArgumentException {
        StringBuffer rs = new StringBuffer();
        int off = 0, slen = s.length(), olen = 0;
        byte ret[];
        byte c1, c2, c3, c4, o;

        if (maxolen <= 0)
            throw new IllegalArgumentException("Invalid maxolen");

        while (off < slen - 1 && olen < maxolen) {
            c1 = char64(s.charAt(off++));
            c2 = char64(s.charAt(off++));
            if (c1 == -1 || c2 == -1)
                break;
            o = (byte) (c1 << 2);
            o |= (c2 & 0x30) >> 4;
            rs.append((char) o);
            if (++olen >= maxolen || off >= slen)
                break;
            c3 = char64(s.charAt(off++));
            if (c3 == -1)
                break;
            o = (byte) ((c2 & 0x0f) << 4);
            o |= (c3 & 0x3c) >> 2;
            rs.append((char) o);
            if (++olen >= maxolen || off >= slen)
                break;
            c4 = char64(s.charAt(off++));
            o = (byte) ((c3 & 0x03) << 6);
            o |= c4;
            rs.append((char) o);
            ++olen;
        }

        ret = new byte[olen];
        for (off = 0; off < olen; off++)
            ret[off] = (byte) rs.charAt(off);
        return ret;
    }

    /**
     * Blowfish encipher a single 64-bit block encoded as two 32-bit halves
     *
     * @param lr  an array containing the two 32-bit half blocks
     * @param off the position in the array of the blocks
     */
    private final void encipher(int lr[], int off) {
        int i, n, l = lr[off], r = lr[off + 1];

        l ^= P[0];
        for (i = 0; i <= BLOWFISH_NUM_ROUNDS - 2;) {
            // Feistel substitution on left word
            n = S[(l >> 24) & 0xff];
            n += S[0x100 | ((l >> 16) & 0xff)];
            n ^= S[0x200 | ((l >> 8) & 0xff)];
            n += S[0x300 | (l & 0xff)];
            r ^= n ^ P[++i];

            // Feistel substitution on right word
            n = S[(r >> 24) & 0xff];
            n += S[0x100 | ((r >> 16) & 0xff)];
            n ^= S[0x200 | ((r >> 8) & 0xff)];
            n += S[0x300 | (r & 0xff)];
            l ^= n ^ P[++i];
        }
        lr[off] = r ^ P[BLOWFISH_NUM_ROUNDS + 1];
        lr[off + 1] = l;
    }

    /**
     * Cycically extract a word of key material
     *
     * @param data the string to extract the data from
     * @param offp a "pointer" (as a one-entry array) to the current offset into
     *             data
     * @return the next word of material from data
     */
    private static int streamtoword(byte data[], int offp[]) {
        int i;
        int word = 0;
        int off = offp[0];

        for (i = 0; i < 4; i++) {
            word = (word << 8) | (data[off] & 0xff);
            off = (off + 1) % data.length;
        }

        offp[0] = off;
        return word;
    }

    /**
     * Initialise the Blowfish key schedule
     */
    private void init_key() {
        P = (int[]) P_orig.clone();
        S = (int[]) S_orig.clone();
    }

    /**
     * Key the Blowfish cipher
     *
     * @param key an array containing the key
     */
    private void key(byte key[]) {
        int i;
        int koffp[] = { 0 };
        int lr[] = { 0, 0 };
        int plen = P.length, slen = S.length;

        for (i = 0; i < plen; i++)
            P[i] = P[i] ^ streamtoword(key, koffp);

        for (i = 0; i < plen; i += 2) {
            encipher(lr, 0);
            P[i] = lr[0];
            P[i + 1] = lr[1];
        }

        for (i = 0; i < slen; i += 2) {
            encipher(lr, 0);
            S[i] = lr[0];
            S[i + 1] = lr[1];
        }
    }

    /**
     * Perform the "enhanced key schedule" step described by Provos and Mazieres in
     * "A Future-Adaptable Password Scheme"
     * http://www.openbsd.org/papers/bcrypt-paper.ps
     *
     * @param data salt information
     * @param key  password information
     */
    private void ekskey(byte data[], byte key[]) {
        int i;
        int koffp[] = { 0 }, doffp[] = { 0 };
        int lr[] = { 0, 0 };
        int plen = P.length, slen = S.length;

        for (i = 0; i < plen; i++)
            P[i] = P[i] ^ streamtoword(key, koffp);

        for (i = 0; i < plen; i += 2) {
            lr[0] ^= streamtoword(data, doffp);
            lr[1] ^= streamtoword(data, doffp);
            encipher(lr, 0);
            P[i] = lr[0];
            P[i + 1] = lr[1];
        }

        for (i = 0; i < slen; i += 2) {
            lr[0] ^= streamtoword(data, doffp);
            lr[1] ^= streamtoword(data, doffp);
            encipher(lr, 0);
            S[i] = lr[0];
            S[i + 1] = lr[1];
        }
    }
```

```java

    /**
     * Perform the central password hashing step in the bcrypt scheme
     *
     * @param password   the password to hash
     * @param salt       the binary salt to hash with the password
     * @param log_rounds the binary logarithm of the number of rounds of hashing to
     *                   apply
     * @return an array containing the binary hashed password
     */
    private byte[] crypt_raw(byte password[], byte salt[], int log_rounds) {
        int rounds, i, j;
        int cdata[] = (int[]) bf_crypt_ciphertext.clone();
        int clen = cdata.length;
        byte ret[];

        if (log_rounds < 4 || log_rounds > 31)
            throw new IllegalArgumentException("Bad number of rounds");
        rounds = 1 << log_rounds;
        if (salt.length != BCRYPT_SALT_LEN)
            throw new IllegalArgumentException("Bad salt length");

        init_key();
        ekskey(salt, password);
        for (i = 0; i < rounds; i++) {
            key(password);
            key(salt);
        }

        for (i = 0; i < 64; i++) {
            for (j = 0; j < (clen >> 1); j++)
                encipher(cdata, j << 1);
        }

        ret = new byte[clen * 4];
        for (i = 0, j = 0; i < clen; i++) {
            ret[j++] = (byte) ((cdata[i] >> 24) & 0xff);
            ret[j++] = (byte) ((cdata[i] >> 16) & 0xff);
            ret[j++] = (byte) ((cdata[i] >> 8) & 0xff);
            ret[j++] = (byte) (cdata[i] & 0xff);
        }
        return ret;
    }

    /**
     * Hash a password using the OpenBSD bcrypt scheme
     *
     * @param password the password to hash
     * @param salt     the salt to hash with (perhaps generated using
     *                 BCrypt.gensalt)
     * @return the hashed password
     */
    public static String hashpw(String password, String salt) {
        BCrypt B;
        String real_salt;
        byte passwordb[], saltb[], hashed[];
        char minor = (char) 0;
        int rounds, off = 0;
        StringBuffer rs = new StringBuffer();

        if (salt.charAt(0) != '$' || salt.charAt(1) != '2')
            throw new IllegalArgumentException("Invalid salt version");
        if (salt.charAt(2) == '$')
            off = 3;
        else {
            minor = salt.charAt(2);
            if (minor != 'a' || salt.charAt(3) != '$')
                throw new IllegalArgumentException("Invalid salt revision");
            off = 4;
        }

        // Extract number of rounds
        if (salt.charAt(off + 2) > '$')
            throw new IllegalArgumentException("Missing salt rounds");
        rounds = Integer.parseInt(salt.substring(off, off + 2));

        real_salt = salt.substring(off + 3, off + 25);
        try {
            passwordb = (password + (minor >= 'a' ? "\000" : "")).getBytes("UTF-8");
        } catch (UnsupportedEncodingException uee) {
            throw new AssertionError("UTF-8 is not supported");
        }

        saltb = decode_base64(real_salt, BCRYPT_SALT_LEN);

        B = new BCrypt();
        hashed = B.crypt_raw(passwordb, saltb, rounds);

        rs.append("$2");
        if (minor >= 'a')
            rs.append(minor);
        rs.append("$");
        if (rounds < 10)
            rs.append("0");
        rs.append(Integer.toString(rounds));
        rs.append("$");
        rs.append(encode_base64(saltb, saltb.length));
        rs.append(encode_base64(hashed, bf_crypt_ciphertext.length * 4 - 1));
        return rs.toString();
    }

    /**
     * Generate a salt for use with the BCrypt.hashpw() method
     *
     * @param log_rounds the log2 of the number of rounds of hashing to apply - the
     *                   work factor therefore increases as 2**log_rounds.
     * @param random     an instance of SecureRandom to use
     * @return an encoded salt value
     */
    public static String gensalt(int log_rounds, SecureRandom random) {
        StringBuffer rs = new StringBuffer();
        byte rnd[] = new byte[BCRYPT_SALT_LEN];

        random.nextBytes(rnd);

        rs.append("$2a$");
        if (log_rounds < 10)
            rs.append("0");
        rs.append(Integer.toString(log_rounds));
        rs.append("$");
        rs.append(encode_base64(rnd, rnd.length));
        return rs.toString();
    }

    /**
     * Generate a salt for use with the BCrypt.hashpw() method
     *
     * @param log_rounds the log2 of the number of rounds of hashing to apply - the
     *                   work factor therefore increases as 2**log_rounds.
     * @return an encoded salt value
     */
    public static String gensalt(int log_rounds) {
        return gensalt(log_rounds, new SecureRandom());
    }

    /**
     * Generate a salt for use with the BCrypt.hashpw() method, selecting a
     * reasonable default for the number of hashing rounds to apply
     *
     * @return an encoded salt value
     */
    public static String gensalt() {
        return gensalt(GENSALT_DEFAULT_LOG2_ROUNDS);
    }

    /**
     * Check that a plaintext password matches a previously hashed one
     *
     * @param plaintext the plaintext password to verify
     * @param hashed    the previously-hashed password
     * @return true if the passwords match, false otherwise
     */
    public static boolean checkpw(String plaintext, String hashed) {
        return (hashed.compareTo(hashpw(plaintext, hashed)) == 0);
    }

} //end class BCrypt
```