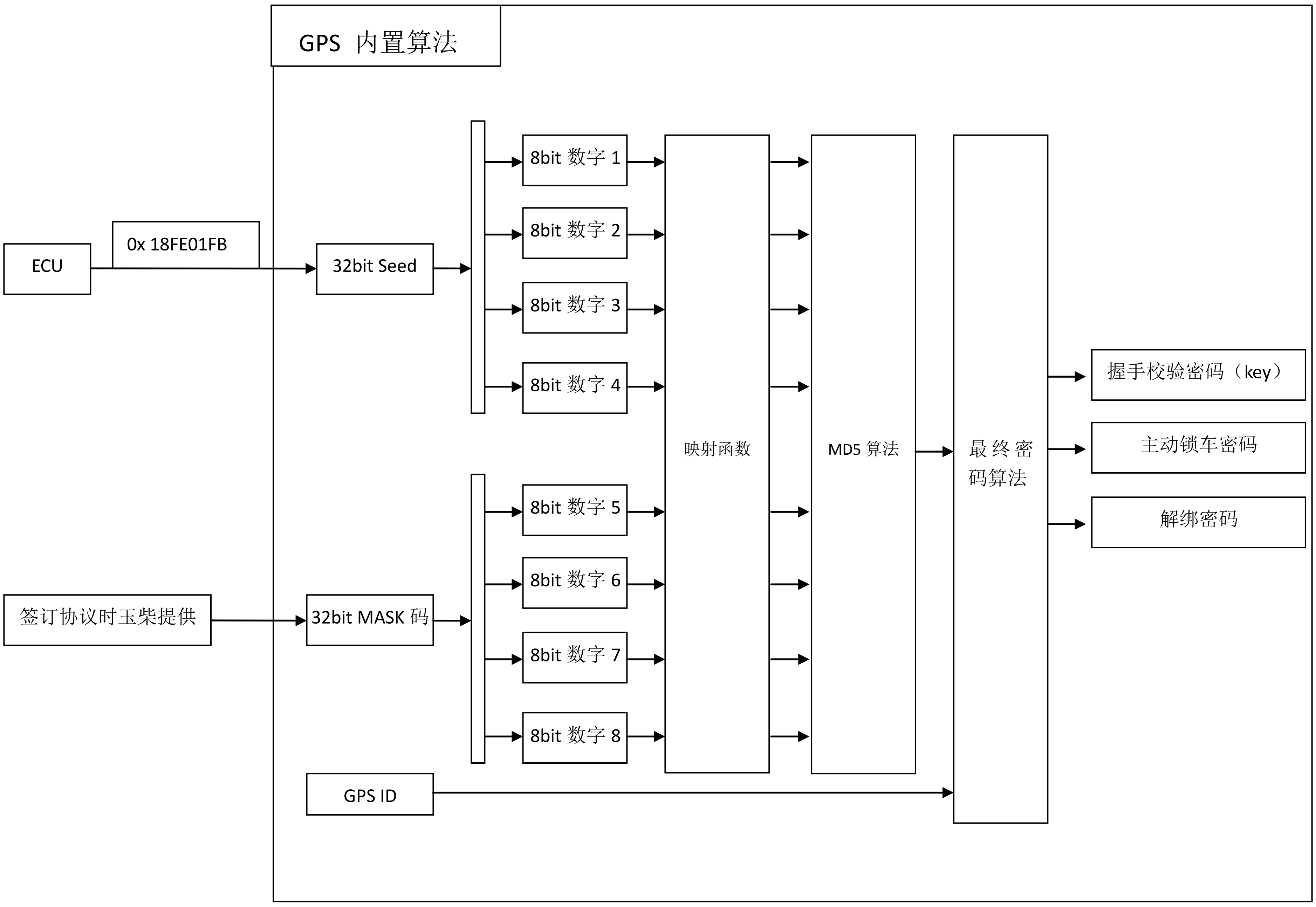
.密码算法

下图是从 seed 生成 key 的算法流程：



1. 32bit seed →8bit 数字 1，8bit 数字 2，8bit 数字 3，8bit 数字 4

	32bit seed																															
位置	0	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	28	29	30	31
	8bit 数字 1								8bit 数字 2								8bit 数字 3								8bit 数字 4							

2. 32bit MASK 码 →8bit 数字 5，8bit 数字 6，8bit 数字 7，8bit 数字 8（MASK 码属于玉柴分配给主机厂的管理代码）

	32bit MASK 码																															
位置	0	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	28	29	30	31
	8bit 数字 5								8bit 数字 6								8bit 数字 7								8bit 数字 8							

3. 映射函数原型：

```
void get_rand_str(unsigned char rand[],unsigned char new[],int number)
{
        char str[64] = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
        str[62]=0;
        str[63]=0;
        int pointer;
        for(pointer=0;pointer< number;pointer++)
            {
                for(;rand[pointer]>=62;)
                    {
                        rand[pointer]=rand[pointer]-62;
                    }
                new[pointer]=str[rand[pointer]];
            }
}
```

原理：每个 8bit 数字，调用一次映射函数，然后可以对应一个 str 数组的字符。举例：10 对应字符”A”，63 对应”1”，140 对应’’G”。

4. MD5 算法

分为 header 和 main

Header 部分：

```
/* POINTER defines a generic pointer type */
typedef unsigned char * POINTER;

/* MD5 context. */
typedef struct {
 uint32 state[4];                                   /* state (ABCD) */
 uint32 count[2];         /* number of bits, modulo 2^64 (lsb first) */
 unsigned char buffer[64];                          /* input buffer */
} MD5_CTX;

void MD5Init (MD5_CTX *context);
void MD5Update (MD5_CTX *context, unsigned char *input, unsigned int inputLen);
void MD5Final (unsigned char digest[16], MD5_CTX *context);
```

main 部分：

```
/* Constants for MD5Transform routine.*/
#define S11 7
#define S12 12
#define S13 17
#define S14 22
#define S21 5
#define S22 9
#define S23 14
#define S24 20
#define S31 4
#define S32 11
#define S33 16
#define S34 23
#define S41 6
#define S42 10
#define S43 15
#define S44 21
```

```
static void MD5_memcpy (POINTER output, POINTER input, unsigned int len);
static void MD5Transform (uint32 state[4], unsigned char block[64]);
static void Encode (unsigned char *output, uint32 *input, unsigned int len);
static void MD5_memset (POINTER output, int value, unsigned int len);
static void Decode (uint32 *output, unsigned char *input, unsigned int len);

static unsigned char PADDING[64] = {
        0x80, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};

/* F, G, H and I are basic MD5 functions.
*/
#define F(x, y, z) (((x) & (y)) | ((~x) & (z)))
#define G(x, y, z) (((x) & (z)) | ((y) & (~z)))
#define H(x, y, z) ((x) ^ (y) ^ (z))
#define I(x, y, z) ((y) ^ ((x) | (~z)))

/* ROTATE_LEFT rotates x left n bits.
*/
#define ROTATE_LEFT(x, n) (((x) << (n)) | ((x) >> (32-(n))))

/* FF, GG, HH, and II transformations for rounds 1, 2, 3, and 4.
Rotation is separate from addition to prevent recomputation.
*/
#define FF(a, b, c, d, x, s, ac) { \
        (a) += F ((b), (c), (d)) + (x) + (uint32)(ac); \
        (a) = ROTATE_LEFT ((a), (s)); \
        (a) += (b); \
        }
#define GG(a, b, c, d, x, s, ac) { \
        (a) += G ((b), (c), (d)) + (x) + (uint32)(ac); \
        (a) = ROTATE_LEFT ((a), (s)); \
        (a) += (b); \
        }
#define HH(a, b, c, d, x, s, ac) { \
        (a) += H ((b), (c), (d)) + (x) + (uint32)(ac); \
        (a) = ROTATE_LEFT ((a), (s)); \
        (a) += (b); \
        }
#define II(a, b, c, d, x, s, ac) { \
        (a) += I ((b), (c), (d)) + (x) + (uint32)(ac); \
        (a) = ROTATE_LEFT ((a), (s)); \
        (a) += (b); \
        }

/* MD5 initialization. Begins an MD5 operation, writing a new context.
  */
void MD5Init (MD5_CTX *context)                                        /* context */
{
        context->count[0] = context->count[1] = 0;
        /* Load magic initialization constants.
        */
        context->state[0] = 0x67452301;
        context->state[1] = 0xefcdab89;
        context->state[2] = 0x98badcfe;
        context->state[3] = 0x10325476;
}

/* MD5 block update operation. Continues an MD5 message-digest
    operation, processing another message block, and updating the
    context.
  */
void MD5Update (MD5_CTX *context, unsigned char *input, unsigned int inputLen)

{
        unsigned int i, index, partLen;

        /* Compute number of bytes mod 64 */
        index = (unsigned int)((context->count[0] >> 3) & 0x3F);

        /* Update number of bits */
        if ((context->count[0] += ((uint32)inputLen << 3))
                < ((uint32)inputLen << 3))
                context->count[1]++;
        context->count[1] += ((uint32)inputLen >> 29);

        partLen = 64 - index;

        /* Transform as many times as possible.
        */
        if (inputLen >= partLen) {
                MD5_memcpy((POINTER)&context->buffer[index], (POINTER)input, partLen);
                MD5Transform (context->state, context->buffer);

                for (i = partLen; i + 63 < inputLen; i += 64)
                        MD5Transform (context->state, &input[i]);

                index = 0;
        }
        else
                i = 0;

        /* Buffer remaining input */
        MD5_memcpy((POINTER)&context->buffer[index], (POINTER)&input[i],inputLen-i);
}

/* MD5 finalization. Ends an MD5 message-digest operation, writing the
   the message digest and zeroizing the context.
  */
void MD5Final (unsigned char digest[16], MD5_CTX *context)
{
        unsigned char bits[8];
        unsigned int index, padLen;

        /* Save number of bits */
        Encode (bits, context->count, 8);

        /* Pad out to 56 mod 64.
        */
        index = (unsigned int)((context->count[0] >> 3) & 0x3f);
        padLen = (index < 56) ? (56 - index) : (120 - index);
        MD5Update (context, PADDING, padLen);

        /* Append length (before padding) */
        MD5Update (context, bits, 8);

        /* Store state in digest */
        Encode (digest, context->state, 16);

        /* Zeroize sensitive information.
        */
        MD5_memset ((POINTER)context, 0, sizeof (*context));
}

/* MD5 basic transformation. Transforms state based on block.
  */
static void MD5Transform (uint32 state[4], unsigned char block[64])
{
        uint32 a = state[0], b = state[1], c = state[2], d = state[3], x[16];

        Decode (x, block, 64);
```

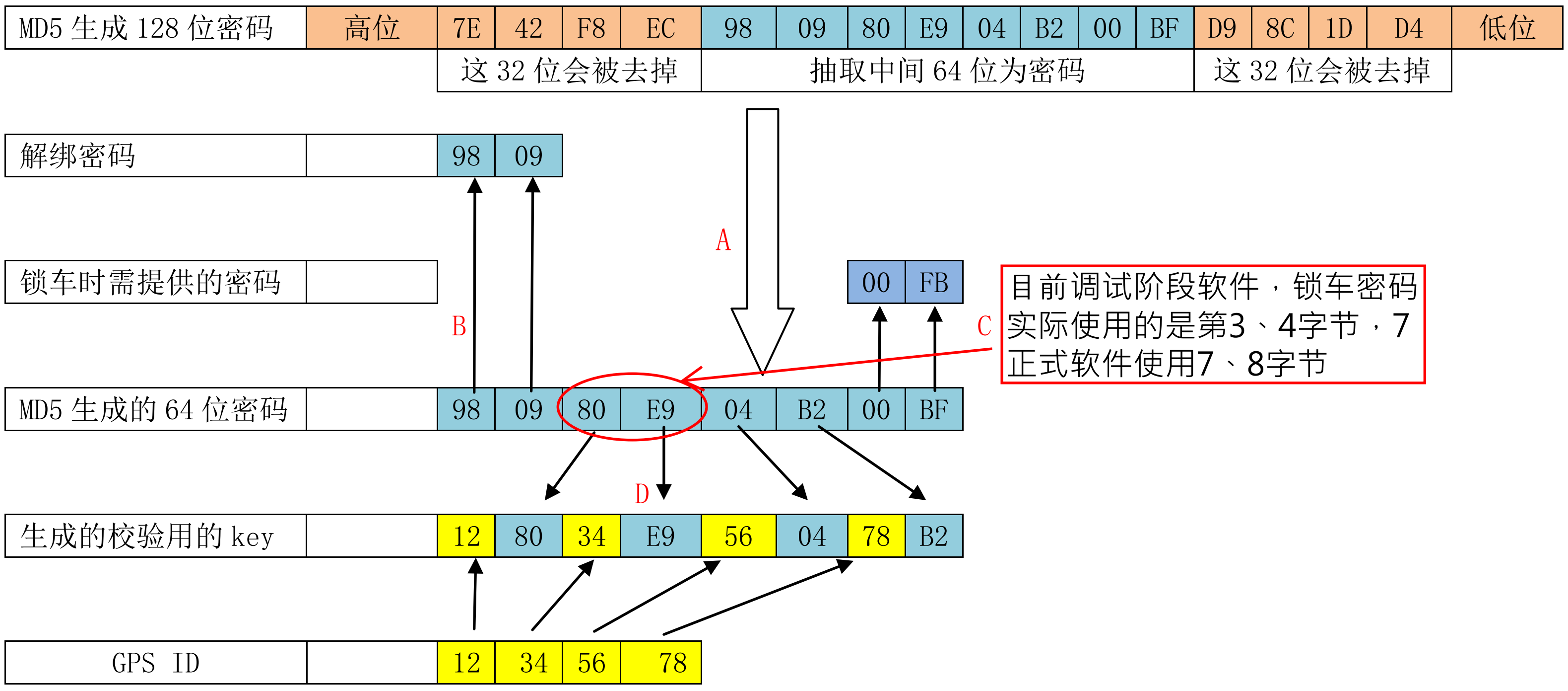
```
/* Round 1 */
FF (a, b, c, d, x[ 0], S11, 0xd76aa478); /* 1 */
FF (d, a, b, c, x[ 1], S12, 0xe8c7b756); /* 2 */
FF (c, d, a, b, x[ 2], S13, 0x242070db); /* 3 */
FF (b, c, d, a, x[ 3], S14, 0xc1bdceee); /* 4 */
FF (a, b, c, d, x[ 4], S11, 0xf57c0faf); /* 5 */
FF (d, a, b, c, x[ 5], S12, 0x4787c62a); /* 6 */
FF (c, d, a, b, x[ 6], S13, 0xa8304613); /* 7 */
FF (b, c, d, a, x[ 7], S14, 0xfd469501); /* 8 */
FF (a, b, c, d, x[ 8], S11, 0x698098d8); /* 9 */
FF (d, a, b, c, x[ 9], S12, 0x8b44f7af); /* 10 */
FF (c, d, a, b, x[10], S13, 0xffff5bb1); /* 11 */
FF (b, c, d, a, x[11], S14, 0x895cd7be); /* 12 */
FF (a, b, c, d, x[12], S11, 0x6b901122); /* 13 */
FF (d, a, b, c, x[13], S12, 0xfd987193); /* 14 */
FF (c, d, a, b, x[14], S13, 0xa679438e); /* 15 */
FF (b, c, d, a, x[15], S14, 0x49b40821); /* 16 */

/* Round 2 */
GG (a, b, c, d, x[ 1], S21, 0xf61e2562); /* 17 */
GG (d, a, b, c, x[ 6], S22, 0xc040b340); /* 18 */
GG (c, d, a, b, x[11], S23, 0x265e5a51); /* 19 */
GG (b, c, d, a, x[ 0], S24, 0xe9b6c7aa); /* 20 */
GG (a, b, c, d, x[ 5], S21, 0xd62f105d); /* 21 */
GG (d, a, b, c, x[10], S22,    0x2441453); /* 22 */
GG (c, d, a, b, x[15], S23, 0xd8a1e681); /* 23 */
GG (b, c, d, a, x[ 4], S24, 0xe7d3fbc8); /* 24 */
GG (a, b, c, d, x[ 9], S21, 0x21e1cde6); /* 25 */
GG (d, a, b, c, x[14], S22, 0xc33707d6); /* 26 */
GG (c, d, a, b, x[ 3], S23, 0xf4d50d87); /* 27 */
GG (b, c, d, a, x[ 8], S24, 0x455a14ed); /* 28 */
GG (a, b, c, d, x[13], S21, 0xa9e3e905); /* 29 */
GG (d, a, b, c, x[ 2], S22, 0xfcefa3f8); /* 30 */
GG (c, d, a, b, x[ 7], S23, 0x676f02d9); /* 31 */
GG (b, c, d, a, x[12], S24, 0x8d2a4c8a); /* 32 */

/* Round 3 */
HH (a, b, c, d, x[ 5], S31, 0xfffa3942); /* 33 */
HH (d, a, b, c, x[ 8], S32, 0x8771f681); /* 34 */
HH (c, d, a, b, x[11], S33, 0x6d9d6122); /* 35 */
HH (b, c, d, a, x[14], S34, 0xfde5380c); /* 36 */
HH (a, b, c, d, x[ 1], S31, 0xa4beea44); /* 37 */
HH (d, a, b, c, x[ 4], S32, 0x4bdecfa9); /* 38 */
HH (c, d, a, b, x[ 7], S33, 0xf6bb4b60); /* 39 */
HH (b, c, d, a, x[10], S34, 0xbebfbc70); /* 40 */
HH (a, b, c, d, x[13], S31, 0x289b7ec6); /* 41 */
HH (d, a, b, c, x[ 0], S32, 0xeaa127fa); /* 42 */
HH (c, d, a, b, x[ 3], S33, 0xd4ef3085); /* 43 */
HH (b, c, d, a, x[ 6], S34,    0x4881d05); /* 44 */
HH (a, b, c, d, x[ 9], S31, 0xd9d4d039); /* 45 */
HH (d, a, b, c, x[12], S32, 0xe6db99e5); /* 46 */
HH (c, d, a, b, x[15], S33, 0x1fa27cf8); /* 47 */
HH (b, c, d, a, x[ 2], S34, 0xc4ac5665); /* 48 */

/* Round 4 */
II (a, b, c, d, x[ 0], S41, 0xf4292244); /* 49 */
II (d, a, b, c, x[ 7], S42, 0x432aff97); /* 50 */
II (c, d, a, b, x[14], S43, 0xab9423a7); /* 51 */
II (b, c, d, a, x[ 5], S44, 0xfc93a039); /* 52 */
II (a, b, c, d, x[12], S41, 0x655b59c3); /* 53 */
II (d, a, b, c, x[ 3], S42, 0x8f0ccc92); /* 54 */
II (c, d, a, b, x[10], S43, 0xffeff47d); /* 55 */
II (b, c, d, a, x[ 1], S44, 0x85845dd1); /* 56 */
II (a, b, c, d, x[ 8], S41, 0x6fa87e4f); /* 57 */
II (d, a, b, c, x[15], S42, 0xfe2ce6e0); /* 58 */
II (c, d, a, b, x[ 6], S43, 0xa3014314); /* 59 */
II (b, c, d, a, x[13], S44, 0x4e0811a1); /* 60 */
```

```
        II (a, b, c, d, x[ 4], S41, 0xf7537e82); /* 61 */
        II (d, a, b, c, x[11], S42, 0xbd3af235); /* 62 */
        II (c, d, a, b, x[ 2], S43, 0x2ad7d2bb); /* 63 */
        II (b, c, d, a, x[ 9], S44, 0xeb86d391); /* 64 */

        state[0] += a;
        state[1] += b;
        state[2] += c;
        state[3] += d;

        /* Zeroize sensitive information.
        */
        MD5_memset ((POINTER)x, 0, sizeof (x));
    }

    /* Encodes input (uint32) into output (unsigned char). Assumes len is
        a multiple of 4.
      */
    static void Encode (unsigned char *output, uint32 *input, unsigned int len)
    {
        unsigned int i, j;

        for (i = 0, j = 0; j < len; i++, j += 4) {
            output[j] = (unsigned char)(input[i] & 0xff);
            output[j+1] = (unsigned char)((input[i] >> 8) & 0xff);
            output[j+2] = (unsigned char)((input[i] >> 16) & 0xff);
            output[j+3] = (unsigned char)((input[i] >> 24) & 0xff);
        }
    }

    /* Decodes input (unsigned char) into output (uint32). Assumes len is
        a multiple of 4.
      */
    static void Decode (uint32 *output, unsigned char *input, unsigned int len)
    {
        unsigned int i, j;

        for (i = 0, j = 0; j < len; i++, j += 4)
            output[i] = ((uint32)input[j]) | (((uint32)input[j+1]) << 8) |
            (((uint32)input[j+2]) << 16) | (((uint32)input[j+3]) << 24);
    }

    /* Note: Replace "for loop" with standard memcpy if possible.
      */

    static void MD5_memcpy (POINTER output, POINTER input, unsigned int len)
    {
        unsigned int i;

        for (i = 0; i < len; i++)
            output[i] = input[i];
    }

    /* Note: Replace "for loop" with standard memset if possible.
      */
    static void MD5_memset (POINTER output, int value, unsigned int len)
    {
        unsigned int i;

        for (i = 0; i < len; i++)
            ((char *)output)[i] = (char)value;
    }
    /* Digests a string and prints the result.
*/
```

5. GPSID 和 MD5 算法→YC 算法→握手校验密码，主动锁车密码，解绑密码的过程
   先由 MD5 生成 128 位密码，之后去掉高 32 位和低 32 位密码，得到中间的 64 位密码。对应下图的 A 标签。

从 64 位密码抽取第七第八字节作为解绑密码（对应下图的 B 标签），抽取第五第六字节作为锁车时需提供的密码（对应下图的 C 标签）。

从 64 位密码抽取第三至第六字节，之后与 GPS ID 交叉混合之后，形成了生成校验用的 key，如下图 D 标签。



目前调试阶段软件，锁车密码实际使用的是第3、4字节，7正式软件使用7、8字节