30 利用计算机模拟实现 Enigma 加密机（C语言版）

Jul 2014

Category: C/C++ enigma security encryption

Enigma，二战时期与德国战车捆绑的顶级加密机，它是世界上首台具有比较强悍的加密算法的加密机器，它彻底淘汰了手功加密。而为了破解它，英国政府几乎倾家荡产。（可阅读《密码传奇》）

现在我们来用计算机模拟一个使用 Enigma 加密算法的加密器，这个加密器使用 C 语言实现，在 Visual Studio 2013 + Windows 8.1 英文企业版下测试通过。（文末会附上代码）

如果您不了解 Enigma 的机器结构，点击 http://zh.wikipedia.org/wiki/Enigma，您必须了解其结构才能理解接下来的加密算法。这里实现的加密方式是，以字节为单位进行加密。

（3个转轮的）Enigma 的加密流程是：

> 输入字符 → 输入轮（一个简单的单表加密，这里简化掉了） → 转轮A → 转轮B → 转轮C → 反射板 →

1、数据结构

Enigma 的核心是 转轮（Wheel）和反射板（ReflectBoard）。每个转轮就是一个会转动的1对1映射表，而反射板则是具有 a→b→a 关系的映射表。

假设有3个转轮，每个的结构都是

```
typedef struct {
    unsigned char uElems[256]; /* 每个字节可以有 0 到255 这 256 个状态。*/
    unsigned char urElems[256]; /* 反向映射表 */
    unsigned char uPos; /* 转轮位置，初始化时应该设置为 0。*/
} EnigmaWheel;
```

由于经过反射板之后，还要依次反向通过转轮加密，所以还得有一个反向映射表。

而反射板只有一个，这里不考虑反射板转动的情况，那么结构很简单，直接用

```
typedef unsigned char EnigmaRefBoard[256];
```

即可。

2、构造转轮和反射板

如上所述，转轮是单表加密的映射表，也就是说每个字节的 256 种值每个对应唯一的映射结果。绝无重复——但是可以映射为本身，不要忽略这个情况。

举个实际的例子：

```
0 -> 233
1 -> 7
2 -> 254
3 -> 255
4 -> 31
...
253 -> 111
254 -> 213
255 -> 99
```

那么一个字节 4 经过这个转轮时就变成了 31。以此类推，经过所有转轮后进入了反射板。

再根据上表构造一个反向映射表：

```
...
7 -> 1
...
31 -> 4
...
99 -> 255
...
111 -> 253
...
```

反射板的的特点是：a → b → a。也就是，{x | x = f(f(x))}。

举个例子就是

```
0 -> 255
1 -> 100
2 -> 123
3 -> 4
4 -> 3
```

```
...
100 -> 1
...
123 -> 2
...
255 -> 0
```

这里我不说构造的算法，因为构造反射板和转轮的算法可以随意创造，甚至可以你手动设置。当然，文末附带的代码里有一个可用的算法。

但是可以说一下转轮反向映射表的构造方法，假设正向映射表已经构造完毕。

```
void enigmaRWheel(EnigmaWheel *pW) {
    int i;
    for (i = 0; i < 256; i++)
        pW->m_uElems[pW->m_uElems[i]] = i;
}
```

3、加密过程

现在我们假设反射板和转轮的设置好了，那么就可以开始加密了。

```
/* 此函数把一个字节输入到某个转轮中，得到转换结果 */
unsigned char enigmaWheelDo(EnigmaWheel *pWheel, unsigned char bIn) {
    /**
     * 转轮转动的实质是 uElems 表转动到 uPos 位置，那么实际上就是
     *    pWheel->uElems[bIn + pWheel->uPos]
     * 但是考虑到 uElems 只有 256 个元素，所以应该写成如下形式
     */
    return pWheel->uElems[(bIn + pWheel->uPos) % 256];
}

/* 此函数把一个字节反向输入到某个转轮中，得到转换结果 */
unsigned char enigmaWheelRDo(EnigmaWheel *pWheel, unsigned char bIn) {
    /**
     * 转轮转动的实质是 uElems 表在 uPos 位置，那么实际上就是
     * pWheel->urElems[bIn] - pWheel->uPos
     */
    return pWheel->urElems[bIn] - pWheel->uPos;
}

/* 此函数对一个缓冲区进行加密，并写回原缓冲区中。 */
void enigmad(
    EnigmaRefBoard pRB, /* 反射板 */
    EnigmaWheel pWheels[3], /* 转轮 */
    unsigned char *pBuf /* 输入输出缓冲区 */,
    size_t uBufSize /* 缓冲区大小 */
) {
    int i, j;
    unsigned char t;

    for (i = 0; i < uBufSize; i++) {

        t = pBuf[i];

        for (j = 0; j < 3; j++) /* 正向通过转轮组 */
            t = enigmaWheelDo(&pWheels[j], t);

        t = pRB[t]; /* 反射板 */

        for (j = 3; j > 0; ) /* 反向通过转轮组 */
            t = enigmaWheelRDo(&pWheels[--j], t);

        pBuf[i] = t;

        /* 一轮转换完毕，第一个转轮转动一位，如果进位，那么uPos变成0 */

        t = 1; /* 用 t 为 1 表示要进位，0 表示不需要*/
        for (j = 0; t && j < 3; j++)
            if (++pWheels[j].uPos)
                t = 0;
        /* 注：其实这里可以利用 C 语言的语言特性进行快捷进位，但是为了描述算法，就不这么做
    }
}
```

好了，算法到此结束。其实 Enigma 是个非常简单的加密算法，但是破解方法却非常复杂，至少人力是很难做到的。而且，如果密文量不够，那是几乎无法破解的。此外，Enigma 算法的加密速度非常快，在个人电脑上实测，100M的文件，只需要 2 秒即可完成加密，而且是未经过优化的情况下。

最后，附上可用代码一份：fenying/libenigma。

程序命令行是

```
enigma <WHEELS-QUANTITY> <KEY> <INFILE> <OUTFILE>
```

感谢阅读。

<<< IPv6 服务开启时 PHP 连接 memcached 失败的原因

Windows 8.1 无法安装到GPT硬盘的解决方案 >>>

comments powered by Disqus