Reinventing the Travois: Encryption/MAC in 30 ROM Bytes

Gideon Yuval

Microsoft Research, Redmond, WA 98052, U.S.A.

Abstract. By using a large number of round, we hope to be able to scrounge an Sbox out of nowhere, in an environment for which even TEA and the SAFERs are gross overdesign.

1 Background

Some people in the software industry are looking into home-control systems, much preferably without stringing new wires. This raises a simple issue: I have no more access to my power-lines than a thief has; and ditto (even more so) for wireless.

Therefore, if we want the same security we now have by owning the wire, we need some kind of cryptologic authentication.

The CPUs considered for this are quite underpowered (by today's standards): 8051 or similar[1], 1KB flash EPROM, 64 bytes RAM, 128 bytes EEPROM, and a peak 1MHz instruction rate; that last figure is relatively very fast, since the wire is 10KBPS or less.

In the classic study of access-control weaknesses, Ali Baba could replay the "open" message, using the authenticator "simsim" (or "sesame"); since replay attacks were not blocked, it did not matter how strong the 40 thieves' crypto & authentication was on other fronts. We therefore have to use EEPROM to keep track of a serial number, and get rid of replayed ones.

To add to the problem, messages (including the authenticator) had better be kept down to 8 bytes or so, to give them some decent chance to get through all the line noise, with the wimpy power-supplies planned.

2 MACs & ciphers

To authenticate a message, between parties who share a secret, we need a keyed MAC. I notice that all the secure hashes in common use are keyed MACs for which the key is frozen at spec time, and used as a chaining variable for longer messages. I also notice that all these hashes/MACs are a block cipher use in Meyer/Davies feedforward mode. So it seems we need a block cipher.

The SAFER-SK family is great on the security front (unless the crooks get smarter than Lars Knudsen); but it needs 512 bytes ROM for its two S-boxes (even if we never decrypt), plus 1-2KB for the code. Getting that much space off a 1KB chip, which is also trying to get some useful work done, is obviously unrealistic.

TEA may indeed be a Tiny Encryption Algorithm on a 32-bit CPU; but on a chip for which 16-bit subtraction is already a design issue, it is liable to be rather less tiny.

In summary, we need a decent block-cipher that uses as little ROM and RAM as possible, except what is available anyway. The only resource we have in some abundance is CPU cycles - the 6.4msec it takes to ship an 8-byte message amount to 6,400 instructions at the peak rate.

3 Stealing the Sbox

Since the chip will also-have non-crypto code running on it, we can try to scrounge an Sbox in the memory used for that code. This Sbox is not designed by Coppersmith; it is not even designed by a semi-competent cryptologist; it is whatever bits are there when the assembler has done its thing.

But a 256 * 8 Sbox (the obvious size for S/W crypto), is large enough to avoid some attacks; and on that kind of chip, the coders want the code to do useful work, and the chip designers want high code-density; so such an Sbox ought to have enough entropy to make life interesting.

Using an 8-byte key and and 8-byte plaintext, the resulting 8051 encryptor is

```
$title(small slow 51 encryptor)
$nomod51
$nopaging
$list
 name slow_51_encrypt
 sbox equ ????h
 text data 32
 key data 41
 ofRounds equ 32
 size equ 8

 cseg
 org ????
 mov dptr,#sbox

Allrounds:
 mov r3,#ofRounds

Oneround:
 mov r0,#text
 mov r1,#key ; 8-byte key, 8-byte text
 mov r2,#size; =8
 mov text+8,text ; to get wraparound logic

subround:
 mov a,@r0
 add a,@r1
 movc a,@a+dptr ; dptr is frozen, pointing to table in ROM, which is
; just part of the code
 inc r0
 add a,@r0
 rl a ; make sure bits in table get ''scrambled'' some
 mov @r0,a
 inc r1
 djnz r2,subround

 mov text,text+8 ; finish wraparound logic
 djnz r3,Oneround
 end
```

For the 51-challenged, a C decompilation follows:

```
for(r=0; r<NumRounds; r++) {
  text[8]=text[0];
  for(i=0;i<8;i++) {
    text[i+1] =(text[i+1] + Sbox[(key[i]+text[i])%256])<<<1;
// rotate 1 left
 }
  text[0]=text[8];
}
```

The full .HEX file is

```
:1004B00090????7B20782079297A08852028E627DD
:0D04C00093082623F609DAF6852820DBE8EC
:00000001FF
```

The .LST file-excerpt below may help correlating the two. Lines have been truncated to fit

```
0200 90 C0 DE  mov dptr,#sbox
0203   Allrounds:
0203 7B 20    mov r3,#ofRounds

0205   Oneround:
0205 78 20    mov r0,#text
0207 79 29    mov r1,#key ; 8-byte key, 8-. . .
0209 7A 08    mov r2,#size; =8
020B 85 20 28  mov text+8,text ; to get . . .

```

```
020E     subround:
020E E6     mov a,@r0
020F 27     add a,@r1
0210 93     movc a,@a+dptr ; dptr is . . .
0211 08     inc r0
0212 26     add a,@r0
0213 23     rl a ; make sure bits . . .
0214 F6     mov @r0,a
0215 09     inc r1
0216 DA F6     djnz r2,subround

0218 85 28 20  mov text,text+8 ; finish . . .
021B DB E8     djnz r3,Oneround
```

The 32 *rounds* (each byte gets hit 32 times; we look up that Sbox 256 times) are there to cover up for the many weaknesses of the design. Also:

1. Not all bit-planes in the Sbox will be equally random, nonlinear, ? ; the rotate instruction spreads the XORing among all the bit-planes.
2. The key schedule is stupid, making parts of the encryptor commute with other parts (with or without rotating the key and text). This is a code-size issue. It also lets us ship the message while the authentication code is running.
3. The modulo-2 additions do not commute with the rotation; this should stop attacks like Biham and Ben-Aroya's on Lucifer, in which the XOR is moved up & down the flowchart.

On the other hand, this cipher lets carry-propagation do its thing at least as many times as TEA does, and uses no >8-bit operations (coding a 32-bit rotate, on a pure 8-bit chip like the 8051, is almost a major design-issue).

Using a public-domain BASIC/51 interpreter as a source for S-boxes, and checking out how many output bits change when one bit is changed in byte 7 of the input, we find

1 round	5+-2 bits change
2 rounds	24+-10 for the worst S-boxes, up to 28 for better ones
3 round	30+-5
4	32+-4

But the minimum over 1024 pairs usually goes something like

1 round	1
2	1
3	1
4	5

5 rounds 20, & stays there (+-)

So 5 rounds are needed before the outliers behave decently; and it seems unwise to go below 8 rounds if we want real security.

Since this cipher is intended to be used when SAFER would be way too expensive, I refer to it as TREYFER.

4 Protocol

To put this cipher into a protocol:

Messages can be repeated indefinitely; but they only get acted on once. The sender will repeat the message until it gets an ACK.

To get the useful content of a message, subtract the previous message from it (since we wait for an ACK, "previous" is well-defined between sender and receiver). Thus, 0 is never a useful content. Other contents are encoded so as to make common contents be small integers. This will make wraparound take a fairly long time; but not as long for (e.g.) "dim light to 75%" as for "open the garage door".

We now can have an 8-byte message that contains a 2-byte from/to field, a 4-byte counter/content field, and a 2-byte authenticator. All of these numbers can be juggled up & down. They will never give us Fort Knox security, but are likely to be more secure than other weaknesses around the house.

Since all messages are one encryption block, it seems the attacks by Preneel & van Oorschott[2] and by Bellare at al.[3] do not apply.

And, since we never code up a decryptor, we have a chance to export non-joke security without going to jail.

5 Standard S-box

If anyone wants to try his hands at breaking this kind of cipher, he should avoid his own code, and get something that someone else wrote. If nothing else is at hand, use the 1st 256 primes (all modulo 256), starting with 2 (the only even value in the table).

References

1. Hersch, Russ: 8051 Microcontroller FAQ, widely available on the Internet
2. Preneel, B. and van Oorschott, P., MDx-MAC and Building Fast Macs from Hash Functions, Crypto'95, p.1.
3. Bellare, M. Canetti, R. and Krawcyzk, H., Keying Hash Functions for Message Authentication, Crypto'96, p.1