Apr 14

Reversing Somfy RTS

13

Some time ago we got new motorized blinds at my office which are controlled by a small wireless remote control. This made me curious how these remotes actually work, and if I would be able to control the blinds from a PC.

In this post I'll explain the process of reverse engineering the protocol. For a detailed explanation of the protocol it self see here.

Capturing the communication

The remotes are of the type Somfy Smoove Origin RTS. Some searching revealed that RTS stands for "Radio Technology Somfy" and is a proprietary system. It uses an OOK(/ASK) modulated signal on 433.42 Mhz.

This was very convenient since I already had a microcontroller with an Aurel RX-4M50RR30SF 433.92 Mhz OOK receiver attached laying around from an other project. Although the frequency's don't exactly match, the receiver is able to pick up the signal if the remote is within ~10 meters. The microcontroller, a Cypress EZ-USB FX2, was running a very simple program that samples the data output from the RF receiver every 35 us and forward it to my PC through USB.

With this setup I created traces of different button presses; 4 times 'up', 4 times 'down' and so on.

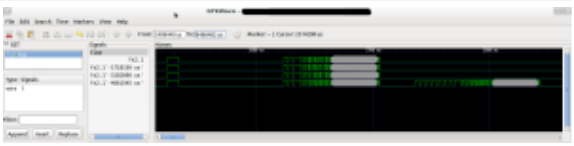
Analysing traces

The trace files are just a big bunch of 1's and 0's, not very readable. So I searched for a way to plot the traces.

I started of using GnuPlot. Although very cumbersome and time consuming, I was able to get some useful images out of it. Later I converted my trace files to VCD files and used GTKWave. GTKWave isn't very user friendly either, but it

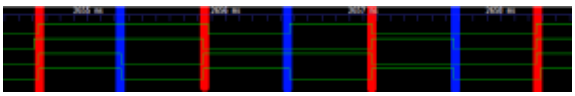
works a lot better then GnuPlot, especially with longer traces.

The image below shows 4 traces of pressing the same button. clearly visible and it is pretty obvious that there is a preamble the frame.



GTKWave showing 4 traces of Somfy RTS frames

The next image is a close up of the data. One thing I noticed when looking at the differences between traces is that there is always one point where all traces change state, followed by a point where only some or none of the traces changes state. I highlighted this in the image below, red always changes, blue sometimes. This suggests that the actual data bits are encoded in the edge at the point where the signal always changes state. One obvious encoding that springs to mind when seeing this is Manchester encoding.



Close up of Somfy RTS data. On the red markers the signal level always changes, on the blue only sometimes

Payload Data

Now I knew how the data was modulated, framed and encoded I could actually analyse the frame payload data. As always when reverse engineering I started by making a list of data fields I would expect to be in the protocol and then started comparing the frames from the different traces.

This directly showed that pressing the same button multiple times resulted in different data. Suggesting that the protocol is not static, but uses some kind of rolling code/sequence number. Comparing the traces of two different buttons shows that the upper nibble of the second byte is probably the button code.

Looking at the bytes that change with every button press I noticed that some actually behave as counters that constantly increase by 1, but with some bits flipped. So I assumed that there was some sort of XOR based obfuscation going on. Using these counters I started XOR'ing nibbles together, till the counters increased as expected. This finally brought me at a point where I was able to predict the data multiple frames ahead of the current frame. But it wasn't perfect.

Patents...

With all the information gathered so far I went back to Google and stumbled upon US Patent 8,189,620 B 2. Skimming through it I found an image of the

frame structure I already determined. Hmmm… interesting! Reading further I found that this patent almost completely describes the RTS protocol. With this new information I was able to determine the obfuscation algorithm used.

An other interesting patent I found was US 7860481 B2. But that one doesn't describe the payload data.

Taking over control!

Now I could decode the protocol I went on to write a program to actually send out my own frames. Given a captured address, rolling code and 'encryption key' I was indeed able to control the blinds. Victory!

To transmit I use a Aurel TX-SAW-MID 5v transmitter hand soldered on an experiment board. The antenna attached is a home-brew 1/4 lambda antenna with 4 radials. I was told this antenna design should sort of match the output impedance of the transmitter, although I've got no tools to test this.

With this setup I was able to control the blinds at about 5 meter distance, with a wall/coated window in between. This isn't very impressive, although expectable given that I'm transmitting at 0.5 MHz above the centre frequency of the receiver. And all soldering and wiring is suboptimal. The actual range also depends on the blinds. One of them I wasn't able to control at all, even when standing next to it.

One problem was that if I controlled the blinds with my program, the rolling code would advance. This caused the original remote to become out of sync. Luckily the blinds allow multiple remotes to be bound to it. This way I could create a virtual remote with my program with its own rolling code.

Security

When I started of I didn't expect this to be secure in any way. The requirements on security probably aren't very high for these systems, since some attacker controlling my blinds would just be very annoying. But it is actually better then I anticipated(as in just replaying a frame doesn't work).

Obviously the 'encryption' doesn't offer any protection. Even if the algorithm would have been strong, only 4 bits of the 8-bit 'key' actually change(in a predictable manner…). And the 'key' is send with the frame in the clear.

So what about the rolling code? Well it is 16-bit, so there is a chance of 1 out of 65535 that you guess the next value of the code. But if the receiver only accepted the next value of the rolling code as valid and the remote is out of range, one button press would mean that receiver and remote will be out of sync. To prevent this the receiver uses a window in which the rolling code must fall. This size of this window is somewhere in the order of ~100. So this would give a chance of 1 in 655 to guess a working rolling code, or not? Somfy actually did something clever here, because the receiver seems to shrink the window every time it receives an incorrect rolling code! So in worst case you still have to try 65535 codes.

An important thing to note is that the rolling code is sequential and starts at 0. So especially for new systems it is relatively easy to predict/calculate an approximation of the rolling code using the usage frequency + age of the system. Also the address isn't a random number, but seems to be assigned sequentially. So given the age of the system, or the address of an other remote in the system you might be able to greatly reduce the address space to search.

Tagged *ISM-band*, *Reverse engineering*, *RTS*, *Somfy*

13 THOUGHTS ON "REVERSING SOMFY RTS"

Thomas Dankert

— MAY 21, 2014 AT 2:44 PM

Hi, I've got somfy blinds installed in my house and would love to automate them as well.
I also found the patent describing RTS, but was unable to figure out the "encryption".

Thanks for this really interesting read!
Do you have the code to send signals available? I have several 433mhz transmitters on hand…

Reply

pushstack

— MAY 22, 2014 AT 3:17 PM

Glad you liked the post.
Currently I'm reluctant to release the code that actually control the blinds. Since that is what the patent describes, and I don't want to get into any patent war.
But implementing the protocol should be trivial.

Reply

Thomas Dankert

— MAY 22, 2014 AT 3:22 PM

I see your point, thanks anyway for the description!
In case you're interested, I based my research on the drivers for control4 devices (found here: http://www.control4.com.tr/eng/drivers_somfy.html)

These are lua-based scripts, with "encryption" that is basically just an XOR.
I "decrypted" them and found the meaning of some more control codes (used for dimming lamps, and setting the "MY" position).

Reply

pushstack

— MAY 22, 2014 AT 3:39 PM

Cool, will definitely take a look at that. Thanks for the info.

Reply

Tim

— MAY 28, 2014 AT 1:36 PM

Hi Thomas, Can you explain me how to decrypt the control4 somfy drivers?

Thanks in advance

Tim

Reply

Thomas Dankert

— MAY 30, 2014 AT 8:02 AM

Hi Tim,

here we go:
The "driver" is a XML file with an embedded lua script. The control4-box seems to know about the air transmission format (OOK, 433.42Mhz, etc), so the script only constructs the frame.

The encryption is standard AES, but I really do not understand why they chose to implement it like that.
They do use AES, but only to encrypt a simple counter (a 16 byte array), that is then used to XOR the plaintext with.

1) Base64-decode the contents of the tag.
2) Setup AES in ECB Mode, with IV = 0 and Blocksize of 128 bits.
3) AES key=…. (I don't think I can post this here, but it can be found in the DriverEditor executable, it is saved in plaintext there)
4) Initialize counter to 0
5) Do this for 16 byte blocks until the end of the encrypted text:
5a) increment counter
5b) encrypt the current counter value with AES
5c) for each byte of the current block of encrypted text:
6a) plaintext[currentBlock + currentByteIndex] = encryptedBytes[currentBlock + currentByteIndex] XOR counter_encrypted[currentByteIndex]
5d) read next 16 byte block (or less, if the end of encrypted text is reached)
5e) repeat from 5a

As I said, I don't know why they choose to encrypt the counter with AES, instead of directly encrypting the plaintext.
The XOR afterwards adds nothing to security.

You may find the Control4 Driver Editor application a really useful source of information concerning the encryption.
Download it from here:
http://www.control4.com/documentation/Composer_Pro_User_Guide/Using_DriverEditor.ht

Use a .NET debugger or disassembler and have a look in the C4AES class, the key and algorithm are all there.

Reply

Marco

— JULY 16, 2014 AT 12:27 AM

Control4 drivers encryption???

Are you saying that you can decrypt the c4 'encryption' found on some drivers?

If so could you email me as I would like some more information on how to do this as I have a couple of drivers I would like to see how they put together to use as a starting point for another driver…

I am not planing to use the code on the original driver

Just learn with it so I can create my own…

Would be really appreciated…

Thanks!

Reply

Rick

— DECEMBER 27, 2015 AT 9:57 PM

Hi Thomas,

thank you for the spoiler
In fact, it is really easy to decompile the DriverEditor .NET application and the code is just right there in front of you. I spent some time and ported the code from C# to standard C, it is attached below – it just needs the key, which can be gotten as you described above. Once the key is set, this little C program will dump the driver code on stdout decoded.
It has been built against the AES256 implementation from literatecode.com. Enjoy

```
#include
#include
#include
#include
#include
#include "aes256.h"

static void incrementCounter();
```

```
#define MAX_FILE_SIZE 1024*1024

void main(argc, argv)
int argc;
char **argv;
{
int AES_BLOCK_SIZE = 0x10;
static uint8_t AES_KEY[32] = { // add 32 bytes of key here
};
aes256_context ctx;
struct stat s;
char *fbuf;
FILE *f;
uint8_t bytes[MAX_FILE_SIZE];
uint8_t buffer2[MAX_FILE_SIZE];
uint8_t counter[16];
uint8_t buffer[16];
int i, j, l;

if (argc != 2)
exit(1);
if (stat(argv[1], &s) == -1) {
perror(argv[1]);
exit(1);
}
if (s.st_size > MAX_FILE_SIZE) {
fprintf(stderr, "File size too large, consider recompiling with larger
MAX_FILE_SIZE value\n");
exit(1);
}
if ((fbuf = calloc(1, s.st_size)) == NULL) {
fprintf(stderr, "cannot allocate %d bytes\n", (int)s.st_size);
exit(1);
}
if ((f = fopen(argv[1], "r")) == NULL) {
perror(argv[1]);
exit(1);
}
if (fread(fbuf, 1, s.st_size, f) != s.st_size) {
fprintf(stderr, "Incomplete read\n");
exit(1);
}
fclose(f);
memcpy(bytes, fbuf, s.st_size);
for (i = 0; i < s.st_size; i += AES_BLOCK_SIZE) {
incrementCounter(counter);
aes256_init(&ctx, AES_KEY);
memcpy(&buffer, &counter, AES_BLOCK_SIZE);
aes256_encrypt_ecb(&ctx, buffer);
if (AES_BLOCK_SIZE < s.st_size – i)
l = s.st_size – i;
```

```
else
l = AES_BLOCK_SIZE;
for (j = 0; j = 0; i–) {
counter[i] = counter[i] + 1;
if (counter[i] != 0)
return;
}
}
```

Reply

Rick

— DECEMBER 27, 2015 AT 10:02 PM

Damn, it got mangled by wordpress.

{code}
Trying
{code}

Reply

Rick

— DECEMBER 27, 2015 AT 10:05 PM

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include "aes256.h"

static void incrementCounter();

#define MAX_FILE_SIZE 1024*1024

void main(argc, argv)
int argc;
char **argv;
{
int AES_BLOCK_SIZE = 0x10;
static uint8_t AES_KEY[32] = { // add 32 bytes of key
here
};
aes256_context ctx;
```

```
struct stat s;
char *fbuf;
FILE *f;
uint8_t bytes[MAX_FILE_SIZE];
uint8_t buffer2[MAX_FILE_SIZE];
uint8_t counter[16];
uint8_t buffer[16];
int i, j, l;

if (argc != 2)
exit(1);
if (stat(argv[1], &s) == -1) {
perror(argv[1]);
exit(1);
}
if (s.st_size > MAX_FILE_SIZE) {
fprintf(stderr, "File size too large, consider recompiling
with larger MAX_FILE_SIZE value\n");
exit(1);
}
if ((fbuf = calloc(1, s.st_size)) == NULL) {
fprintf(stderr, "cannot allocate %d bytes\n",
(int)s.st_size);
exit(1);
}
if ((f = fopen(argv[1], "r")) == NULL) {
perror(argv[1]);
exit(1);
}
if (fread(fbuf, 1, s.st_size, f) != s.st_size) {
fprintf(stderr, "Incomplete read\n");
exit(1);
}
fclose(f);
memcpy(bytes, fbuf, s.st_size);
for (i = 0; i < s.st_size; i += AES_BLOCK_SIZE) {
incrementCounter(counter);
aes256_init(&ctx, AES_KEY);
memcpy(&buffer, &counter, AES_BLOCK_SIZE);
aes256_encrypt_ecb(&ctx, buffer);
if (AES_BLOCK_SIZE < s.st_size – i)
l = s.st_size – i;
else
l = AES_BLOCK_SIZE;
for (j = 0; j < l; j++)
buffer2[i + j] = (uint8_t)(bytes[i + j] ^ buffer[j]);
}
printf ("%s\n", buffer2);
exit (0);
}
```

```
static void incrementCounter(counter)
char *counter;
{
int i;

for (i = 15; i >= 0; i–) {
counter[i] = counter[i] + 1;
if (counter[i] != 0)
return;
}
}
```

Reply

Thomas Dankert

— JULY 16, 2014 AT 6:12 AM

Hi Marco,

shoot me an email, and we can discuss this further: thomas AT dankerts DOT eu

Bye,
Thomas

Reply

Dimitris Staikopoulos

— MAY 5, 2015 AT 7:58 AM

Dear Pushstack and Thomas . I was trying as well to control my Somfy in a different way and read your posts. Great job although I am not into programming so much. I was browsing to see if a broadlink RM Pro would work for my Somfy. No information on the Internet.
Broadlink they were not sure if works. Then I read about the rolling codes and figured that wont work.

Reply

epub

— OCTOBER 9, 2015 AT 9:39 AM

I really like what you guys are up too. This sort

of clever work and reporting! Keep up the fantastic works guys I've included you guys to my blogroll.

Reply

Leave a Reply

Enter your comment here...

Somfy + RTL SDR →
