Encryption Made Easy

Abstract: Encypted communication between small microcontrollers can be used for firmware upgrade, or to exchange of sensitive data. These microcontrolles do not have the power to implement a full RSA, DES or AES, but need an equally safe, and very simple algorithm.

1. History

The need to create, transmit and decode messages dates back long before the computer era. As soon as information was recorded in writing, the need to protect it from prying eyes had arisen. Encryted communication was especially important in wartime. And, although it is not a thing to be proud of, war is just as old as mankind. Learning the intention of the enemy often decided the fate of a battle, sometimes even a war. Breaking of the Enigma code by the British during World War 2 was a substantial aid to the Allied war effort.

After the industrial revolution the need to protect information increased in the civil sphere too. A new machine could mean a fortune to those who invented and used it first. Complicated legal system, copyright and patent laws are built to protect the intellectual property.

Today, in the information era, information is the most valuable, and the best protected asset. Legal protection is no longer enough, because it is too slow and cumbersome. Microcontrollers employ immensely sophisticated methods to store, transmit and receive information securely. And hackers with high power computers are trying to break the codes and steal the information.

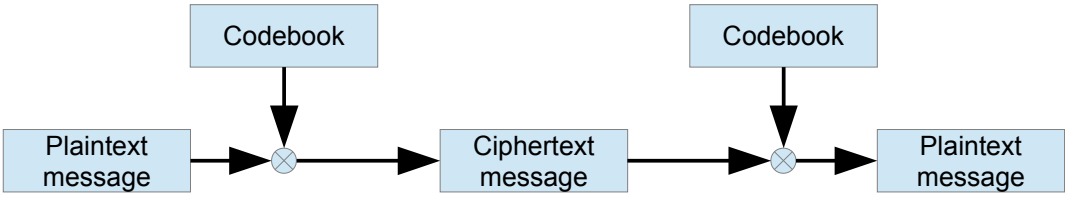
The complexity of microcontrollers, and the size of the firmware stored inside them, are steadily increasing. Today a firmware stored inside a small microcontroller could worth tens, or even hundreds, of thousands of euros. As programs get more complicated, they become more and more susceptible to programming errors. It is increasingly important to correct these errors, upgrade the firmware regularily and include new functions.

The safest way to upgrade a firmware is to bring back all microcontrollers to the factory, and do it in a safe environment by trusted employes. But it is expensive, uncomfortable, and not always possible.

Firmware upgrade over an insecure communication channel, or on the field by not fully trusted employes, is cheaper, but presents security risks. Encrypting data greatly reduces these risks, but the decoder uses some of the very limited resources of the microcontroller, and therefore it also costs money. The use of full blown RSA, DES or AES encryption is almost always out of question, because they are too complicated. There is a need for an equally safe, but far simpler algorithm.

2. Requirements

The general structure of an encrypted communication is:



Assume the attacker can

- capture almost all messages,
- suppress some messages,
- modify, replace or retransmit any message,
- get hold of some messages both in plain and in encrypted form,
- figure out the encryption function,
- but cannot get the codebook.

Ensure that despite these he cannot

- decode any message
- create any valid message
- modify any message without invalidating it.

Also desirable, if

- the encrypted message is not much longer than the original message
- the encryption function is simple, fast and symmetrical.

3. One-time pad encoding and the synchronous stream cipher

The oldest, and still the best, encryption method is the one-time pad. Both the sender and the receiver have an identical, quite large, codebook composed of completely random letters. The sender selects one page of the book that has never been used before, and sends the page number. Then he breaks up the message into letters, encrypts the first letter using the first letter, the second letter using the second letter, etc. of the selected page of the codebook. The encryption function is a simple, reversible lookup table operation. The receiver decodes the message by applying the inverse function to each letter of the encrypted message.

A computerized version of this algorithm is the synchronous stream cipher. The codebook is replaced with a smart bit generator, the page number is called key, the message is broken into individual bits, and the encryption function is the binary exlusive or operation.

The synchronous stream cipher has one flaw: if an attacker can change a bit in the ciphertext, he can make predictable changes to the corresponding plaintext bit; flipping a bit in the ciphertext causes the same bit to be flipped in the plaintext.

4. The codebook

The best codebook would have been made by recording a truly random data stream. Unfortunately small microcontrollers do not have enough memory to store this, and there is no way to compress it.

The next best thing is to use a high quality pseudo random number generator. These use simple algorithms to generate a data streams which have similar properties to true random data streams.

The simplest, and the most well known, pseudo random number generator is the Linear Feedback Shift Register. A maximum-length LFSR, with shift register length of n, cycles through all possible 2n-1 states within the shift register, except the state where all bits are zero. It is very easy to implement any kind of LFSR even on a small 8-bit microcontroller.

One LFSR may not produce enough bits for the entire codebook, and LFSRs are too well known to be used alone in a cryptographic algorithm. But combining the output of more than one LFSRs in a nonlinear way would make a longer and less predictable data stream. For example: combining the outputs of four different length LFSRs by first OR-ing the outputs of every two, and then AND-ing the results would make a reasonably good pseudo random number generator. The initial value of one LFSR can be used as key and the whole codebook would fit into less than 100 bytes on a PIC16 microcontroller.

The quality of the output can be further improved by replacing one LFSR with a small truly random data table. Fortunately one such table is always freely available: the binary code of the program implementing this algorithm is already there, and can be used as a poor quality random data table.

5. Signature

The weakness of the synchronous stream cipher is that changing one bit in the ciphertext, changes exactly one bit in plaintext. One bit change in firmware, either malicious or accidental, is worse than a failed firmware upgrade.

But this flaw can be corrected easily by appending a signature to the message. A signature can be a simple checksum, or a CRC generated by an LFSR, but the best way is to encrypt the message with another key, using the already implemented algorithm, and feed the result into the CRC generator.

6. Limitations

The algorithm, I described here, can only be used for coding time invariant messages. If the the meaning of the message depends on when it is sent, then the attacker can capture, reorder and resend messages to confuse the receiver. Firmware upgrade is time invariant. Every message is valid every time, and has the same meaning: one version of the firmware.

The length of the encrypted message is proportional to the length of the original message. If the length of the message has any relation to its contents then this would reveal some information to a listener. A simple solution to this problem is to pad every message to the same length.

The codebook is the weakest point of this encryption. If the codebook is broken then the whole exercise is pointless. Therefore it is imperative to design a good codebook. Combining the outputs of different type pseudo random generators in different ways, using long LFSRs, and translating the result using a small prerecorded table could make the codebook generator very hard to break.

7. As used for firmware upgrade

Facts:

- The microcontroller does not have enough flash memory to store two copies of the application firmware.
- And it does not have enough RAM to store more than one flash memory page of data.
- Erasing flash memory takes a long time, and it is irreversible.
- During erasing or programming the flash memory, the microcontroller is stalled and unable to communicate.
- Power or communication break can happen anytime. If there is a power break while erasing or programming the flash memory the contents will be undefined and unreliable.

Goal:

Create an algorithm that ensures the microcontroller can always ready to receive a new firmware, it never executes unverified or unreliably programmed code, and eventually upgrades the application firmware to the received version.

Solution:

The program memory of the microcontroller is divided into three regions:

BootCode

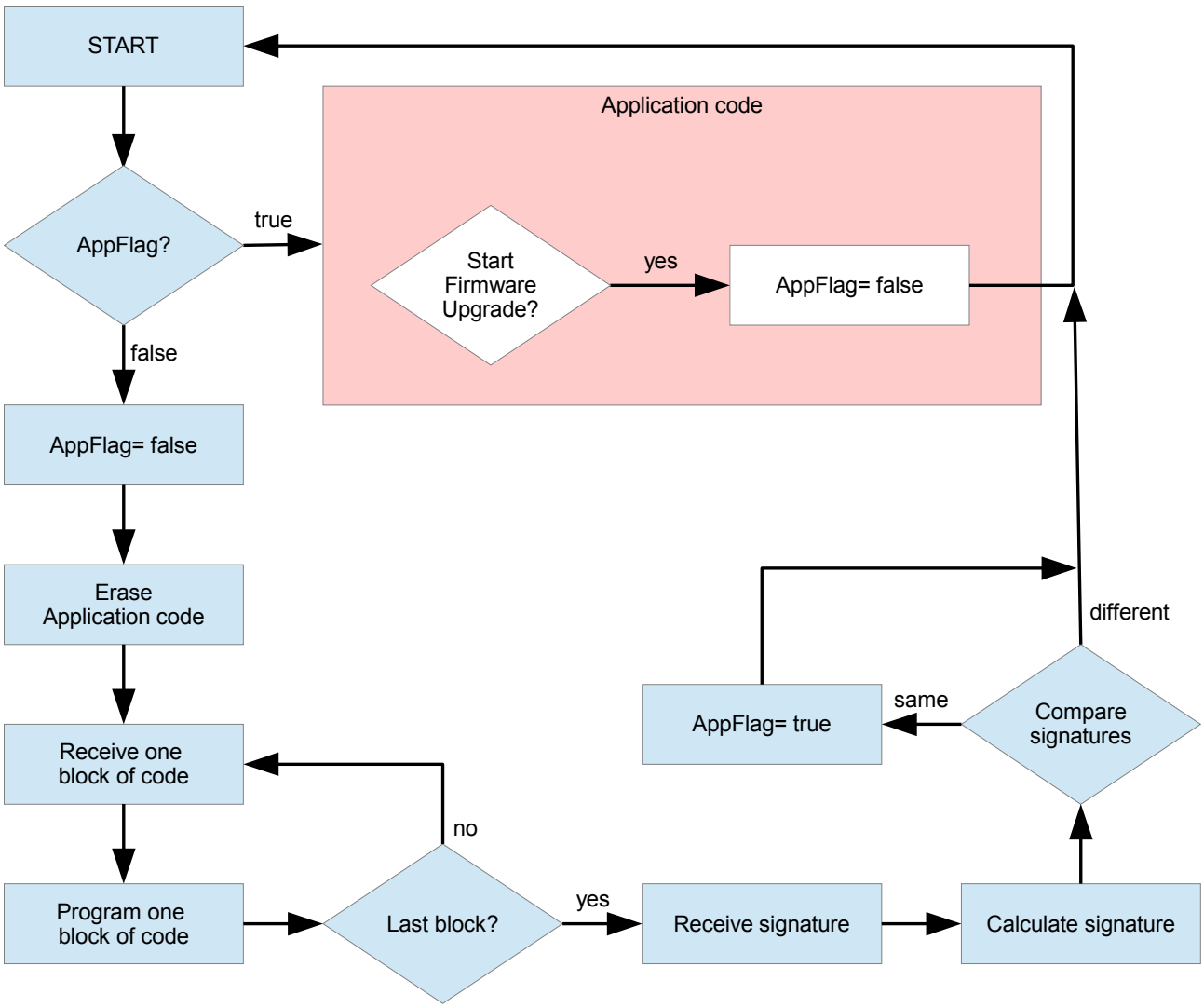
It holds the minimum, receive only, communication code, the codebook generator, the decoding algorithm and the flash erase and write functions. The boot code is write protected.

AppFlag

One bit long. (Actually one flash page long, of which only one bit is used.)

ApplicationCode

The rest of the program memory.



After power on, the BootCode starts. If AppFlag is true, it transfers control to ApplicationCode.

If AppFlag is false, it erases it again to make sure it is reliably false, and then erases the entire ApplicationCode too. Then it waits for the book number and initializes the codebook generator. Then it receives the first block of data, decodes it and programs it into the first page of the ApplicationCode memory. Data is transmitted in blocks, and there is enough time between blocks to finish programming. Then it receives the next block of data, and programs it into the next page of ApplicationCode memory, and repeats this till the end of the stream. Then it calculates the signature of the Application code and compares it to the received signature. If the signature is correct it sets the AppFlag, signals success and waits for reboot.

The ApplicationCode recognizes only the "Start Firmware Upgrade" command. When it receives it, it erases the AppFlag and reboots.

About me

I am an innovative freelance electronic design engineer, with more than 30-years industrial experience across Europe and beyond. Based in Austria, near the Hungarian border, with a complimentary team at hand for managing larger workloads, I can deliver novel and cost effective solutions using Microchip and other low cost embedded microcontrollers as both prototypes and production ready products. Amongst the hundreds of projects delivered are video cards, communications products, display systems, USB, I^2C, RS485, RFID, keyboard and other peripheral controllers. As well as in-depth problem solving to deliver a design, my dedication and input has meant many of these have gone on to become commercially successful in their own right, leading to repeat business and recommendations to new customers. Let me know, if I can help you to meet a design need.