Parallax Propeller 2
Code Authentication and Protection

By Perry Harrington

DRAFT v1.0

Code protection is a common feature of many microcontrollers available today. Protection of proprietary IP is a significant concern when bringing new products to market. Additional capabilities are found in higher level devices such as code signing for authenticating firmware updates and authentication to prevent untrusted 3rd party code from running on a product. The proposed system is more elaborate than conventional code protection schemes used in existing microcontrollers because the Propeller family does not have on-chip non-volatile memory but loads program code from external memory. I believe the proposed system is more robust than proprietary methods used by other manufacturers because it is open and uses well understood technologies and principles to meet the requirements of the market.

Propeller 2 Key

The Parallax Propeller 2 has 172 one time programmable fusible links within the chip. These fusible links are buried within the chip and are difficult to alter or detect due to the nature of how the metal layers of the chip are deposited. This document assumes that the fusible bits are the second weakest link within the system; key management is always considered the weakest link in any cryptographic system.

The fuse bits are readable using the COGID instruction 1 bit at a time via the carry flag. Upon reset, the Propeller 2 permits access to these bits until the COGID instruction is executed with a sentinel value in the destination field of the instruction. Once the internal flag has been set, the fuse bits may no longer be read until the next reset. This ensures the fuse bits are unavailable to user code.

I propose to allocate 128 of the 172 bits for key storage, 42 bits for user purposes, and reserve 2 bits to indicate external memory is encrypted and to write protect the fuse bits from further programming.

Trust Levels

Within computer security there is the concept of hierarchical trust, often called ring levels[1]. This document describes 3 levels, numbered Ring 0 to Ring 2 in descending levels of trust. Ring 0 is entered upon reset, Ring 1 is entered upon execution of the boot-loader, Ring 2 is entered upon execution of the user code.

Ring 0 is considered the most privileged level and has access to all chip resources. It is important Ring 0 authenticates the boot-loader used in Ring 1, so that untrusted code is not executed with escalated permissions. Ring 0 code does not employ any obfuscation or trickery to hide security aspects from scrutiny.

Ring 1 is the boot-loader code and is authenticated with a 2 round SHA-256 HMAC to obscure the key used for decrypting Ring 2 code. Because Ring 1 is considered privileged, it must be transparent and well written, thus a reference implementation will be provided to end users. The reference implementation of the Ring 1 boot-loader will implement AES-128-CBC

1 http://en.wikipedia.org/wiki/Ring_%28computer_security%29

decryption to load Ring 2 code into memory.

Ring 2 code is the user application written for the microcontroller, it is considered insecure and all privileges are dropped prior to execution. Ring 2 code may be susceptible to security vulnerabilities, which may allow untrusted code to be executed on the microcontroller, however the security keys are safe from untrusted Ring 2 code.

Propeller 2 Memory Map

The Propeller 2 is a Harvard architecture design with separate volatile memory for user code and data storage, with executable code running from local volatile memory in each core. There is 126KB of volatile HUB RAM and 2KB of volatile COG RAM for each COG. There are 8 COGs within the chip. User code cannot execute directly from HUB RAM, it must be copied into COG ram to execute. Upon boot-up, an external non-volatile memory is read and loaded into HUB memory, then user code is loaded into the first COG and executed.

The 2KB located above HUB memory is OTP ROM and contains the Ring 0 code. This leaves 2KB of space in EEPROM for the boot-loader and authentication hash.

ROM code

The ROM code executed in Ring 0 contains house-keeping and a SHA-256 implementation. Upon reset various chip setup tasks are performed, then the chip looks for a connection from an external device.

If an external device is present, the ROM code checks for the desired action, run or program. If the action is run, and the encrypted flag is set, access to the fuse bits is disabled, an unencrypted program is downloaded to HUB memory and [optionally][2] authenticated, then Ring 2 code is executed directly, skipping Ring 1.

If the action is to program the external memory, the ROM code simply downloads the data to be programmed into external memory and writes to the external memory. No authentication or decryption needs to be performed, this is done upon the next reset.

If an external device is not attached to the serial pins at reset, the ROM code proceeds to load the Ring 1 boot-loader from external memory. The *encrypted* flag is consulted to determine whether the fuse bits should be protected and whether authentication should be performed.

Programming the fuse bits for the first time is accomplished by downloading a program to HUB memory and running it. If the write protect is not enabled, the Ring 2 code can program the fusible links. It is the responsibility of the developer to set the write protect bit after the fusible bits are programmed. The process of setting the fusible bits should be handled by a wizard program on the PC to prevent accidental bricking of the chip.

HMAC[3]

Hash-based Message Authentication Code is a protocol for salting and hashing a message in a 2 round procedure. One iteration of HMAC consists of an inner and outer round of hashing, each with the same key XORed with a different value. The purpose of the two rounds is to obscure the key by mixing two sets of hashed data.

SHA-256

The SHA-256 hash algorithm is presently the minimum viable standard for hash-based authentication. The algorithm processes data in 512 bit blocks and produces a 256 bit hash of the message. There are presently no known attacks against this algorithm and it is

2 If the encrypted flag is **unset**, the fusible bits are assumed to be a user defined and access is allowed.

3 http://en.wikipedia.org/wiki/HMAC

presumed to be secure for up to 20 years.

AES-128

The AES algorithm is a symmetric block cipher, recognized as the NIST standard for encryption. This algorithm processes 16 byte blocks and uses a 128bit key. The algorithm is considered to be secure for up to 20 years with a 128 bit key, so there is symmetry between SHA-256 and AES-128 being used as a pair.

CBC[4]

One of the faults of a symmetric block cipher is that similar plaintexts will result in the same ciphertext. If you have a pattern that repeats in 16 bytes, the ciphertext will be the same. Furthermore, files that differ only slightly will have ciphertexts that contain a lot of duplication. From a security standpoint this is undesirable. IBM invented CBC in 1976 to ensure duplicate blocks within a file do not have duplicate ciphertexts, additionally there is a random initialization vector which is used to salt the ciphertext so that identical plaintexts do not generate identical ciphertexts. This initialization vector should not be reused in subsequent encryption operations.

How it all comes together

$00 - $1F	SHA-256 hash
$20 - $2F	CBC IV
$30 - $3F	Reserved
$40 - $1FF	Boot-loader
$200 - $1FFFF	User code

Table 1: Memory map of the external memory

The ROM code loads the boot-loader from external memory and performs HMAC authentication. The resulting hash is compared against the hash stored in the first block of external memory. If they match, the boot-loader is given the key and control of the chip. The boot-loader reads the CBC Initialization Vector from external memory and loads the User code into HUB memory. Next the boot-loader decrypts the first block in HUB memory then XORs the IV with the decrypted block, copies the ciphertext to the IV, then stores the decrypted block in HUB memory.

In the end, the whole authentication and encryption system is simple, uses industry accepted algorithms, and implements security in an open and robust manner.

4 http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation#Cipher-block_chaining_.28CBC.29