6.375: Complex Digital System Design
Final Proposal, High-level design and Test Plan

Lauren DE MEYER   -   Candace ROSS

1 Lightweight cryptography & SPECK

In the last few years, the need for small devices with very low computing power has awoken an interest for lightweight cryptography. Consider for example wireless sensors and the rapidly growing number of Internet of Things devices, becoming smaller and smarter.

SPECK [1] is a family of lightweight block ciphers that was introduced by the NSA in 2013. It was designed for flexibility and optimized for hardware implementations. Its round function only requires modular addition (A), rotations (R) and exclusive OR (X). The use of these three efficient operations is popular and the schemes that use them are called ARX ciphers. The round function is shown in Figure 1.
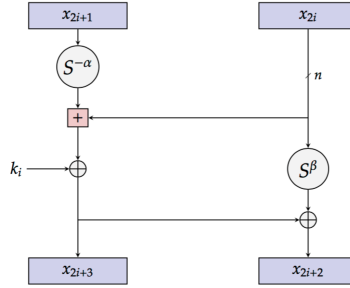

Figure 1: The SPECK Round function

One block of plaintext or ciphertext is $2n$ bits long with n the word size. As a family of block ciphers, SPECK allows for different choices for n, ranging from 16 to 64. For each choice of word size, there are one or two possible key sizes mn. For each pair of parameters (n, m), the designers have specified the number of rounds to use and the rotation parameters α and β (See Table 1).

The round function depicted in Figure 1 is applied T times, starting with x_1 and x_0 respectively the most significant and least significant n bits of a plaintext block.

$$R_k(x, y) = \left((S^{-\alpha}x + y) \oplus k, S^{\beta}y \oplus (S^{-\alpha}x + y) \oplus k\right)$$

Each round transforms (x_{2i+1}, x_{2i}) to (x_{2i+3}, x_{2i+2}) with R_{k_i} and after the last round, we obtain the 2 words (x_{2T+1}, x_{2T}) which form the ciphertext block.

The round keys k_i are determined from the master key K by a key schedule, which requires exactly the same hardware as the round function. We split the key K in its m words:

$$K = (l_{m-2}, \ldots, l_0, k_0)$$

word size n	key words m	block size $2n$	key size mn	α	β	# rounds T
16	4	32	64	7	2	22
24	3	48	72	8	3	22
	4		96			23
32	3	64	96	8	3	26
	4		128			27
48	2	96	96	8	3	28
	3		144			29
64	2	128	128	8	3	32
	3		192			33
	4		256			34

Table 1: Parameters for SPECK

The first round key k_0 is thus the least significant word of K. Subsequent keys are calculated as follows:

$$l_{i+m-1} = (S^{-\alpha}l_i + k_i) \oplus i$$
$$k_{i+1} = S^{\beta}k_i \oplus l_{i+m-1}$$

This is equivalent to applying the round function with round key i:

$$(l_{i+m-1}, k_{i+1}) = R_i(l_i, k_i)$$

2 High-level design

Apart from the fact that it uses the same hardware, another important aspect of the key schedule is that it can be executed "on the fly". There is no need to generate all round keys before the start of encryption. During each round of encryption, we can calculate the next round key in parallel. As a result, we only need to keep the current key k_i in memory and don't need to remember the complete array of round keys. We do however need a register for the array l.

The decision of performing the key schedule in parallel with encryption actually depends enormously on the intended application. If the same key is used for a long 'conversation', then this scheme reduces the efficiency of encryption too much. In that case, it is better to generate all round keys once and save them to a register to reduce 'on-line' computation complexity. If on the other hand the cryptoprocessor is used for one-time messages with different keys, the current scheme is better.

An essential property of a lightweight cipher is its low-area design. We will therefore opt for a folded pipeline rather than a linear pipeline. For many lightweight applications, throughput is not the top priority. The pipeline will contain the hardware for two round functions such that we can perform encryption and key schedule in parallel. In addition, we will need a register for the array l, inputFIFO's for the plaintext and key and and outputFIFO for the ciphertext.
The implementation will use polymorphism to allow for different choices of parameters and we will choose one or two specific sets for synthesis on the FPGA.
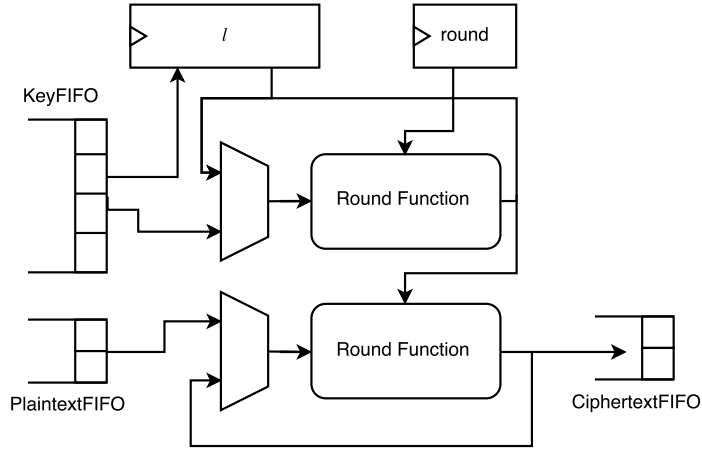
Figure 2: High level design

3 Why embedded cryptography?

Why is it useful to create cryptographic hardware instead of using software implementations? Firstly and most obviously, a dedicated hardware implementation achieves better performance. More importantly, a separate cryptography processor creates an explicit physical barrier around sensitive information. Cryptographic software performed in an environment with other applications running concurrently can leak information through for example timing variations or the cache. It is better for secret keys to be handled in their own environment with a private cache. Furthermore, many devices in the Internet of Things have limited resources or even no generalized core at all, in which case cryptography in hardware is the only option.

4 Test Plan

We have a working C implementation and a set of test vectors to check for correctness from the original paper [1]. Since the paper only provides one test for each parameter set, we can use the reference implementation to generate more test vectors for the parameter sets that we will use on the actual FPGA.
We will use a testing infrastructure similar to that of the Audio Pipeline, using Sce-Mi. A software test bench in c++ will feed the test vectors to port proxies and compare the received results with our reference ciphertexts. We will wrap our SPECK implementation in the `SceMiLayer` to connect it to the Sce-Mi ports.
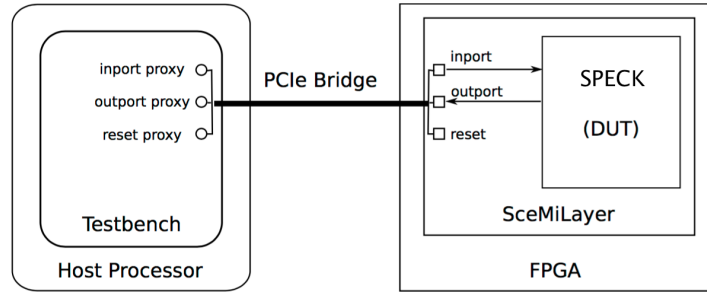
Figure 3: Test infrastructure with Sce-Mi

References

[1] R. Beaulieu et al. The simon and speck families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. `http://eprint.iacr.org/`.