KLEIN cipher, optimised for high speed

Crypto Engineering – Assignment 1

Koen van Ingen, s4058038
Joost Rijneveld, s4048911

March 2014

In this report, we will describe our implementation of the KLEIN-64 cipher (optimised for high speed) on the AVR ATTiny 45 microcontroller.

1 The KLEIN cipher

The KLEIN cipher is a lightweight block cipher that shows many similarities with the Rijndael (AES) cipher. KLEIN consists of five operations: AddRoundKey, SubNibbles, RotateNibbles, MixNibbles and KeySchedule. For these operations, the main difference with AES is the fact that KLEIN operates on nibbles rather than bytes. We will now briefly describe these operations.

AddRoundKey This operation consists of xoring the round key onto the state.

SubNibbles In this step, the state is passed through the sbox, one nibble at a time.

RotateNibbles This operation rotates the state by four nibbles (or two bytes).

MixNibbles This step consists of a series of shifts and polynomial field multiplications that can be summarised in two tables. It operates on concatenations of nibbles, effectively mimicking the AES MixColumns operation.

KeySchedule This operation computes the next round key by performing various shifts, sbox lookups and xor operations.

2 Our implementation

The ATTiny45 is an 8-bit processor that has 32 registers, 512 bytes SRAM and 4096 bytes Flash memory. As it uses 16-bit addresses, addressing memory requires two 8-bit registers. It is worth mentioning that only register 30 and 31 can be used to address Flash memory.

During the execution of our program we keep the round key in register 0 to 7, while the state is alternately in register 8 to 15 and register 16 to 23. The initial key and plaintext are loaded using `ldi` instructions, and the resulting ciphertext ends up in register 16 to 23.

2.1 Major optimisations

2.1.1 Squaring the sbox

The sbox as presented in the KLEIN paper operates on nibbles, but our state is stored in 8-bit registers. Additionally, loading from memory always returns a full byte of data. As all nibbles are passed through the sbox during the SubNibbles step, we can also pass them pairwise as bytes. This requires an sbox that operates on bytes, which can be constructed by squaring the original 8 byte sbox to 512 bytes. By storing this sbox at `0x0500`, we can conveniently perform lookups by copying a state register to the lower half of the address.

2.1.2 Rotating by renaming

At various stages in a round, several registers are rotated. Naively this would result in several `mov` instructions, but all of these can be prevented by implicitly renaming registers. It requires some extra administration, although this can be largely simplified by combining rotation with loading to and from memory. The fact that all our `mov` operations move to register 30 are a testament to this.

2.1.3 Lookup tables for MixNibbles

The MixNibbles step requires matrix multiplication of

2.2 Minor optimisations

3 Results