Small and High-Speed Hardware Architectures for the 3GPP Standard Cipher KASUMI

Akashi Satoh and Sumio Morioka

IBM Research, Tokyo Research Laboratory, IBM Japan Ltd., 1623-14,
Shimotsuruma, Yamato-shi, Kanagawa 242-8502, Japan
{akashi, e02716}@jp.ibm.com

Abstract. The KASUMI block cipher and the confidentiality (*f*8) and integrity (*f*9) algorithms using KASUMI in feed back cipher modes have been standardized by the 3GPP. We designed compact and high-speed implementations and then compared several prototypes to existing designs in ASICs and FPGAs. Making good use of the nested structure of KASUMI, a lot of function blocks are shared and reused. The data paths of the *f*8 and *f*9 algorithms are merged using only one 64-bit selector. An extremely small size of 3.07 Kgates with a 288 Mbps throughput is obtained for a KASUMI core using a 0.13-μm CMOS standard cell library. Even simultaneously supporting both the *f*8 and *f*9 algorithms, the same throughput is achieved with 4.89 Kgates. The fastest design supporting the two algorithms achieves 1.6 Gbps with 8.27 Kgates.

1 Introduction

A 64-bit block cipher KASUMI [1-4] was developed based on MISTY [5] for the 3GPP (3rd Generation Partnership Project) standard algorithm used in the WCDMA (Wideband Code Division Multiple Access) cellular phone systems. KASUMI has an 8-round Feistel structure with nested round functions, and is suitable for small hardware implementations. A high-speed KASUMI hardware design that has eight round function blocks was reported in [6], and a throughput of 5.78 Gbps with 47.66 Kgates was obtained in pipelined operation. However, the pipelined operation cannot be applied to the confidentiality algorithm *f*8 and the integrity algorithm *f*9 where KASUMI is used in feedback modes.

In this paper, we propose three compact but still high-speed hardware architectures, and implement them using an ASIC library and FPGAs. A performance comparison between a conventional implementation [6] and ours is also done using the same FPGA platform.

2 KASUMI Algorithm

2.1 Round Functions

KASUMI has an 8-round Feistel network, and encrypts 64-bit data using a 128-bit key. Fig. 1 shows the nested structure of the KASUMI data path excluding the key

A.H. Chan and V. Gligor (Eds.): ISC 2002, LNCS 2433, pp. 48–62, 2002.

scheduler. The network has a linear 32-bit function FL and a nonlinear 32-bit function FO as the main round functions. The FO function consists of a 3-round network with a 16-bit nonlinear function FI. The FI function consists of a 4-round network with two S-Boxes, S9 and S7. In the odd-numbered rounds of the 8-round main network, 64-bit data is divided into two 32-bit blocks, and the left block is transformed by FL followed by FO, and then the FO output is XORed with the right block. In the even-numbered rounds, the order of the functions is swapped with FO followed by FL. At the end of each round, the left and right 32-bit blocks are also swapped.

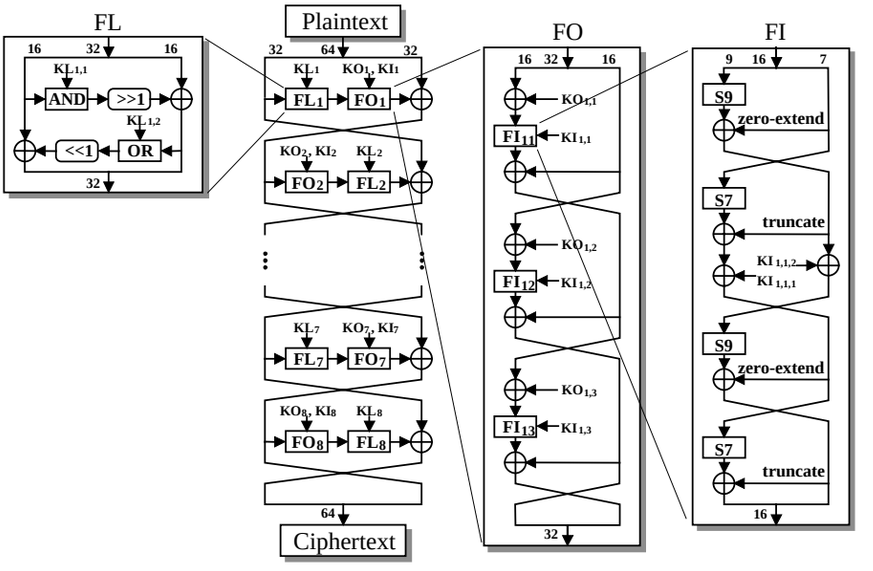

Fig. 1. KASUMI encryption data path

The FL function transforms the 32-bit data with two 16-bit sub-keys $KL_{i,1}$ and $KL_{i,2}$, using AND, OR, XOR, and 1-bit cyclic shift operations. The FO function divides the 32-bit input data into two 16-bit blocks, and then the left block is XORed with the 16-bit sub-key $KO_{i,j}$, transformed by the FI function with a 16-bit sub-key $KI_{i,j}$, and XORed with the right block. This routine is iterated three times with swaps of the left and right blocks.

A 16-bit data block entering the FI function is divided into two smaller blocks for S-Box transformations. The leftmost 9 bits become one block, and the rightmost 7 bits become another block, and then they are transformed twice using the 9-bit S-box S9 and the 7-bit S-box S7 respectively. These S-boxes are defined as AND-XOR matrix operation. The two data blocks are XORed with each other, but the bit length is different, so zero-extension is done to the 7-bit blocks by adding two '0's, and the two most significant bits of the 9-bit blocks are truncated. In the middle of the 4-round network, an XOR operation is done with the 16-bit sub-key $KI_{i,j}$ (where $KI_{i,j,1}$ is the upper 9 bits and $KI_{i,j,2}$ is the lower 7 bits).