Information Processing Letters 111 (2011) 512–515

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

On the cryptanalysis of the hash function Fugue: Partitioning and inside-out distinguishers

Jean-Philippe Aumasson [a], Raphael C.-W. Phan [b,*]

[a] Nagravision SA, route de Genève 22, 1033 Cheseaux, Switzerland
[b] Electronic & Electrical Engineering, Loughborough University, UK

ARTICLE INFO

Article history:
Received 12 October 2010
Accepted 23 February 2011
Available online 1 March 2011
Communicated by D. Pointcheval

Keywords:
Cryptography
Hash functions
Cryptanalysis
Fugue

ABSTRACT

Fugue is an intriguing hash function design with a novel shift-register based compression structure and has formal security proofs e.g. against collision attacks. In this paper, we present an analysis of Fugue's structural properties, and describe our strategies to construct distinguishers for Fugue components.

1. Introduction

Fugue is a hash function designed by Halevi, Hall and Jutla of IBM Research, and is one of the 14 candidates in the second round of the US National Institute of Standards & Technology (NIST) public competition [1] to select a new cryptographic hash standard (SHA-3), in the same manner as the encryption standard AES. Among these 14, Fugue has one of the least successful third-party analysis published.[1]

Fugue follows a novel type of design, which allows for formal security arguments against collision attacks and distinguishing attacks on a dedicated PRF mode [2]. However, no formal argument is given in favor of its "random" behavior when the function is unkeyed, as in many hash function applications.

Fugue-256 (the version of Fugue with 256-bit digests) updates a state S of 30 words of 32 bits using a transform **R** parametrized by a 32-bit message block. **R** essentially consists of two AES-like transforms called **SMIX** applied to 128-bit windows of S, and thus can be easily distinguished from a random transform. As Fugue adopts a "little at a time" strategy wherein small message blocks are processed with a cryptographically weak function, it needs strengthening via a stronger output function.

To achieve pseudorandomness Fugue-256 relies on a much stronger transform than **R**, called the *final round* **G**, computed after all message blocks are processed through the **R** transform. **G** returns a 256-bit digest from the 960-bit state by making 18 rounds involving 36 calls to **SMIX** (this versus just two calls to **SMIX** in **R**). Unlike **R**, **G** does not have trivial distinguishers.

Our preliminary results on Fugue-256 have been presented informally at the second SHA-3 conference without proceedings. As a sequel, this paper analyzes specific properties of Fugue-256's structure and discusses cryptanalysis strategies that we optimized to construct distinguishers for Fugue-256's output function **G**. Indeed, since **R** is weak by design, the crux of Fugue's security lies in **G**. First, Section 3 presents our cryptanalysis strategies for constructing a distinguisher for **G**'s $G1$ rounds that needs only two *unknown related inputs*. Then, Section 4 discusses the **G** properties and corresponding techniques we used to construct an efficient distinguisher for the full **G** using *chosen inputs*.

* Corresponding author.
E-mail address: r.phan@lboro.ac.uk (R.C.-W. Phan).

[1] See the SHA-3 Zoo wiki: http://ehash.iaik.tugraz.at/wiki/The_SHA-3_Zoo.

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2011.02.012

The latter distinguisher consists in exhibiting tuples of inputs and outputs satisfying some "evasive" property. Paraphrasing [3], such a property is easy to check but impossible to achieve with the same complexity and a non-negligible probability using oracle accesses to an ideal primitive—in the present case, a fixed-input-length random oracle. Such distinguishers are for example relevant to disprove indifferentiability of permutation constructions [4,5], or to invalidate indifferentiability claims of hash constructions [6]. These distinguishers often use an inside-out strategy [7,8] to determine inputs and outputs satisfying an evasive property.

2. Fugue-256

The hash function Fugue-256 processes a message m by appending it with zeros and an 8-byte encoding of its bit length to obtain a chain of 32-bit words denoted $m_0, m_1, \ldots, m_{N-1}$. This is processed by updating the 30-word state $S = (S_0, \ldots, S_{29})$ as $S \leftarrow \mathbf{R}(S, m_i)$ for $i = 0, \ldots, N-1$, where S is initialized to a fixed IV. The output digest is $\mathbf{G}(S)$.

For conciseness, we omit the description of the round transformation **R**, as our cryptanalysis is irrespective of this. The reader is referred to [2] for more details of **R**. It is sufficient to note that one **R** is analogous to two AES rounds. In contrast, **G** is analogous to 36 AES rounds.

The final round **G** transforms the internal state $(S_0, \ldots, S_{29})$ by doing five $G1$ *rounds*:

ROR3; **CMIX**; **SMIX**;

ROR3; **CMIX**; **SMIX**

followed by 13 $G2$ *rounds*:

$S_4 += S_0$; $S_{15} += S_0$; **ROR15**; **SMIX**;

$S_4 += S_0$; $S_{16} += S_0$; **ROR14**; **SMIX**

where **ROR3**, **ROR15** and **ROR14** right-rotate S by 3, 15 and 14 words, respectively, and where $+$ denotes bitwise exclusive-or (XOR). **CMIX** is defined as:

$$S_0 += S_4; \quad S_1 += S_5; \quad S_2 += S_6;$$
$$S_{15} += S_4; \quad S_{16} += S_5; \quad S_{17} += S_6.$$

SMIX bijectively transforms the 128-bit vector (S_0, S_1, S_2, S_3). Inspired by the AES round function, **SMIX** views its input as a 4×4 matrix of bytes. First each byte passes through the AES S-box, then a linear transformation called Super-Mix is applied. Unlike AES' MixColumn, Super-Mix operates on the whole 128-bit state rather than on each column independently, which makes **SMIX** stronger than the original AES round. Super-Mix is the only Fugue component that provides bytewise mixing within word borders, the other Fugue components only provide wordwise mixing. We refer to [2] for a detailed description of Super-Mix.

G finally returns as hash value the eight words

$$S_1,\ S_2,\ S_3,\ (S_4 + S_0),\ (S_{15} + S_0),\ S_{16},\ S_{17},\ S_{18}.$$

G thus makes in total 36 calls to **SMIX**.

Throughout this paper, we will let S_i^j denote the value of S_i at the input of **G**'s round $(j+1) \geqslant 1$. For example, S_5^0 is the word S_5 of the initial state S^0, i.e. to be input to round 1. If the context is clear, we may omit the round index j for simplicity of notation.

3. Partitioning distinguisher for $G1$ rounds

We show how to construct a distinguisher that applies to all the five $G1$ rounds of **G**, and analyze the specific Fugue properties that we exploit. Note that a $G1$ round offers more diffusion than a $G2$ round, i.e. **CMIX** in a $G1$ round involving three XORs diffuses more words than the two XORs in a $G2$ round.

3.1. Analysis of Fugue properties

The first property we exploit is the fact that the 32-bit wordsize of Fugue has a fairly large domain, meaning that more effort is required within the design to ensure that full bitwise diffusion is achieved, i.e. that a change in any bit affects all output bits after some Fugue internal component. What this means is, even if a change in an input word is said to affect all output words, this may not be entirely true at the bit level for all bit positions within the corresponding words. The partitioning distinguisher that we present in this section exploits this.

The second exploited property is that Fugue respects word boundaries and byte boundaries, i.e. the boundaries between words and bytes in any state of **G** do not become misaligned. Although it is common for word based functions to preserve word boundaries, Fugue's preservation of byte boundaries (within words) as well, allows to trace a word down to the granularity of its component bytes. And this is further amplified by the fact that Fugue's operations only perform word-level mixing and no internal mixing within a word except for the Super-Mix operation of **SMIX**.

This means that we can trace the different bytes within a word and observe that they influence bytes at the same byte location within other words. Let a word be decomposed as four bytes $b_0 b_1 b_2 b_3$; if the byte at b_0 affects another word via **ROR3**, **ROR15**, **ROR14**, **CMIX** or the S-box, that effect will also be at byte position b_0 within the latter word. Only Super-Mix performs bytewise mixing but this is still largely traceable since it is based on matrix multiplication with a constant and sparse linear matrix.

3.2. Constructing the distinguisher

We trace the propagation of the input word S_5^0 through **G**, where we denote the bytes of S_5^0 as $B_0 B_1 B_2 B_3$. This S_5^0 propagates with probability one to S_{29}^4. In round 5, **ROR3** moves this to position S_2, which then enters **SMIX**. Let the bytes after the S-box be $b_0 b_1 b_2 b_3$. By inspecting the definition of the matrix **N** of Super-Mix, the corresponding output words $S_0 S_1 S_2 S_3$ of the S-box are composed of the following bytes:

$$f(0)f(1)f(0123)f(3), \quad f(0)f(023)f(\emptyset)f(3),$$
$$f(0123)f(1)f(\emptyset)f(3), \quad f(0)f(1)f(\emptyset)f(0123)$$