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*

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.

Keywords: Hash functions, cryptanalysis, Fugue

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)'s 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 favour 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

Email address: r.phan@lboro.ac.uk (Raphael C.-W. Phan)

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

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, §3 presents our cryptanalysis strategies for constructing a distinguisher for **G**'s $G1$ rounds that needs only two *unknown related inputs*. Then, §4 shows how we construct an efficient distinguisher for the full **G** using *chosen inputs*.

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; the distinguisher in §4 is inspired by that strategy.

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*:

$$\textbf{ROR3}; \textbf{CMIX}; \textbf{SMIX}$$
$$\textbf{ROR3}; \textbf{CMIX}; \textbf{SMIX}$$

followed by 13 $G2$ *rounds*:

$$S_4 += S_0; S_{15} += S_0; \textbf{ROR15}; \textbf{SMIX}$$
$$S_4 += S_0; S_{16} += S_0; \textbf{ROR14}; \textbf{SMIX}$$

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:

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

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 $(j + 1) \geq 1$. 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 $b_0b_1b_2b_3$; 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 $B_0B_1B_2B_3$. 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 $b_0b_1b_2b_3$. By inspecting the definition of the matrix **N** of Super-Mix, the corresponding output words $S_0S_1S_2S_3$ of the S-box are composed of the following bytes:
$f(0)f(1)f(0123)f(3)$, $f(0)f(023)f(\emptyset)f(3)$, $f(0123)f(1)f(\emptyset)f(3)$, $f(0)f(1)f(\emptyset)f(0123)$ where $f(i)$ denotes some undetermined function of the input byte(s) b_i and where $f(\emptyset)$ denotes no influence of any of the input b_i's.

These words then become $S_3S_4S_5S_6$ after **ROR3** and subsequently **CMIX** XORs $S_4S_5S_6$ with $S_0S_1S_2$ (which up to this point has not been influenced by S_5^0 at all), to form the new $S_0S_1S_2$. Thus, $S_0S_1S_2S_3$ entering the second **SMIX** of round 5 are influenced by the the same bytes of S_5^0 that had entered the first **SMIX** of round 5, except the order is shifted, i.e. the $S_0S_1S_2S_3$ inputs to **SMIX** at this point are:
$f(0)f(023)f(\emptyset)f(3)$, $f(0123)f(1)f(\emptyset)f(3)$, $f(0)f(1)f(\emptyset)f(0123)$, $f(0)f(1)f(0123)f(3)$ and after **SMIX** we have $S_0S_1S_2S_3$ of the form:
$f(\bullet)f(\bullet)f(\bullet)f(\bullet)$, $f(013)f(\bullet)f(013)f(\bullet)$, $f(\bullet)f(\bullet)f(\bullet)f(\bullet)$, $f(\bullet)f(\bullet)f(\bullet)f(\bullet)$ where $f(\bullet)$ denotes some undetermined function of all the input bytes $b_0b_1b_2b_3$.

Observe that the first and third bytes of **SMIX** output word at S_1^5 are only influenced by $b_0b_1b_3$ and *not* by b_2. This property has been empirically verified using the Fugue source code submitted to NIST's hash competition.

This property can be exploited as a distinguisher: we obtain just *one* pair of inputs to **G** such that only the byte B_2 of S_5^0 varies while the other bytes are constant, and check that after five rounds the first and third bytes in S_1^5 remain unchanged. Note that this does not require the cryptanalyst to know what the input values to **G** are, nor that s/he be able to choose or know the difference between the inputs (as would differential cryptanalysis). Instead, the cryptanalyst only needs to vary the input byte B_2, e.g. flip one bit. This is akin to a partition on the input space (where only the byte B_2 of S_5^0 is non-constant) that leads to an output partition of colliding values (first and third bytes in S_1^5 are constant) indexed by these two constant byte positions. This distinguisher succeeds with probability one and is sufficient to distinguish all five $G1$ rounds from random by just looking at one output word S_1^5, more specifically in fact two bytes.

This result is surprising since **SMIX** which is seen to be stronger than one AES round, unfortunately is not used optimally within a **G** round although the same bytes of S_5^0 enter **SMIX** twice within round 5 and yet do not fully influence all bytes of the final 128-bit **SMIX** output. In contrast, AES achieves full diffusion on its 128-bit output state, i.e. each output byte is influenced by all input bytes after two rounds.

One of the reasons for this is because the constant matrix **N** used by Super-Mix within **SMIX** is too sparse. Furthermore, if we analyze this bytewise, this sparsity causes some output bytes of Super-Mix to not be influenced by any of the four bytes composing a particular input word S_i.

4. Inside-out distinguisher for full G

We will now show how to construct a distinguisher for the full 18 rounds of **G**, which consists in an efficient method to find pairs of initial states (S, S') that differ in 66 bits on average, and such that $\mathbf{G}(S)$ and $\mathbf{G}(S')$ remain constant for all pairs found. To the best of our knowledge, such a distinguisher type (i.e. where the distinguishing output pattern consists of checking for pairs with a constant difference) is novel. Along the way, we make explicit the particular Fugue properties that we exploited.

We first present probability-one differential characteristics for 15 rounds of **G** which will later be used to construct our full round distinguisher. The Fugue analysis in [2, §12.4.2] considered a difference to be induced in the right half of the input state such that after one $G1$ round, differences exist in the *last* few words. This difference propagation was then traced through *five* $G1$ rounds before $G2$ and it was concluded that if done this way then no differences enter the **SMIX** for most of the subsequent $G2$ rounds. In fact, differentials need not necessarily start from the initial round; it is common to construct differentials for some middle rounds and then exploit this in distinguishers that cover more rounds.

For an arbitrary state S, we consider instead an arbitrary input difference Δ in S_5 (left half of the state) and no difference in the other S_i's. Then after four $G1$ rounds followed by 11 $G2$ rounds (thus 15 rounds in total), the state S always has a difference Δ in S_{18} and no difference in the other S_i's. The strategy here is to keep in mind that **SMIX**, **CMIX** and addition are the only Fugue components that provide diffusion, so the trick is to avoid having a difference enter any of these operations.

4.1. Constructing the distinguisher for 18 **G** *rounds*

To construct the 18-round distinguisher, we take the inside-out strategy, i.e. start from the middle of **G**. This is possible because a hash function as is Fugue is unkeyed, i.e. it is not dependent on any secret, so any component can be computed by the cryptanalyst starting from the middle of **G** moving outwards to both ends of **G**.

More precisely, we start from the output of round 16, i.e. S^{16}. In the forward direction we move through rounds 17∼18. In the other direction, we move through rounds 16∼1 backwards, firstly by applying the 15-round probability-one characteristic to rounds 2∼16 in reverse and then further inverting round 1 to get to the initial state.

4.1.1. Forwards: constant $\mathbf{G}(S)$ *and* $\mathbf{G}(S')$

In more detail, recall that after its last (18th) round, **G** extracts 256 bits from the 960-bit final state by returning $S_1^{18}, S_2^{18}, S_3^{18}, (S_4^{18}+S_0^{18}), (S_{15}^{18}+S_0^{18}), S_{16}^{18}, S_{17}^{18}, S_{18}^{18}$ to form its output. This output digest does *not* depend on the values of $S_7^{16} \sim S_{13}^{16}$ and $S_{21}^{16} \sim S_{28}^{16}$ entering the 17th round.

This exploits the fact that **G** uses truncation on the round 18 output S^{18} to produce the output digest, so the round 18 output words that are truncated out (and thus the corresponding S^{16} words that affect these S^{18} words) do not affect the output digest.

Thus we can fix $S_0^{16} \sim S_6^{16}$, $S_{14}^{16} \sim S_{20}^{16}$, and S_{29}^{16} to ensure that $\mathbf{G}(S)$ and $\mathbf{G}(S')$ remain unchanged while varying the other words (recall that S' is S after setting a difference in S_{18}^{16}). This observation allows us to find the values of distinct pairs of states (S^{16}, S'^{16}) before round 17 such that their respective digests $(\mathbf{G}(S), \mathbf{G}(S'))$ are constant for all pairs found (and thus their difference is constant as well).

4.1.2. Backwards: characteristic and sparse initial differences

Having determined a pair of states (S^{16}, S'^{16}) entering the 17th round with a difference Δ in S_{18}, it is guaranteed that after inverting 11 $G2$ rounds and then four $G1$ rounds backwards to the state after round 1, one obtains a pair of states (S^1, S'^1) with only a difference Δ in their S_5 word, as they follow the 15-round characteristic backwards with probability one. Since there is only one $G1$ round left to invert in order to get to the initial state, most of the state is unaffected by Δ: it is easy to show that the corresponding pair of initial states (S^0, S'^0) will have difference Δ in S_{10} and S_{25}, caused by the first inverse **CMIX**, and undetermined differences in $S_0, S_{27}, S_{28}, S_{29}$, caused by the second inverse **SMIX**.

Therefore, only six words of the initial state have nonzero differences, and if Δ is chosen of minimal Hamming weight (i.e. 1), then the two states have on average 66 bit differences (assuming that the inverse **SMIX** causes in average differences in half the bits of its input; this seems a reasonable assumption, as the linear layer of the inverse **SMIX** has a much denser algebraic description than the original, as shown in [2, Fig. 5]). Our experiments did not contradict that estimate.

5. Final Remarks

Summarizing, the fact that word and byte boundaries are preserved, no internal mixing within a word, except for Super-Mix and the fact that the diffusion matrix $\mathbf{N}$ is very sparse facilitate the task of tracing bytes within words. The fact that there is heavy truncation of intermediate state to form output digest allows to vary the omitted state words to obtain (via inverting) the different input states that would preserve the output digest values.

[1] NIST, Cryptographic Hash Competition, `http://www.nist.gov/hash-competition`, accessed October 2010.

[2] S. Halevi, W. E. Hall, C. S. Jutla, The Hash Function Fugue, Submission to NIST (updated), 2009.

[3] H. Gilbert, T. Peyrin, Super-Sbox Cryptanalysis: Improved Attacks for AES-like permutations, Cryptology ePrint Archive, Report 2009/531, 2009.

[4] Y. Dodis, P. Puniya, On the Relation Between the Ideal Cipher and the Random Oracle Models, in: TCC, LNCS, Springer, 184–206, 2006.

[5] J.-S. Coron, J. Patarin, Y. Seurin, The Random Oracle Model and the Ideal Cipher Model are Equivalent, Cryptology ePrint Archive, Report 2008/046, full version of [9], 2008.

[6] J.-P. Aumasson, W. Meier, Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi, NIST mailing list, 2009.

[7] A. Biryukov, D. Wagner, Slide Attacks, in: FSE, LNCS, Springer, 245–259, 1999.

[8] L. R. Knudsen, V. Rijmen, Known-Key Distinguishers for Some Block Ciphers, in: ASIACRYPT, LNCS, Springer, 315–324, 2007.

[9] J.-S. Coron, J. Patarin, Y. Seurin, The Random Oracle Model and the Ideal Cipher Model are Equivalent, in: CRYPTO, LNCS, Springer, 1–20, 2008.