From: hash-forum@nist.gov on behalf of Chang, Shu-jen H. [shu-jen.chang@nist.gov]
Sent: Thursday, October 15, 2009 11:52 AM
To: Multiple recipients of list
Subject: FW: OFFICIAL COMMENT: Fugue (Round 2)

FYI

The official comment shown below was not sent to hash-forum. I'd like to inform you that we have received an update package of Fugue and have posted it on the NIST site as an **update**. We don't plan to accept future updates unless for a very good reason. Submitters are advised to send OFFICIAL COMMENT and post corrections on their web site.

Regards,
Shu-jen

From: Charanjit Jutla [mailto:csjutla@us.ibm.com]
Sent: Friday, October 02, 2009 4:50 PM
To: hash-function@nist.gov
Subject: OFFICIAL COMMENT: Fugue (Round 2)

Dear all,

1. Stefan Tillich (of Graz University of Technology) has found a bug in the __implementation__ of Fugue's padding. In particular, for messages which are not of byte-multiple length, the implementation erroneously zeroes the last incomplete byte. The bug is in all implementations, including reference, and the way the KAT files were generated. As a result, the KATs for non-byte-confirming messages are wrong.

2. We will correct the implementation, and post new KAT files and the implementations on the IBM site next week.
The link will be posted here again. We will see how/when/if it can be incorporated into NIST's posted version.

3. To emphasize, the bug is NOT in the Fugue padding specification, but in the implementation. The specification remains unchanged.

4. The implementations submitted to eBASH remain unchanged, as they are for byte length messages only.

5. We appreciate the fact that Stefan contacted us first, rather than posting directly on the forum.

6. For those who are curious, the bug is not in Fugue.c, but in the wrapper file SHA3api_ref.c. Line 62 reads:

```
memset ((uint8*)state->Partial+((state->TotalBits&31)/8), 0, need/8);
```

It should instead be:

```
memset ((uint8*)state->Partial+(((state->TotalBits&31)+7)/8), 0, need/8);
```

7. Thanks for your patience,

The Fugue Team

From: Charanjit Jutla [csjutla@us.ibm.com]

Sent: Thursday, August 12, 2010 5:07 PM

To: ha sh-function@nist.gov

Subject: OFFICIAL COMMENT: Fugue (Round 2)

Dear all,

Jean-Philippe Aumasson has been kind enough to point out a confusing statement about the diffusion tables in the Fugue document, and we report it here.

In Section 4.3, "Diffusion in the TIX-less rounds G1", the text correctly states what the corresponding table 6 on diffusion properties of final round stage G1 shows. However, in section 11 on External collisions this information is worded wrongly. This has no impact on the proof in that section, except that the wording can be confusing.

Specifically, section 4.3 para two states, and I quote
"Starting right after the last TIX step and ending after the end of G1, the table shows how many initial bytes influence each of the final bytes".

Here the "last TIX step" is from the last input round, and this is where the adversary may supply input to that round. After this TIX step, there are two SMIX rounds. So, the final round effectively starts right after the last TIX step.

BTW, these diffusion tables and round G1 are just acting as tolerance buffers, and the proof of improbability of collisions is really about final round stage G2, which starts after G1.

Thanks,

Fugue Team

From: Charanjit Jutla [csjutla@us.ibm.com]
Sent: Tuesday, August 24, 2010 2:31 AM
To: ha   sh-function@nist.gov
Cc: h   ash-forum@nist.gov
Subject: OFFICIAL COMMENT: Fugue (Round 2)

Dear all,

We have written a note
"Weak Ideal Functionalities for Designing Random Oracles
with Applications to Fugue".

The note is attached. It will be expanded to a full paper shortly,
when we will post it here again.

Thanks,

Charanjit

Weak Ideal Functionalities for Designing Random Oracles with Applications to Fugue

Shai Halevi William E. Hall Charanjit S. Jutla Arnab Roy
IBM T.J. Watson Research Center

August 24, 2010

1 Introduction

Fugue is a variable input length (VIL) hash function which maps arbitrary length input messages to a fixed length output, e.g. 256-bit. As opposed to traditional designs Fugue has “light-weight” input rounds, which cannot be claimed to be random permutations or ideal ciphers. This approach was first embodied in the hash function Grindahl.

The main idea behind this design approach is that hash functions (as opposed to block ciphers) do not have an inversion oracle; or in other words, in the security definition of a hash function the adversary does not have access to an inversion oracle, and hence it can be securely realized by components which themselves do not have any inversion oracles. If one tries to build hash functions using components which are required to be secure against inversion oracles, one may not get an optimal speed/security tradeoff. Fugue capitalizes on this observation, along with many other such practical issues regarding full security of a hash function, and focuses instead on proving the full hash function to be resistant to known cryptanalytic techniques like differential attacks, and linear cryptanalysis. For collision resistance, Fugue is actually proven to be resistant to differential attacks under very reasonable assumptions (rather than very ideal assumptions).

In this paper, we formalize this intuition, and show that one can build variable input length (VIL) hash functions, which are indifferentiable [1] from VIL Random Oracles (VIL-RO), from ideal functionalities which do not have inversion oracles. Currently, this ideal functionality is still a VIL functionality, but in a forthcoming work, we will show how to do this with a fixed input length (FIL) ideal functionality. Normally, this would not be a difficult task, but we also want to make these ideal functionalities *as weak as possible*, so that one can actually argue that Fugue's components (or some other hash function) actually realize these ideal properties. Thus, we will also argue that given the concrete proofs of security already given for Fugue (e.g. collision resistance to differential attacks), one can say with high confidence that Fugue's components realize these idealize functionalities.

So, coming back to the formalization, one may wonder that since it is obvious that from a VIL collision resistant functionality and a FIL-RO one can realize a VIL-RO, then what is new to prove here. However, given the focus of Fugue to be not un-necessarily trying to defend against internal properties which are not needed in the full realization of a secure VIL-RO, one needs to take a different approach to prove Fugue to be VIL-RO. For example, no adversary has access to the final transformation by itself; the only access it has is via the input rounds, i.e. the only input that can be supplied to the final transformation is what the adversary can produce from the input rounds. To capture this intuition, one does not define two different ideal functionalities, but just one ideal functionality with its public interface having two functions one for the input rounds, and one for the final transformation. But, they have a shared storage.

The next step is to let the two functions not necessarily split into just input rounds and final transformation, but the final transformation may be required to have a few input rounds as well (unless the input itself is tiny).

So, the overall approach in the rest of the paper is going to be as follows.

1. First an ideal functionality $\mathcal{I}$ is defined.

2. Next, in the $\mathcal{I}$-hybrid model a real-world realization called *Hash* of VIL-RO is given, i.e. *Hash* employing $\mathcal{I}$ is shown indifferentiable [2] from VIL-RO.

3. A functionality Fugue-$\mathcal{I}$ is defined which has the same interface as $\mathcal{I}$ but which is implemented using Fugue's components, e.g. Fugue's input round R and final transformation G. It is shown that *Hash* using Fugue-$\mathcal{I}$ is exactly the same as actual full Fugue.

4. It is argued using already known proofs about Fugue's collision resistance (and partial collision resistance) that it is extremely realistic to assume that Fugue-$\mathcal{I}$ realizes $\mathcal{I}$. This is to be contrasted with the approach most hash function designs take where they assume that their component realizes a strong ideal functionality such as ideal cipher or random permutation based only on heuristic bounds/analysis of differential attacks.

2 Ideal Functionality $\mathcal{I}$ for Fugue-like designs

.

The ideal functionality $\mathcal{I}$ has two public functions: $\mathcal{I}-$prefix and $\mathcal{I}-$final. It also has two tables PSTORE and CSTORE. The store PSTORE has entries of the kind $\langle m, |m|, t\rangle$, where t is supposedly a 960-bit internal state, and m is supposedly a prefix of a message. The store CSTORE has entries of the kind $\langle t, \text{prefix-len}, m, r\rangle$, where as before t is a 960-bit internal state supposedly obtained by running the input rounds on a prefix of a message of prefix length "prefix-len", and when this state is continued on with a suffix m, the final 256-bit returned is supposedly r.

Some of the arguments to $\mathcal{I}-$prefix and $\mathcal{I}-$final are optional, and when not supplied are represented by '*'.

Here is the definition of $\mathcal{I}-$prefix. It takes two arguments, (i) a message m of arbitrary length (in words) but say, bounded by 2^{128}, and (ii) a value t-fake for the intermediate state, say 960-bits.

On input m and t-fake, $\mathcal{I}-$prefix computes and returns the following:

- If m is in PSTORE, then return the intermediate state associated with m. Else,
  - If t-fake is already the intermediate state of some entry in PSTORE, then return $\perp$. Else,
  - Let t be t-fake. if t-fake is not supplied, let t be a random 960-bit value. If t is already the intermediate state of some entry in PSTORE, then abort, and return $\perp$. Else, insert entry $\langle m, |m|, t\rangle$ in PSTORE. Return t.

Here is the definition of $\mathcal{I}-$final. It takes four arguments, (i) a message m of length 8 or fewer words, purportedly the suffix of the actual full message, (ii) a prefix-len, supposedly the length of the prefix of the actual full message, (iii) a 960-bit intermediate state t, and (iv) a 256-bit value r-fake.

Here is how $\mathcal{I}-$final computes. There are two main cases: if the prefix-len is zero and if the prefix-len is not zero. So, lets first see how $\mathcal{I}-$final computes if prefix-len is *zero*.

- If $|m| \geq 8$, return $\perp$. Else,
  - If $\langle 0, 0, m, r\rangle$ is in CSTORE for some value r, then return r. Else,
  - Set r to a random 256-bit value. Insert $\langle 0, 0, m, r\rangle$ in CSTORE. Return r.

Now, let's see how $\mathcal{I}-$final computes if prefix-len is *not-zero*.

- If $|m| \neq 8$, return $\perp$. Else,
  - If $\langle p, |p|, t\rangle$ is in PSTORE for some message p, then
    - if $\langle t, \text{prefix-len}, m, r\rangle$ is in CSTORE for some r, then return r, else
    - choose a 256-bit r at random, insert $\langle t, \text{prefix-len}, m, r\rangle$ in CSTORE, and return r.
  - Else, insert $\langle t, \text{prefix-len}, m, r\text{-fake}\rangle$ in CSTORE.

3 Realizing VIL-RO using $\mathcal{I}$

For any message P of arbitrary bit length, let Hash(P) be computed as follows. First let, m be defined by first extending P with zero bits to a word boundary, and then appending it with two words of length of P in bits. Next,

- If $|m| < 8$, return $\mathcal{I}-$final(m,0,*,*). Else,
- let $m = m'||m''$, where $|m''| == 8$. Let t= $\mathcal{I}-$prefix(m', *). If $t \neq \perp$, return $\mathcal{I}-$final(m'', $|m'|$, t, $*$), else return $\perp$.

The ideal functionality VIL-RO is straightforward. It has a public interface with one function RO(m), for any message m of arbitrary length. VIL-RO has a store, in which it keeps pairs of the kind $\langle m, r\rangle$, where m are arbitrary length messages and r are 256-bit values. On any query RO(m), it first checks if m is already in its store, and if so, it just returns the associated r. Otherwise, it generates a fresh random 256-bit value r, inserts $\langle m, r\rangle$ in the store and returns r.

Theorem 3.1 *The function Hash above using ideal functionality $\mathcal{I}$ is indifferentiable from VIL-RO.*

Proof: We will show that there exists a simulator S, such that no adversary making q calls can distinguish between $\text{Hash}^{\mathcal{I}}$ and S[VIL-RO], with probability greater than $2^{-960} \cdot q^2$. The simulator S is actually a dummy, and just passes the values directly to VIL-RO.

If the adversary calls the two scenarios with a message P of length less than 6 words, then Hash calls $\mathcal{I}-$final$(m, 0, *, *)$, where m is P appended with two words of count. Note that $\mathcal{I}-$final returns a fresh random value (unless m was called before). Similarly, in this case, S just passes through the input and output, and the returned value is random.

If the adversary calls the two scenarios with a message P of length greater than or equal to 6, note that the padding scheme of Hash would result in a m of length 8 or greater. In this case $m = m'||m''$. First, $\mathcal{I}-$prefix is called with m' and *, which results in $\langle m', |m'|, t\rangle$ value being stored in PSTORE, where t is a random 960-bit value. There is a probability of $q^2 \cdot 2^{-960}$, that $\perp$ was returned, as when the random t collides with an earlier t. If $\perp$ is not returned, then $\mathcal{I}-$final is called with m'' and t, which then returns a 256-bit random value r. Thus, the only discrepancy comes from t colliding. ■

4 Defining functionality Fugue-$\mathcal{I}$ using Fugue's components

First, let's define Fugue-$\mathcal{I}-$prefix, which takes two arguments m and t-fake. Set the initial 960-bit state to the IV prescribed by Fugue. Iterate Fugue's input round R on m, word by word. Return the resulting 960-bit internal state. The argument t-fake is ignored.

Now, let's see how Fugue-$\mathcal{I}-$final is defined, which takes four arguments: m, prefix-len, 960-bit s, and 256-bit r-fake. Again, r-fake is ignored.

- If prefix-len is zero, and $|m| \geq 8$, then return $\perp$.
  If prefix-len is zero, and $|m| < 8$, then set s to the prescribed Fugue IV. Iterate Fugue's input round R on state s with input m, and then apply the final Fugue transformation G to the resulting state, and return the 256-bit output.
- Else, if prefix-len is not zero, set s to the prescribed IV, and iterate Fugue's input round R on input m (word by word), and return the resulting state.

It is easy to see that Hash defined in the previous section when using Fugue-$\mathcal{I}$ computes exactly the same function as Fugue-256.

5 Arguing Fugue-$\mathcal{I}$ is indifferentiable from $\mathcal{I}$

In the indifferentiability paradigm [1], an adversary (or Environment) is given access to two public functionalities, an ideal public functionality, say $\mathcal{I}$, and a real-world public functionality, say Fugue-$\mathcal{I}$. The ideal functionality $\mathcal{I}$ is indifferentiable from Fugue-$\mathcal{I}$ if there is a simulator S such that any adversary A cannot differentiate between Fugue-$\mathcal{I}$ and S[$\mathcal{I}$] (i.e. S sits between adversary and $\mathcal{I}$). We will conjecture that it takes time close to 2^{128} Fugue-256 evaluations to differentiate between the two with high probability.

The simulator S works as follows. It saves the history of all the calls the adversary makes. On calls of the kind $\mathcal{I}-$prefix (and Fugue-$\mathcal{I}-$prefix), the simulator actually calls real Fugue, and whatever real Fugue returns, it passes that as t-fake. Thus, the behavior will be same on both sides. Next, on calls to $\mathcal{I}-$final (and Fugue-$\mathcal{I}-$final), if the Simulator determines that the call is with a state s which was not legitimate, i.e. not returned earlier by some call to $\mathcal{I}-$prefix, it calls real Fugue with that internal state and message to get a 256-bit value, and it then passes that as r-fake. Thus again, the simulation is perfect.

Thus, there are are two ways that the adversary may notice a difference

- when the functionalities return $\perp$,
- when $\mathcal{I}$ returns random and independent values.

When for "syntactic" reasons the functionalities return $\perp$, the simulation is perfect, so the only difference could be when in $\mathcal{I}-$prefix the functionality $\mathcal{I}$ determines that t-fake is already the intermediate state of some entry in PSTORE, and it returns $\perp$. This can happen if the Simulator S called real Fugue, and it returned a collision (i.e. same intermediate internal state) for two different messages. However, in the Fugue document it is proven that internal collisions are improbable to achieve with differential attacks (i.e. prob $< 2^{-128}$), and hence it is a *reasonable assumption*, that no internal collisions can be obtained by adversary in time less than 2^{128}.

As for returning random and independent values, the only places that $\mathcal{I}$ returns random 256-bit values are when

- prefix-len not zero, and $|m| == 8$, and $\langle p, |p|, t\rangle$ is in PSTORE for some message prefix p and the supplied internal state t,
- prefix-len is zero, and $|m| < 8$.

For the second case, the claim is that the output for all messages of length less than 8, and starting from the Fugue IV, are random and independent of all other outputs. For there to be any dependence, these short messages either have to obtain a collision or a partial collision after one round of G1. However, it is shown that both these cases are improbable with differential attacks, and hence it is reasonable to assume that the outputs are random and independent.

For the first case, since we already assumed there are no internal collisions, the further claim is that if Fugue actually returned an internal state t on some prefix message p starting from a valid Fugue IV, then for all m of length 8 extending this message p (and state) the returned value is random and independent.

Now, if we have two prefixes (of messages) p_1 and p_2 with resulting internal states t_1 and t_2, ($D = t_1 \oplus t_2 = 0$), then the Fugue document proves that even partial collisions within the final round G1 are improbable, and hence it is reasonable to assume that final round G2 leads to random and independent behaviour.

If p_1 and state t is itself extended by two different m_1 and m_2, then this case is similar to starting from initial IV, hence again it is reasonable to assume that the output values are random and independent.

References

[1] U. Maurer, R. Renner, and C. Holenstein. Indifferentiability, impossibility results on reduction, and applications to the random oracle methodology. Theory of Cryptography, TCC 2004, LNCS 2951.

[2] J. Coron, Y. Dodis, C. Malinaud and P. Puniya. Merkle-Damgard revisited: How to construct a hash function. Advances in Cryptography, Crypto 2005, LNCS 3621.

From: Charanjit Jutla [csjutla@us.ibm.com]
Sent: Monday, September 20, 2010 11:02 PM
To: ha sh-function@nist.gov
Cc: h ash-forum@nist.gov
Subject: OFFICIAL COMMENT: Fugue (Round 2) -- in response to John Kelsey's ECRYPT presentation

This was mailed to John Kelsey (regarding his talk at ECRYPT, and specifically related to Fugue).

Dear John,

1.
We would like to point out that Fugue is **directly** defined to be a variable input length PRF (and hence a MAC). This is in reference to slide 11, where you mention some other functions with additional properties.

This definition of Fugue PRF is given in section 4.5 titled
"Pseudo Random Function PR-Fugue-256", and it obviates the need to use HMAC.

Further, in section 4.7 we show that Fugue can be used as a direct drop in replacement for SHA-256 in many other modes including randomized hashing and HMAC.

2. Speed.

The slides under-represent Fugue in S/W as well as ASIC speed, especially in comparison to other AES S-Box based designs (particularly Shasvite, Grostl, ECHO -- see below for more details). (This is mostly the case for the 256 bit version, and for 512-bit versions we do not argue that much about speed, as long as it is reasonably good.)
Also, on x86 and many other architectures, our speed handily beats many other functions as well. ANSI-C implementations bring out the best and the worst in some of the candidates. See ebash one to one shootout at the following link
http://bench.cr.yp.to/xweb-hash/long-fugue256.html

If one looks at these four 256-bit designs (i.e. Shavite, Grost, ECHO and Fugue), it can be seen that Fugue-256 requires 2 SMIXES per input word (4 bytes), which is equivalent to 2 AES rounds + 2 additional linear mixings.

Whereas all other designs are at least 4 AES rounds per input word (see below for specifics). Since the additional mixing in Fugue is generally cheaper than an AES round (it becomes 2 times cheaper by SSE4), Fugue-256 is faster than all of these.

This is clearly reflected in most results on x86-32 and x86-64, as well as micro-controllers, including ARM. Please see eBash and the micro-controller talks from 2nd SHA-3 conference. Fugue also beats them in pure ANSI-C code on all x86-32 and x86-64 architectures.

The only places where these designs beat Fugue is if (1) there is an AES instruction, (2) older Intel architectures, where the designers have submitted dedicated assembly code... we are pretty confident that those highly optimized assembly code can be copied to Fugue to achieve better results.

As for the AES instruction, we mentioned at the 2nd SHA-3 conference that Fugue just needs a 128 bit (16 byte) to 128 bit (16 byte) linear mixing instruction to get to 3 to 4 cycles per byte speed. This should be relatively easy for Intel to implement given that they have implemented 6 instructions for AES.

We just need one more instruction, and that too in the usual SSE format, i.e. 16 bytes to 16 bytes. It mostly requires shift and shuffle operations, and should be relatively easy for Intel, given that they already have micro-code for such instructions.

"More specific performance numbers for other AES -based designs showing better performance of Fugue as mentioned above":

ECHO-256 requires 16/3 AES rounds per input word (4 bytes).

SHAVITE-256 : 4 AES rounds per input word (4 bytes)... this includes 3 AES per word in block cipher, and 1 AES per word in message expansion.

Grostl-256 - requires 2*10*64 AES S-Box lookups for 512 bit input,
this is equivalent to 5 AES rounds per input word.

Given these facts, it is safe to say that Fugue should be clubbed with Cubehash and Keccak in your slide 35. If you see the Ebash link mentioned above, we do beat Keccak on most x86 architectures. Moreover, in ANSI-C implementations we handily beat Cubehash, JH and SIMD.

-Fugue Team

From: hash-forum@nist.gov on behalf of Charanjit Jutla [csjutla@us.ibm.com]
Sent: Friday, October 22, 2010 10:21 AM
To: Multiple recipients of list
Subject: OFFICIAL COMMENT: Fugue (Round 2)

Dear All,

You can now access the program used to compute the rank (and minimum weight) of all the linear codes used in Fugue; in particular the ranks of codes used in the proof of differential attack resistance. You may modify the program for further analysis/cryptanalysis work, as long as it remains opensource.

Currently, the program uses Victor Shoup's NTL library for finite field arithmetic, so you will need to install that to use this program.

The link is

http://domino.research.ibm.com/comm/research_projects.nsf/pages/fugue.codes.html

Thanks,

Fugue Team

From: hash-forum@nist.gov on behalf of Charanjit Jutla [csjutla@us.ibm.com]
Sent: Wednesday, December 08, 2010 3:38 PM
To: Multiple recipients of list
Subject: OFFICIAL COMMENT: Fugue (Round 2) : Upcoming Tweaks to Fugue

Dear All,

Here are the tweaks we are planning to propose for the next round (if selected).
Although, we do not expect NIST to consider these tweaks in making their decision,
this should indeed be seen as our confidence in the extra-ordinary security (and efficiency) provided by Fugue.

Over the last one year we have been able to come up with new proofs which drastically enhance the upper bounds on differential attacks on the current embodiment of Fugue. With these proofs, we can now confidently tweak Fugue parameters so that Fugue-256 runs at twice the speed (in hardware and software) for long messages. For messages of size 512 bits the speed will remain the same for Fugue-256. This means we have added 150% more G1 rounds to Fugue-256. This alleviates any concern which Aumasson-Phan (and Gauravaran et al) analysis had, even though we have shown (because of our proofs on partial collisions) that that leads to no attacks. For this reason, in Fugue-224 we do not add many more G1 rounds, and hence Fugue-224 will run faster than before for smaller messages.

The new parameters for different Fugues are as follows:

There are 5 parameters : n,k,r,s,t as specified in the Fugue document.
n: number of words of output
k: number of SMIX per input word
s: internal state size
r: number of G1 rounds...total of rk SMIX
t : number of G2 rounds...total of t * ceil(n/4) SMIX

new Fugue-224
n = 7 , k = 1, s = 30, r = 15, t = 13. (note, rk = 15)
old Fugue-224
n= 7, k = 2, s= 30, r = 5, t= 13 (note, rk = 10)

new Fugue-256
n=8, k = 1, s = 30, r = 26, t= 13 (note, rk = 26)
old Fugue-256
n = 8, k =2, s= 30, r = 5, t = 13 (note, rk = 10)

new Fugue-384
n = 12, k = 2, s = 36, r = 14, t= 13
old Fugue-384
n = 12, k = 3, s = 36, r = 6, t= 13 (note rk = 18)

new Fugue-512

n = 16, k = 3, s = 36, r= 14, t = 13
old Fugue-512
n = 16, k = 4, s = 36, r = 8, t=13 (note rk = 32)

Each of the Fugue's TMIX has also been tweaked. Rest of the specification remains exactly same.
The new TMIXes are as follows:
TMIX (for 224 and 256) : S_4 += S_0; S_0 = Input, S_14 += S_0; S_20 += S_0; S_8 += S_1.

TMIX (for 384)   S_7 += S_0; S_0 = Input; S_10 += S_0; S_14 += S_0; S_4 += S_1.

TMIX (for 512)   S_10 += S_0; S_0= Input; S_7 += S_0; S_11 += S_0; S_4 += S_1; S_22 += S_1.

A new weak Fugue will be specified which will consist of only 20 word state, and drastically shortened final round.

Thanks,

Fugue Team

From: hash-forum@nist.gov on behalf of Praveen Gauravaram [p.gauravaram@mat.dtu.dk]
Sent: Tuesday, December 14, 2010 2:59 PM
To: Multiple recipients of list
Subject: RE: OFFICIAL COMMENT: Fugue (Round 2) : Upcoming Tweaks to Fugue

Attachments: Fugue-analysis-NIST.pdf

Dear all,

Here we submit improved analytical results on the Fugue-256 and its weaker version as submitted for the second round of the competition.

Best regards,
Praveen

From: hash-forum@nist.gov [hash-forum@nist.gov] On Behalf Of Charanjit Jutla [csjutla@us.ibm.com]
Sent: Wednesday, December 08, 2010 21:37
To: Multiple recipients of list
Subject: OFFICIAL COMMENT: Fugue (Round 2) : Upcoming Tweaks to Fugue

Dear All,

Here are the tweaks we are planning to propose for the next round (if selected). Although, we do not expect NIST to consider these tweaks in making their decision, this should indeed be seen as our confidence in the extra-ordinary security (and efficiency) provided by Fugue.

Over the last one year we have been able to come up with new proofs which drastically enhance the upper bounds on differential attacks on the current embodiment of Fugue. With these proofs, we can now confidently tweak Fugue parameters so that Fugue-256 runs at twice the speed (in hardware and software) for long messages. For messages of size 512 bits the speed will remain the same for Fugue-256. This means we have added 150% more G1 rounds to Fugue-256. This alleviates any concern which Aumasson-Phan (and Gauravaran et al) analysis had, even though we have shown (because of our proofs on partial collisions) that that leads to no attacks. For this reason, in Fugue-224 we do not add many more G1 rounds, and hence Fugue-224 will run faster than before for smaller messages.

The new parameters for different Fugues are as follows:

There are 5 parameters : n,k,r,s,t as specified in the Fugue document.
n: number of words of output
k: number of SMIX per input word
s: internal state size
r: number of G1 rounds...total of rk SMIX t : number of G2 rounds...total of t * ceil(n/4) SMIX

new Fugue-224
n = 7 , k = 1, s = 30, r = 15, t = 13. (note, rk = 15) old Fugue-224
n= 7, k = 2, s= 30, r = 5, t= 13 (note, rk = 10)

new Fugue-256

n=8, k = 1, s = 30, r = 26, t= 13 (note, rk = 26) old Fugue-256
n = 8, k =2, s= 30, r = 5, t = 13   (note, rk = 10)

new Fugue-384
n = 12, k = 2, s = 36, r = 14, t= 13
old Fugue-384
n = 12, k = 3, s = 36, r = 6, t= 13 (note rk = 18)

new Fugue-512
n = 16, k = 3, s = 36, r= 14, t = 13
old Fugue-512
n = 16, k = 4, s = 36, r = 8, t=13 (note rk = 32)

Each of the Fugue's TMIX has also been tweaked. Rest of the specification remains exactly same.
The new TMIXes are as follows:
TMIX (for 224 and 256) : S_4 += S_0; S_0 = Input, S_14 += S_0; S_20 += S_0; S_8 += S_1.

TMIX (for 384)   S_7 += S_0; S_0 = Input; S_10 += S_0; S_14 += S_0; S_4 += S_1.

TMIX (for 512)   S_10 += S_0; S_0= Input; S_7 += S_0; S_11 += S_0; S_4 += S_1; S_22 += S_1.

A new weak Fugue will be specified which will consist of only 20 word state, and drastically shortened final round.

Thanks,

Fugue Team

Improved security analysis of Fugue-256 and its weaker version

Praveen Gauravaram[1], Nasour Bagheri[2], and Lars R. Knudsen[1]

[1] Department of Mathematics, Technical University of Denmark, DK-2800 Kgs. Lyngby, Denmark
[2] Shahid Rajaee Teacher Training University, Iran

Abstract. Fugue is one of the fourteen candidate hash algorithms in the second round of NIST's SHA3 competition. We consider Fugue-256, the 256-bit instance of Fugue and its weaker version called weakFugue-256. Fugue-256 updates a state of 960 bits with a *round transformation* **R** parametrized by a 32-bit message word. Twice in every state update, this transform invokes an AES like round function called **SMIX**. Fugue-256 relies on a *final transformation* **G** to output digests that look random which uses 18 rounds where each round invokes **SMIX** twice and finally the 960-bit output of the **G** transform is mapped to a 256-bit digest. The hash function weakFugue-256 has half the complexity of Fugue-256.

In this paper, we improve some of the previous analytical results of these two designs. First we present a pseudo differentiability attack which distinguishes unpadded Fugue-256 from a random oracle for a probability of 1. This attack finds many input states for the unpadded Fugue-256 that differ on average in only 80 bits such that their digests remain fixed for all pairs found. In addition, we extend this attack to full Fugue-256 in less than birthday complexity where it is possible to find two pairs of initial states that differ on average in 332 bits with their digest difference remain fixed. This improves over the previous result of Aumasson and Phan whose attack on the **G** transform when extended to the unpadded Fugue-256 produces input states that differ on average in 194 bits and it does not extend to full Fugue-256 in less than birthday complexity. Next, we improve the meet-in-the-middle preimage attack of Fugue-256 and reduce its complexity to 2^{416} from 2^{480} time and memory. Next, we improve Aumasson and Phans' integral distinguisher on the 5.5 rounds of the **G** transform to 16.5 rounds. Finally, we show a free start collision attack on weakFugue-256 for 2^{70} complexity when some other internal conditions hold showing that the designers' claim that it is provably free-start collision resistant upto 2^{96} is false and their analysis is incomplete.

Keywords: Fugue, Hash function analysis, SHA3 competition.

1 Introduction

Fugue [6] is a cryptographic hash function designed by Halevi, Hall and Jutla and is one of the fourteen second round SHA-3 hash function candidates [13,14]. The Fugue design can be viewed as an enhancement to the Grindahl hash function designed by Knudsen *et al.* [9] in which a large evolving internal state is maintained and the message words inserted into the state are processed using a *round transformation* and the complete state is maintained. After all the message words are processed, an extensive *final transformation* is applied to the state and part of its output is used as the digest.

The Fugue design has two main instances called Fugue-256, denoted **F**-256 and Fugue-512, denoted **F**-512 that produce digests of sizes 256 bits and 512 bits respectively and other instances such as Fugue-224 and Fugue-384 are related to these two designs. The designers of Fugue also proposed a weaker version of **F**-256 called weakFugue-256, denoted **wF**-256 [6] to encourage analysts to sink their teeth in. All versions of Fugue can hash messages of lengths upto $2^{64} - 1$ bits. In the rest of this paper, we consider **F**-256 and **wF**-256.

F-256 maintains an internal state S of 960 bits as a 4×30 matrix with each column containing a 32-bit word S_i for $i = 0, \ldots, 29$. It updates S by using a *round transformation* **R** parametrized by a message word of 32 bits. Once all input has been processed, the state undergoes through a *final transformation* **G**, composed of 5 rounds of **G1** and 13 rounds of **G2**. Subsequently, eight words of the output state are used as the digest of **F**-256. The main component of **F**-256 is a 16 byte to 16 byte permutation transformation called **SMIX** which consists of a substitution box

SBox followed by a linear transformation **SMIX-T** resembling the round functions of the AES block cipher [3]. The structure of **wF**-256 is similar to **F**-256 except that it has half the number of **SMIX**-es per input message word and half the complexity of **G1** and 5 rounds of **G2**.

By using proof-oriented methods, the designers proved that mounting a collision attack on **F**-256 would require a work factor of at least 2^{128}. They also claimed that the best second preimage and preimage attacks require a work factor of 2^{256} [6, § 1.2]. They also noted that since the *round transformation* and *final transformation* of **F**-256 are invertible, a generic meet in the middle attack can be applied to **F**-256 to find preimages with a work factor of 2^{480}. The hash function **wF**-256 was claimed to have a 2^{96} security against free-start collision attacks [6, § 1.2].

1.1 External analysis on Fugue

To our knowledge, so far there are only two external analytical results published for the Fugue hash function. Khovratovich [7] showed a collision attack on the internal states of **F**-256 and **F**-512 hash functions with time and memory complexities of 2^{352} and 2^{480} respectively. Recently, Aumasson and Phan [1] showed a probability 1 differentiability attack on the *final transformation* **G** of **F**-256 which distinguishes it from a random function. In this attack, they showed pairs of input states to the **G** transform that differ on average in only 66 bits such that their digests are the same for all pairs found. They also showed an integral distinguisher on the 5.5 rounds of the **G** transform and on a tweaked variant of the **G** transform.

1.2 Our contributions

In this paper, we investigate the security of **F**-256 and **wF**-256 against some attacks considered by the designers and improve those attacks. We also consider the analysis of Aumasson and Phan [1] on the **G** transform of **F**-256 and extend it to show some interesting differential properties of **F**-256. Our results and their impact are summarized as follows:

1. Improved analytical results on **F**-256:
   (a) Pseudo differentiability attack: We show a method which finds pairs of *initial states* for **F**-256, without the provision of length-padding, that differ on average in only 80 bits such that their digests are the same under **F**-256 for all pairs found. Our method makes use of a new probability 1 differential characteristic which propagates after the first **SMIX** of the **G** transform for upto 14.5 rounds of **G** and also to one round of the **R** transform without activating their respective **SMIX**-es. This differential property of **F**-256 is unknown until now. In addition, we extend this differentiator to full **F**-256 in about 2^{97} work factor which is less than birthday complexity of 2^{128}.
   In comparison, when the differentiability attack of [1] on the **G** transform of **F**-256 is extended to one round of the R transform of **F**-256, pairs of *input states* (including message word) differ on average in 194 bits for their digests to be the same. Moreover, this attack extends to full **F**-256 in not less than birthday complexity. We remark that the designers of Fugue considered a variant of the portion of our differential path in the analysis of **F**-256 pseudo random function (PRF) to prove an upper bound probability of 2^{-142} to find partial collisions on the internal state. The designers noted that this argument also applies for a known or chosen key/Initial Value (IV) model of **F**-256 [5]. We remark that this argument also applies if our differential path is considered to find partial collisions on the internal state. Moreover, these partial collisions on the internal state are needed for our pseudo differentiability attack and they are not feasible with the current cryptanalytical techniques.

(b) Meet-in-the-middle preimage attack: We improve the designers' generic meet-in-the-middle preimage attack on any instance of Fugue with n-bit internal state from a complexity of $2^{n/2}$ to $2^{n/2-16}$. We then further improve our attack on **F**-256 by a factor of 2^{48} by showing a sophisticated technique that applies with a probability of 1 by exploiting the freedom available in the message words and in some words of the internal state. Thus, we reduce the complexity of the designers' attack on **F**-256 from 2^{480} to 2^{416}.

(c) Integral distinguisher for the **G** transform: We improve the previously known integral distinguisher [1] on the 5.5 rounds of the **G** transform of **F**-256 to 16.5 rounds.

2. Improved analytical results on **wF**-256:

   (a) Free-start collision attack on **wF**-256: We show a 2^{70} free-start collision attack on **wF**-256, assuming some other favorable internal conditions to hold. Since, this is the model in which **wF**-256 is proven to be resistant to collisions within 2^{96} by the designers [6, § 14], the claim that **wF**-256 is provably resistant to collisions within 2^{96} is FALSE. This claim was made in [6, § 1.2]. The discrepancy is resolved by noting that **wF**-256 uses a much weaker *final transformation* than **F**-256, and the designers have given no proof of external collision resistance for this weak *final transformation.* We show that a partial collision obtained in this model, goes on to produce an external collision. We remark that this just shows that the proof for full **wF**-256 is not complete, and we do not know (yet) how to obtain collisions for **wF**-256.

The rest of the paper is organised as follows: In §2, we describe **F**-256 and **wF**-256 designs. In §3, we provide some notation and definitions that are later used in some analysis. In §4,§5, §6, §7 and §8, we present our improved security analysis of **F**-256 and **wF**-256 hash functions. Finally, we provide concluding remarks in §9.

2 F-256 hash function

F-256 parses the 256-bit initial value (IV) as eight 4-byte words $IV_0, \dots, IV_7$. It initializes a state S of 30 4-byte words S_i for $i = 0, \dots, 29$, as a 4×30 matrix by assigning $S_j = 0$ for $j \in [0, 21]$ and $S_j = IV_{j-22}$ for $j \in [22, 29]$. This state of **F**-256 is called *inital state.* Hereafter, we denote by $S_{i\sim j}$ the consecutive words of a state S from the index i to j (including i and j). Streams of 4-byte message word inputs are processed from this state using a *round transformation* **R**. If the input message is not a multiple of 32 then **F**-256 pads the message with sufficient 0 bits so that the padded message is a multiple of 32. The padded message is appended with an additional 64 bits (i.e two 4-byte words) that represent the binary encoding of the length of the unpadded message in big-endian notation. Once the length encoded message is processed with the **R** transform, a *final transformation* **G** is applied to the internal state to obtain an *output state* of 30 words. The eight words $S_{1\sim 4}$,$S_{15\sim 18}$ of the *output state* are used as the digest. The transforms **R** and **G** are discussed below where the addition $+$ is addition of vectors of four bytes in GF(2^8), and hence is the same as 32-bit exclusive-or and for 4-byte vectors a and b, $a+=b$ means $a = a + b$.

Round transformation (R). The **R** transform takes a state S and a 4-byte message word m as inputs and outputs a new thirty column state. The transformation **R** calls a sequence of functions: **TIX**(m), **ROR3**, **CMIX**, **SMIX**, **ROR3**, **CMIX**, **SMIX**.

- The function **TIX**(m) has the following steps:
  - $S_{10}+=S_0$; $S_0 = m$
  - $S_8+=S_0$; $S_1+=S_{24}$

- The function **ROR3** rotates the state to the right by three columns, that is $S_i = S_{i-3 \bmod 30}$.
- The column mix function **CMIX** has the following steps:
  - $S_0 += S_4$; $S_1 += S_5$; $S_2 += S_6$
  - $S_{15} += S_4$; $S_{16} += S_5$; $S_{17} += S_6$
- The **SMIX** transform operates only on the first four columns $S_{0\sim3}$ of the state S that are viewed as a 4×4 matrix of 16 words. Each byte of these columns first undergoes an **SBox** transform which is the one as in AES [3] and the resulting matrix undergoes an **SMIX-T** transform denoted by a 16×16 matrix $\mathbf{N}$ of 256 bytes. That is, $S'_{0\sim3} = \mathbf{N}.(S_{0\sim3})$ where $\mathbf{N}$ is multiplied (.) with a 16-byte 4×1 column matrix output of **SBox**. Similarly, $(S_{0\sim3}) = \overline{\mathbf{N}}.(S'_{0\sim3})$ where $(S'_{0\sim3})$ is a 16-byte 4×1 column matrix.

Final transformation (G). The **G** transform takes the output S of the **R** transform and produces a *final state* of 30 words. The function **G** consists of 5 rounds of **G1**, 13 rounds of **G2** and a binary addition of two state words. The five rounds of **G1** are denoted by $\mathbf{G1_1}, \ldots, \mathbf{G1_5}$ and thirteen rounds of **G2** are denoted by $\mathbf{G2_1}, \ldots, \mathbf{G2_{13}}$. These operations on a state S are given by:

- Function **G1**: It is a sequence of functions: **ROR3**, **CMIX**, **SMIX**, **ROR3**, **CMIX**, **SMIX**.
- Function **G2**: It has the following steps:
  - $S_4 += S_0$; $S_{15} += S_0$; **ROR15**; **SMIX**
  - $S_4 += S_0$; $S_{16} += S_0$; **ROR14**; **SMIX**
- $S_4 += S_0$; $S_{15} += S_0$

In the above, **ROR15** and **ROR14** mean rotations of the state S to the right by 15 and 14 columns respectively. The resultant state is called *final state*.

256-bit digest. After discarding the words S_0,$S_{5\sim14}$ and $S_{19\sim29}$ from the *final state*, the concatenation of the words $S_{1\sim4}$ and $S_{15\sim18}$ is used as the digest. This step of producing the digest from the **G** transform is denoted by $\tau(\mathbf{G}(S))$ where S is the input state of **G**.

2.1 wF-256 hash function

Similar to **F**-256, **wF**-256 maintains a 4×30 state S which is initialized by setting $S_j = 0$ for $j = 0, \ldots, 21$ and $S_{22+j} = IV_j$ where $j = 0, \ldots, 7$. However, **F**-256 uses fewer rounds for the **R** and **G** transforms compared to **F**-256 and has different **CMIX** and **TIX(m)** functions. These transforms on a state S are defined below:

- **TIX(m)** is a sequence of steps: $S_4 += S_0$, $S_0 = m$, $S_8 += S_0$.
- The **CMIX** transformation has the following steps:
  - $S_0 += S_4$; $S_1 += S_5$; $S_2 += S_6$
  - $S_{15} += S_4$; $S_{16} += S_5$; $S_{17} += S_6$

Round transformation (R). This transform takes a 4-byte message word m and a state S as inputs and outputs a new internal state. It is a sequence of functions: **TIX(m)**, **ROR3**, **CMIX**, **SMIX**.

Final transformation (G). This transform takes the output state of the **R** transform as its input and produces a *final state*. It consists of five rounds of **G1**, five rounds of **G2** and a binary addition of two state words. Each round of these operations on a state S and the state word addition operation are described as follows:

- Function **G1**: It is a sequence of steps: **ROR3**, **CMIX** and **SMIX**.
- Function **G2**: It has the following steps:
  - $S_4 + = S_0$; $S_{15} + = S_0$; **ROR15**; We call this step **G2**. **SMIX**.
  - $S_4 + = S_0$; $S_{16} + = S_0$; **ROR14**; **SMIX**. We call this step $\mathbf{G2}'$.
- $S_4 + = S_0$; $S_{15} + = S_0$

The five rounds of **G1** are denoted by $\mathbf{G1}_1, \ldots, \mathbf{G1}_5$ and those of **G2** by $\mathbf{G2}_1, \mathbf{G2}'_1, \ldots, \mathbf{G2}_5, \mathbf{G2}'_5$. The concatenation of the words $S_{1\sim4}, S_{15\sim18}$ of the *final state* is the digest of **wF**-256.

3 Notation and Definitions

In this section, we introduce some notation on **F**-256 and definitions that are used later in our analysis. Notation specific to some parts of the analysis will be introduced in the relevant sections. Notation used for the free-start collision attack on **wF**-256 is provided in §8.

3.1 Notation

In any round i of **R**, the internal state (also the starting state of Round i) is denoted by *State-i* and its words are denoted by $S_0^i, S_1^i, \ldots, S_{29}^i$, i.e, $S_{1\sim29}^i$. The internal state words after the first **SMIX** in a round i are denoted by $S_0^{i.5}, \ldots, S_{29}^{i.5}$, i.e, $S_{0\sim29}^{i.5}$. In any round i of R, the internal state words after the first **ROR3**, **CMIX** and **SBox** transformations are denoted by $x_0'^i, \ldots, x_{29}'^i$, $x_0^i, \ldots, x_{29}^i$ and $\hat{x}_0^i, \ldots, \hat{x}_{29}^i$ respectively and those after the second **ROR3**, **CMIX** and **SBox** transformations are denoted by $y_0'^i, \ldots, y_{29}'^i$, $y_0^i, \ldots, y_{29}^i$ and $\hat{y}_0^i, \ldots, \hat{y}_{29}^i$ respectively. We indicate any non-zero difference in the state words by placing δ before those words. For example, differences in the words of *State-i* are denoted by $\delta S_0^i, \ldots, \delta S_{29}^i$, i.e, $\delta S_{0\sim29}^i$. A message word inserted in the i^{th} round of **R** is denoted by m^i. Message words in the rounds from i to j are denoted by $m^i \sim m^j$ and differences in the message words in these rounds by $\delta m^i \sim \delta m^j$.

3.2 Distinguishers and Differentiators

In a distinguishability attack on a hash function, an adversary distinguishes the hash function from a random oracle by querying the hash function as a black box. Instead of accessing the hash function as a black box, if the adversary accesses its internal components or the internal state to distinguish it from a random oracle then such an attack is called the differentiability attack following the indifferentiability notion for hash functions [2, 11]. These two attacks are also called a distinguisher and a differentiator respectively. For example, the well-known message extension attack [2, 10] on the popular Merkle-Damgård hash function construction [4, 12] is a differentiability attack as an attacker uses the digest of a message without knowing the message except its length to compute the digest of a new message. Similarly, we can define these attacks for other components of a hash function such as the round and final transformations. If an adversary distinguishes a hash function with an *initial state* different from the one defined for the hash function or with two distinct states then we call such attacks the pseudo differentiators.

4 Pseudo differentiator for unpadded F-256

We present a pseudo differentiator for **F**-256 for a probability of 1 by finding pairs of states (S, S^*) for **F**-256 that differ on average in only 80 bits such that $\mathbf{F}\text{-}256(S, m) \oplus \mathbf{F}\text{-}256(S^*, m)$ remains fixed for all the pairs of (S, S^*) found. Our attack applies for the composition of 1 round of **R** transform and full **G** transform and hence, does not include the length padding of **F**-256 as its provision requires at least three **R** transforms. The attack is outlined below:

4.1 Differential property of SMIX-T

Recall that **SMIX-T** is a linear transform represented as a 16×16 matrix $\mathbf{N}$ and $\mathbf{N}.(S_{0\sim3}) = (S'_{0\sim3})$ and therefore, $\overline{\mathbf{N}}.(S'_{0\sim3}) = (S_{0\sim3})$ where $(S_{0\sim3})$ and $(S'_{0\sim3})$ are 4×1 column matrices. We note that it is easy to find an input difference $(\delta S'_0, \delta S'_1, \delta S'_2, \delta S'_3)$ to $\overline{\mathbf{N}}$ such that we get an output difference $(\delta S_0, \delta S_1, \delta S_2, \delta S_3)$ where $\delta S'_0 = 0$, $\delta S'_3 = 0$ and $\delta S_3 = 0$. That is, we can find a pair of inputs (S'_0, S'_1, S'_2, S'_3) and $(S'_0, S'^*_1, S'^*_2, S'_3)$ to $\overline{\mathbf{N}}$ that collide on the word S_3 for any S'_0 and S'_3 where $\delta S'_1 = S'_1 \oplus S'^*_1$ and $\delta S'_2 = S'_2 \oplus S'^*_2$. This attack can be precomputed and the solution for this attack is the solution to 8 linear equations in 8 unknowns which requires negligible computational cost. We can also fix 3 bytes in one of the differences of $\delta S'_1$ and $\delta S'_2$ to zeroes which leads to solving 5 linear equations in 5 unknowns. As shown later in our attack, the difference $\delta S'_1$ propagates to more words in the input state of the **R** transform compared to $\delta S'_2$. Hence, to minimise the number of *active* state input bits of our attack, we fix 3 bytes of $\delta S'_1$ to a zero difference and vary the other word for a solution. By running an experiment for a minimum weight solution, we found $\delta S'_1 = 0\text{x}\ 00000009$ and $\delta S'_2 = 0\text{x}\ 5042d427$. These differences are *fixed* and we call them δ_1 and δ_2 respectively.

4.2 Differentiability attack on the $\tau(\mathrm{G}(S))$ transform

We can choose any intermediate state in the **G** transform and proceed *forwards* and *backwards* to compute its corresponding *final* and input states. This is called *inside-out* strategy [1] which we use to differentiate the $\tau(\mathbf{G}(S))$ transform from a random function with a probability of 1.

Forwards from $\mathbf{G2}_{11}$. We choose an internal state at the start of the round $\mathbf{G2}_{11}$ such that the words S_{18} and S_{19} have the differences δ_1 and δ_2 computed in § 4.1 and the remaining words ($S_{0\sim17}$ and $S_{20\sim29}$) have the zero difference. The differences δ_1 and δ_2 activate **SMIX**-es in the rounds $\mathbf{G2}_{11}$ and $\mathbf{G2}_{12}$ respectively. The digest returned after the **G** transform depends on the differences δ_1 and δ_2 but not on all the state words at the start of the round $\mathbf{G2}_{11}$. This observation allows us to find distinct pairs of states at $\mathbf{G2}_{11}$ such that the difference of their respective digests is fixed for all pairs found. A closer analysis shows that the digest does not depend on the state words $S_{8\sim14}$ and $S_{22\sim29}$ at the start of the round $\mathbf{G2}_{11}$. Hence, in the round $\mathbf{G2}_{11}$ we can build many pairs of states by fixing the words $S_{0\sim7}$, $S_{15\sim17}$ and $S_{20\sim21}$ with the same actual value, the words S_{18} and S_{19} with some values that differ by δ_1 and δ_2 respectively and varying the remaining words with zero difference for all pairs of states to obtain digests that have the fixed difference.

Backwards from $\mathbf{G2}_{11}$. Consider any two internal states at the start of $\mathbf{G2}_{11}$ that satisfy the above constraint of having the words $S_{0\sim7}$, $S_{15\sim17}$ and $S_{20\sim21}$ fixed with the same value (i.e, zero difference) and the words S_{18} and S_{19} fixed to some value but with the differences δ_1 and δ_2 respectively. When we process these two states until the end of the round $\mathbf{G1}_2$ in the backward direction, we get two intermediate states with the differences δ_1 and δ_2 appearing in the words

S_4 and S_5. All other words have zero differences. When the second half of $\mathbf{G1}_1$ is inverted, we obtain the difference δ_1 (resp. δ_2) in the words S_1, S_{12} and S_{27} (resp. S_2, S_{13} and S_{28}). When we invert the remaining half of $\mathbf{G1}_1$, the differences δ_1 and δ_2 in the state words S_1 and S_2 activate $\overline{\mathbf{SMIX}}$ creating uncontrolled differences in the words $S_{0\sim2}$ and zero difference in the word S_3 as shown in § 4.1. Hence, we get the input state to the $\mathbf{G}$ transform with the difference δ_1 (resp. δ_2) in the words S_9 and S_{24} (resp. S_{10} and S_{25}) and uncontrolled differences in the words $S_{27\sim29}$. That is, only seven words of the input state to the $\mathbf{G}$ transform have differences. Recall that the differences δ_1 and δ_2 are not chosen arbitrarily and they are determined by the attack described in § 4.1. For our solution, δ_1 and δ_2 have the weights of 2 and 12 respectively, and the uncontrolled differences in the words $S_{27\sim29}$ have, on average, a weight of 48. Hence, we have shown a method which finds pairs of state inputs (S, S^*) to the $\mathbf{G}$ transform that differ, on average, in 76 bits such that the difference of $\tau(\mathbf{G}(S))$ and $\tau(\mathbf{G}(S^*))$ remains *fixed* for all such pairs found.

4.3 Extending to differentiate unpadded F-256: Pseudo differentiability attack

Following § 4.2, the differentiator for the $\tau(\mathbf{G}(S))$ transform has produced the difference δ_1 in the words S_9 and S_{24}, difference δ_2 in the words S_{10} and S_{25}, random differences in the words $S_{27} \sim S_{29}$ and zero difference in the other words of the state of $\mathbf{G}$. When this state is inverted through one round of $\mathbf{R}$, the two $\overline{\mathbf{SMIX}}$-es of the $\mathbf{R}$ transform receive zero input difference and hence, they are not activated. Thus, we receive a message difference of zero ($\delta m = 0$) for the $\mathbf{R}$ transform. The difference δ_1 (resp. δ_2) appears in the words S_3, S_{14}, S_{18} and S_{29} (resp. S_4, S_{19}) and the uncontrolled differences appear in the words $S_{21\sim23}$ of the input state of $\mathbf{R}$. Note that δ_1 and δ_2 have a weight of 2 and 12 respectively and the words $S_{21\sim23}$ have an average weight of 48. Thus, we have shown a method which finds pairs of input states (S, S^*) for $\mathbf{F}$-256 that differ, on average, in only 80 bits such that $\mathbf{F}\text{-}256(S, m) \oplus \mathbf{F}\text{-}256(S^*, m)$ is fixed for all pairs found.

The differential path which demonstrates this attack is shown in Figure 1. This path shows the differences in the state words (represented as square cells) at the start and the end of every full round of the $\mathbf{G}$ and $\mathbf{R}$ transforms. The cells in Magenta color represent δ_1, those in Blue color represent δ_2 and those in Red color represent random differences. Cells in Green color are zero difference words chosen at the start of the *Forwards* phase of the attack with their actual values fixed from the round $\mathbf{G2}_{11}$ and plain cells are zero differences whose values we vary in the attack. In Table 2 of Appendix A, we have shown experimental results of our attack.

Remark 1. Our probability 1 pseudo differentiator on the unpadded $\mathbf{F}$-256 does not extend to the full $\mathbf{F}$-256 as $\mathbf{F}$-256 reserves the last two rounds of $\mathbf{R}$ to process the 64-bit representation of the length of the unpadded message. Hence, the attack must be extended to a minimum of three $\mathbf{R}$ transforms in the **Backwards from $\mathbf{G2}_{11}$** step of the attack in such a way that the last two $\mathbf{R}$ transforms process the length encoding part of the input message. If we intend to extend the attack to 3 rounds of the $\mathbf{R}$ transform, then the second and last $\mathbf{R}$ functions must process the words 0x00000000 and 0x00000200 (representing 32-bit length message) respectively. However, since we have no control over the exact value of the message produced in the last round of $\mathbf{R}$ by the attack, it is difficult to force it to 0x00000200. Similarly, it is difficult to force the second message word to 0x00000000 for the pairs of states involved in the attack as $\mathbf{SMIX}$-es are activated in the second round creating differences in the message word.

5 Pseudo differentiator for full F-256 in less than birthday complexity

Although the attack outlined in §4.2 does not let us to mount a probability 1 pseudo differentiator for the full F-256 hash function as noted in §1, we can still achieve this result in less than birthday complexity as follows:

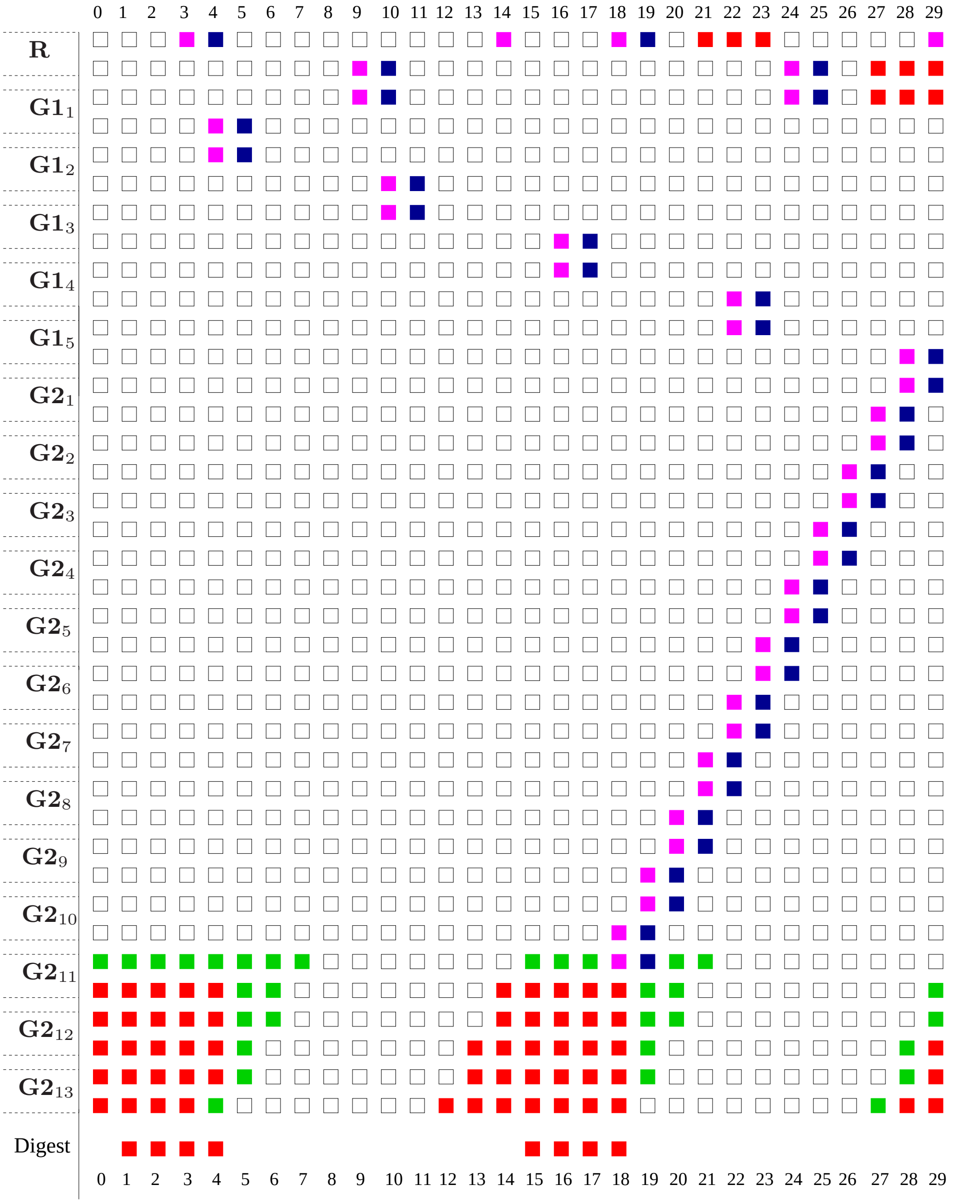

Fig. 1. Differential path showing pseudo differentiability attack on unpadded **F**-256

1. **Obtaining** 0x00000200 **pad bits in the last round of R**: Since there are no active **SMIX**-es in this last round of **R**, the message difference of this round is always zero as shown in §4.3. Hence, to match the padding bits of 0x00000200 in the last round input of **R**, we need to repeat the **Backwards from G2**$_{11}$ step of the attack for only 2^{32} times by using the freedom available in the words $S_{8\sim14}$ and $S_{22\sim29}$ at the start of **G2**$_{11}$. That is, one out of 2^{32} pairs of states (S, S^*) for the unpadded **F**-256 whose digests remain fixed for all the pairs of (S, S^*) found would have $m =$ 0x00000200. To produce two pairs of such states, we need to repeat **Backwards from G2**$_{11}$ step of the attack for 2^{32} times more (i.e in total 2^{33} times).
2. A pair of states that have produced the same digest difference in Step 1 will have the padding bits of 0x00000000 in the second last round of **R** for a probability of 2^{-64} as **SMIX**-es are active in this round. Therefore, the **Backwards from G2**$_{11}$ step of the attack must be repeated for 2^{96} times by using the freedom in the words $S_{8\sim14}$ and $S_{22\sim29}$ to obtain a pair of states that contain the correct padding bits (i.e 64 bits) and producing the same digest difference for **F**-256. The complexity of the attack is 2^{97} in order to produce two such pairs of states.
3. The two pairs of states from Step 2 can be further inverted for one more round of **R** to obtain two pairs of states for the full **F**-256 such that their digests remain fixed in 2^{97} work factor. On average, we can obtain two pairs of input states that differ on average in 332 bits such that their digest difference is the same. In total, we can produce at most 31 such pairs of states that differ on average in 332 bits with their digest difference remain fixed in 2^{127} work factor.

This attack demonstrates insufficient diffusion in the full **F**-256. Further improvement can be made to this attack as pointed out in Appendix B.

5.1 Comparison with the previous analysis of [1] and [6]

Aumasson and Phan [1] showed a differentiator for the $\tau(\mathbf{G}(S))$ transform of **F**-256 by showing pairs of input states to the **G** transform that differ on average in only 66 bits such that the difference in the digests remain fixed for all the pairs found. In this attack, the state at the round **G2**$_{12}$ is chosen such that the word S_{18} has a difference δ and the remaining words have zero difference. This differential state propagates *backwards* for 15 rounds of **G** and then activates the $\overline{\mathbf{SMIX}}$ of $\mathbf{G}_1$, leading to the difference δ in the words S_{10} and S_{25} and random difference in the words S_0 and $S_{27\sim29}$ at the input state of **G**. When $\delta = 1$, we can find pairs of input states to the **G** transform that differ on average in only 66 bits such that their digests are constant for all pairs found. When this attack is extended to one round of **R**, both **SMIX**-es in that round get activated leading to pairs of input states (including message word) to the unpadded **F**-256 that differ on average in 194 bits such that their digests remain fixed for all pairs found. In addition, the differentiator of [1] cannot be extended to the full **F**-256 for a complexity better than birthday attack due to the activation of **SMIX**-es in the last round of the **R** transformation.

The differential characteristic of our pseudo differentiator can also be viewed as an improved variant of the characteristic considered by the designers in the analysis of the PRF mode of **F**-256 [6, §12.4.2]. The internal state they have considered at the start of the **G** transform for the partial collisions differ in the same words as ours and in an additional word of S_0. They proved that the probability of obtaining this partial collision for the **F**-256 PRF is atmost 2^{-142} and observed that this result also holds for **F**-256 with the chosen key/IV [5]. We remark that the designers' argument is also applicable if we consider our differential characteristic for the partial collisions on the internal state. Firstly, as noted in Remark 1 extending our attack to the full **F**-256 with such a partial collision on the internal state is difficult for a probability of

1. In addition, it is also difficult to obtain the partial collision on the internal state for a pair of chosen IVs by following the differentiator of §5 in less than birthday complexity because the first 22 columns of the *initial state* would not end up with a zero difference (*initial state* is identical for the words $S_{0\sim 21}$ regardless of the IV). Finally, the characteristic considered by the designers also has a similar drawback as that of Aumasson-Phan in differentiating full **F**-256 in less than birthday complexity.

6 Improved meet-in-the-middle attacks on Fugue

6.1 Generic meet-in-the-middle preimage attack on Fugue [6, p.77]

The designers of Fugue noted the application of a generic meet-in-the-middle preimage attack on any t-bit instance of Fugue [6, p.77] with n-bit ($n/32$-word) internal state for $2^{n/2}$ time and memory complexity. For a given digest Y, this attack finds a preimage for Fugue as follows:

1. *Forward process:* Choose $2^{n/2}$ messages of equal length and process them to the corresponding internal states at *State-i* of some round i of **R** from the *initial state* of Fugue. Store these messages and their internal states at *State-i* in a Table L_1.
2. *Backward process:* Fill the output state words $S_{1\sim 4}$ and $S_{15\sim 18}$ with the digest Y. Build $2^{n/2}$ *final states* by choosing random values for the remaining 22 words and invert the *final transformation* **G** for these states and may also invert the round transformation **R** by using a few random message words (e.g. 2 words)[3] to build $2^{n/2}$ internal states at *State-i*. Store these state values together with the corresponding message words in a Table L_2.
3. Due to birthday paradox, an internal state in L_1 collides with an internal state in L_2 with a good probability and this event is called a *collision match.* The internal state *State-i* at which the *collision match* occurs is called the *middle state.* Let M and $M^{'}$ be the corresponding messages of the colliding internal states respectively. Finally, produce $M\|M^{'}$ as the preimage of Fugue for a given digest Y.

The complexity of this preimage attack is influenced by the internal state size. On **F**-256 and **F**-512, the time and memory complexity of this attack is 2^{480} and 2^{576} respectively.

6.2 Improved generic meet-in-the-middle preimage attack on Fugue

Let *State-i'* be the internal state in any round i of the **R** transform after the step $S_{10}^i = S_0^i \oplus S_{10}^i$. This is a $(n-32)/32$-word internal state without the word S_0^i and except for the word S_{10}^i in the *State-i'* all other words are the same as in the $n/32$-word *State-i*. The generic meet-in-the-middle attack on Fugue can be improved by inverting at least one round of the **R** transform for some random message in the *backward process* of the attack and using any *State-i'* as the *middle state* for a *collision match.* The improved attack reduces the complexity of the generic attack on Fugue to $2^{n/2-16}$ which is 2^{464} and 2^{560} for **F**-256 and **F**-512 respectively.

6.3 Improved meet-in-the-middle preimage attack on F-256

We further improve the preimage attack on **F**-256 from § 6.2 by exerting control over 3 words of the 29-word *middle state.* This technique allows us to use a birthday attack to match only 26 words of the *middle state*, thereby reducing the complexity of the attack to 2^{416} from 2^{464}. Let 0 be the round of the **R** transform at which we aim for a *collision match.* Let $-1, -2, -3, \ldots$ and $1, 2, 3, \ldots$ be the **R** transforms from the 0^{th} round in the *Forward process* and *Backward process* of the attack. The attack is outlined below:

[3] It is not necessary to invert any **R** transforms by using message words as the attacker can obtain enough degrees of freedom to invert the *final state* from the 22 truncated output words.

1. We show that the words S^0_{17}, S^0_{23} and S^0_{27} in the *middle state* (i.e *State-0'*) can be controlled such that the internal states evolving from the *initial state* and the *final state* of **F**-256 can be matched in these words deterministically with a probability of 1 by solving a simple system of equations. We do this by first assigning fixed values[4] to the words S^0_{17}, S^0_{23} and S^0_{27} that are controlled by using the **R** transforms -3, -2 and -1 in the *Forward process* and the **R** transforms 3, 2 and 1 in the *Backward process.* In the *Forward process*, the desired value for the words S^0_{27}, S^0_{23} and S^0_{17} is obtained consecutively by using the freedom available in the message words m^{-3}, m^{-2} and m^{-1} of the **R** transforms -3, -2 and -1 respectively. In the *Backward process*, the desired values for the words S^0_{17}, S^0_{23} and S^0_{27} are obtained consecutively by using the freedom available in the words S^3_0, S^2_0 and S^1_0 in the **R** transforms 3, 2 and 1 respectively. Below we explain how the word S^0_{17} can be controlled and a similar explanation follows for controlling the words S^0_{23} and S^0_{27}.

   (a) *Controlling the word S^0_{17}:* Below we will show how we can obtain the desired word S^0_{17} of the *middle state* from the *final state* and *initial state* of **F**-256 through the *Backward process* and *Forward process* respectively.

      i. *Backward process:* The word S^0_{17} in the *middle state* of the 0th **R** transform will be the word S^2_{29} in the *State-2'* of the 2nd **R** transform. Now $x'^2_2 = S_{17^0}$, $x^2_2 = x'^2_2 \oplus x'^2_6$. Now $\hat{x}^2_2 = \mathbf{SBox}(x^2_2)$. Note that $(S^{2.5}_1, S^{2.5}_2, S^{2.5}_3) = (S^3_4, S^3_5, S^3_6)$. A *final state* of **F**-256 inverted in the *backward* direction till the **R** transform 3, the *State-3'* of the 3rd **R** transform is fixed. Therefore, we can only use $S^{2.5}_0$ input to $\overline{\mathbf{N}}$ to obtain the desired $\hat{x}^2_2$ and therefore, we can obtain the desired S^0_{17}. The matrix **N** has a property that by controlling one of the input words, we can obtain one desired output word by solving a system of 4 equations in 4 unknowns for a negligible complexity. This property is also applicable for $\overline{\mathbf{N}}$. Hence, we can find a $S^{2.5}_0$ such that $\overline{\mathbf{N}}.(S^{2.5}_0, S^{2.5}_1, S^{2.5}_2, S^{2.5}_3)$ produces the desired word $\hat{x}^2_2$. This process also determines the message word m^2 which is $\overline{\mathbf{SBox}}(\hat{x}^2_3) = x^2_3 = x'^2_3$. Note that $S^{2.5}_0 = y'^2_3 = y^2_3$, $\hat{y}^2_3 = \mathbf{SBox}(y^2_3)$ and the state words $S^3_{1\sim 29}$ of the state *State-3'* are determined by the *final state* of **F**-256. Now we vary S^3_0 such that $\overline{\mathbf{N}}.(S^3_0, S^3_1, S^3_2, S^3_3)$ produces the desired $\hat{y}^2_3$. Once we have found the candidate S^3_0, we can determine $S^3_{10} = S^3_0 \oplus x'^3_{13}$ of the state *State-3.*

      ii. *Forward process:* For an *initial state* of **F**-256 processed till the end of the -2nd **R** transform, the *State-(-1)* of the -1th **R** transform is fixed. This implies that y'^{-1}_{17} has already been fixed. To obtain the desired value of S^0_{17}, we need to control y'^{-1}_6 which is $y'^{-1}_{17} \oplus S^0_{17}$. The word y'^{-1}_6 is the same as $S^{-1.5}_3$. Note that the words $S^{-1}_{1\sim 29}$ had already been fixed. Hence, we can determine the words ($\hat{x}^{-1}_0$, $\hat{x}^{-1}_1$, $\hat{x}^{-1}_2$), the first three word input to **N**, as follows:

         A. $\hat{x}^{-1}_0 = \mathbf{SBox}(S^{-1}_{27} \oplus S^{-1}_1 \oplus S^{-1}_{24})$
         B. $\hat{x}^{-1}_1 = \mathbf{SBox}(S^{-1}_{28} \oplus S^{-1}_2)$
         C. $\hat{x}^{-1}_2 = \mathbf{SBox}(S^{-1}_{29} \oplus S^{-1}_3)$

         Having determined the words $\hat{x}^{-1}_0$, $\hat{x}^{-1}_1$ and $\hat{x}^{-1}_2$, we can use the freedom available in the message word m^{-1} to determine the candidate $\hat{x}^{-1}_3$ such that we obtain the desired $S^{-1.5}_3 = y^{-1}_6$ and therefore, we obtain the desired word $S^0_{17} = y^{-1}_6 \oplus y'^{-1}_{17}$ in the *middle state.*

Remark 2. It is difficult to exert control over more than 3 words in the *middle state* of the 0th **R** transform deterministically with a probability of 1 as the message word m^{-3} from the -3rd **R** transform influences the starting internal state of the 0th **R** transform. This diffusion property of **F**-256 was also noted in [6, p.41].

[4] These three words can have different values but they must be fixed.

7 Integral distinguisher for the 16.5 rounds of the G transform of F-256

Our integral attack is a first order integral attack. We follow the notation of [8] for the bytes included in the integral as follows: The symbol $\mathcal{C}$ (for Constant) in the i^{th} byte means that the values of all i^{th} bytes in the attack are equal. The symbol $\mathcal{A}$ (for All) means that all bytes in the attack are different, and the symbol $\mathcal{S}$ (for Sum) means that the sum of all i^{th} bytes is predictable and we write ? when the sum of the bytes is not predictable. We count the rounds of the **G** transform from 0 to 17 and a state in any round i where $i = 0, 0.5, 1, \ldots, 16, 16.5, 17$ is denoted by S^i and the words of S^i by $S^i_{0\sim 29}$.

Integral distinguisher on the 5.5 rounds of the G transform [1]. Aumasson and Phan [1] presented an integral distinguisher for 5.5 rounds of the **G** function. Their distinguisher fixes all the bytes of the state S^0 except for the first byte of S^0_2 at the start of the **G** transform. All possible values are assigned to the first byte of S^0_2. They have shown that for $S^0_5 = \mathcal{A}\|\mathcal{C}\|\mathcal{C}\|\mathcal{C}$ one would receive $S^{5.5}_0 = ?\|?\|?\|?, S^{5.5}_1 = \mathcal{A}\|?\|?\|?, S^{5.5}_2 = \mathcal{A}\|?\|?\|?, S^{5.5}_3 = \mathcal{S}\|?\|?\|?$ which presumably shows a non-randomness property in the first 5.5 rounds of the **G** function. In addition, this attack was extended to a tweaked version of **G**-function. We improve their attack on 5.5 rounds of **G**-function such that it can be applied to 16.5 rounds out of 18 rounds.

7.1 Improved attack

A closer analysis of integrals reveals that the values of the integral before the **ROR** functions of **G2** play a crucial role on the success of the distinguisher. It turns out that this word remains unchanged through many rounds of **G2** before being affected by other words. However, for the given integral, all bytes of $S^{5.5}_0$ are unknown ('?') and out of control of the adversary. Hence, the integral of Aumasson and Phan does not seem to extend to more rounds of the **G** transform. Our analysis revealed an integral that runs for more rounds. The propagation of our integral has been depicted in Table 3. It should be mentioned that values with notation $\mathcal{A}$ and $\mathcal{C}$ in the integral are unchanged through **SBox**, but values with notation $\mathcal{S}$ are unknown (?) after **SBox**.

In our integral, we fix whole state bytes of S^0, except for the second byte of S^0_4 where we consider all possible values. The word S^0_4 propagates to S^5_{28} with probability 1. Hence, the **ROR3** transform in the 5^{th} round of **G1** (i.e 4th round of **G**) shifts this word as one of the inputs to the **SMIX**. Hence, we obtain $S^{5.5}_0 = ?\|?\|\mathcal{S}\|?$. It means that we know the sum of the values ($\mathcal{S}$) for this word. On the other hand, this word is propagated to S^{16}_4 with probability 1.

In the next step, we have $S^{16}_4 + = S^{16}_0$ which destroys our integral. However, after the **ROR15** function in the 16^{th} round of **G**, S^{16}_0 and S^{16}_4 are propagated to S^{16}_{15} and S^{16}_{19} respectively. Now if we assume that the adversary has also access to S^{16}_{19} then he can combine S^{16}_{15} and S^{16}_{19} and retrieve the integral values as $S^{16.5}_4 = S^{16.5}_{15} \oplus S^{16.5}_{19}$.

Hence, we have an integral which applies to 16.5 out of 18 rounds of the **G** transform. Our findings illustrate the weak diffusion of the **G** transform.

8 A free-start collision attack on wF-256

We first present some notation relevant for the free-start collision attack on **wF**-256. The rounds of the **R** transform are numbered $\ldots, -3, -2, -1, 0$. At the start of a round i of **R**, the internal state and its difference are denoted by S^i and δS^i respectively. The words of S^i and δS^i are denoted by $S^i_{0\sim 29}$ and $\delta S^i_{0\sim 29}$ respectively. The message difference in a round i is denoted by δm^i. In any round i of R, after the **ROR3**, **CMIX** and **SBox** transforms, the internal state words are denoted by $x'^i_0, \ldots, x'^i_{29}$, $x^i_0, \ldots, x^i_{29}$ and $\hat{x}^i_0, \ldots, \hat{x}^i_{29}$ respectively and the differences are denoted by $\delta x_0'^i, \ldots, \delta x_{29}'^i$, $\delta x_0^i, \ldots, \delta x_{29}^i$ and $\delta \hat{x}_0^i, \ldots, \delta \hat{x}_{29}^i$ respectively. The functions of R in a round i are denoted by $\mathbf{TIX}_i$, $\mathbf{ROR3}_i$, $\mathbf{CMIX}_i$ $\mathbf{SBox}_i$ and $\mathbf{N}_i$ and their inverses by $\overline{\mathbf{TIX}}_i$, $\overline{\mathbf{ROR3}}_i$, $\overline{\mathbf{CMIX}}_i$ $\overline{\mathbf{SBox}}_i$ and $\overline{\mathbf{N}}_i$. Let C be a linear code which models $\overline{\mathbf{N}}$ and I be a set such that $I \subset \{0, 1, 2, 3\}$. By C_I, we mean a subset of C such that the columns that are not included in I are set to zero and the rest are non-zero. For instance, C_0 includes those inputs to $\overline{N}$ such that $S_0 \neq 0$, $S_1 = 0$, $S_2 = 0$ and $S_3 = 0$. Differences in the state and message words are indicated by δ and those with "?" are unfixed whose freedom can be used in the attacks.

Table 1. Distinguisher path for the composition of 1 round of **R** and **G** transforms of wF-256

S_i, $i =$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
S^0	0	0	0	0	0	0	δ	δ	δ	δ	δ	δ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\mathbf{G1}_1$	0	0	0	0	0	0	0	0	0	δ	δ	δ	δ	δ	δ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\mathbf{G1}_2$	0	0	0	0	0	0	0	0	0	0	0	0	δ	δ	δ	δ	δ	δ	0	0	0	0	0	0	0	0	0	0	0	0
$\mathbf{G1}_3$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	δ	δ	δ	δ	δ	δ	0	0	0	0	0	0	0	0	0
$\mathbf{G1}_4$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	δ	δ	δ	δ	δ	δ	0	0	0	0	0	0
$\mathbf{G1}_5$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	δ	δ	δ	δ	δ	δ	0	0	0
$\mathbf{G2}_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	δ	δ	δ	δ	δ	δ
$\mathbf{G2}_1'$	0	0	0	0	0	0	0	0	0	δ	δ	δ	δ	δ	δ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\mathbf{G2}_2$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	δ	δ	δ	δ	δ	δ	0
$\mathbf{G2}_2'$	0	0	0	0	0	0	0	0	δ	δ	δ	δ	δ	δ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\mathbf{G2}_3$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	δ	δ	δ	δ	δ	δ	0	0
$\mathbf{G2}_3'$	0	0	0	0	0	0	0	δ	δ	δ	δ	δ	δ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\mathbf{G2}_4$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	δ	δ	δ	δ	δ	δ	0	0	0
$\mathbf{G2}_4'$	0	0	0	0	0	0	δ	δ	δ	δ	δ	δ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\mathbf{G2}_5$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	δ	δ	δ	δ	δ	δ	0	0	0	0
$\mathbf{G2}_5'$	0	0	0	0	0	δ	δ	δ	δ	δ	δ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Final state	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	δ	δ	δ	δ	δ	δ	0	0	0	0	0
Digest		0	0	0	0											0	0	0	0											

8.1 Free-start collision attack for wF-256

The designers of Fugue [6, Sec. 14] proved that **wF**-256 is provably free-start collision resistant upto 2^{96} by considering differentials with message differences that collide for the hash function through an internal state collision at the end of the **R** transform. In this section, we show a free-start collision attack on **wF**-256 for a reduced complexity of 2^{70} by finding partial collisions on the internal state at the end of the **R** transform that extend to produce a collision for the hash function provided some other internal conditions hold.

Our attack uses the observation (also noted in [6, Table 7.p 47]) that the digest of **wF-256** does not depend on the differences in the words $S_{9\sim14}$ of the internal state at the start of the **G** transform, showing that this transform is easily distinguishable from a random oracle. Hence, it is sufficient to show a free-start partial collision for the internal state of the **G** transform such that the words $S_{0\sim8}$ and $S_{15\sim29}$ collide and the words $S_{9\sim14}$ partially collide (i.e these words have some difference). The propagation of differences, denoted with δ, in the words $S_{9\sim14}$ through the **G** transform and its extension to one round of the **R** transform are shown in Table 1.

The designers of Fugue considered an adversary which finds a collision for **wF**-256 if it can find a $\delta S^{-5} = S'^{-5} \oplus S^{-5}$ and $\delta m^{-5} \sim \delta m^0$ such that $\delta S^0 \neq 0$ and the starting state of the **G** transform has zero difference, without control over the exact values of S^{-5} and S'^{-5}. We improve this attack on **wF**-256 by using the distinguisher in Table 1 on the composition of round 0 of **R** and the **G** transform. Our adversary wins the game of finding a collision in **wF**-256 if it can find δS^{-5} and $\delta m^{-5} \sim \delta m^0$ such that $\delta S^0 \neq 0$ and $\delta S^0_{0\sim5} = 0$ and $\delta S^0_{12\sim29} = 0$, without control over the exact values of S^{-5} and S'^{-5}. This implies a partial collision on the internal state after the **R** transform where there is a collision for the words $S_{0\sim8}$ and $S_{15\sim29}$ and differences in the remaining words. In our attack, we start with $\delta S^0_{0\sim5} = 0$ and $\delta S^0_{12\sim29} = 0$ at the start of round

0 and proceed *backwards* by inverting $\mathbf{R}$ transforms until round -6 where all words have a difference. Below we outline our analysis between the rounds 0 and -5 and the manner in which the internal state is affected up to round -6 has been illustrated in Table 4 of Appendix C. Our attack makes uses of a differential property of $\overline{\mathbf{SMIX}}$ that the number of *active* **SBox**es in an $\overline{\mathbf{SMIX}}$ function can be decreased by increasing the number of non-zero difference words at the input of $\overline{\mathbf{SMIX}}$ [6, §7].

Round 0: In this round, we maintain $\delta m^0 = 0$ so that the state differences after the $\overline{\mathbf{TIX}}_0$ operation do not extend to more words except that the words get shifted to the left by 3 words. We also do not inject any differences into the words S_0^0 and S_4^0 to ensure that $\overline{\mathbf{SMIX}}_{-1}$ remains *inactive*.

Round -1**:** In this round, the input difference to $\overline{\mathbf{SMIX}}_{-1}$ is zero leading to a zero output difference. However, $\overline{\mathbf{CMIX}}_{-1}$ propagates the differences to some other words. The $\overline{\mathbf{ROR3}}_{-1}$ operation makes the differences to affect the input of $\overline{\mathbf{SMIX}}_{-2}$. Hence, if $\delta S_6^0 \neq 0$ then $\overline{\mathbf{SMIX}}_{-2}$ would be *active*. To decrease the number of *active* **SBox**es at the output of $\overline{\mathbf{SMIX}}_{-2}$, we should increase the number of non-zero difference words at the input of $\overline{\mathbf{SMIX}}_{-2}$. In addition, when we inject differences into the state words, we have to ensure that the state difference δS^0 from which we have started is maintained at round 0. Hence, we inject differences into the words S_0^{-1} and S_4^{-1} such that $\delta S_0^{-1} \oplus \delta S_4^{-1} = \delta S_7^0$. Therefore, $\overline{\mathbf{SMIX}}_{-2}$ would be a $C_{0,3}$ code.

Round -2**:** We can inject differences through $\overline{\mathbf{TIX}}_{-2}$ such that all input words to $\overline{\mathbf{SMIX}}_{-3}$ have non zero differences. We inject differences into the words S_0^{-2} and S_4^{-2} in the round $\overline{\mathbf{TIX}}_{-2}$ such that $\delta S_0^{-2} \oplus \delta S_4^{-2} = \delta S_{10}^0$. Since all input words to $\overline{\mathbf{SMIX}}_{-3}$ have a difference, it is a $C_{0,1,2,3}$ code.

Round -3**:** The input word S_3^{-3} to $\overline{\mathbf{SMIX}}_{-4}$ is the same as the word S_{12}^0 and hence, $\delta S_3^{-3} = 0$ as $\delta S_{12}^0 = 0$. The words S_1^{-3} and S_2^{-3} have a non-zero difference. Hence, we inject differences through $\overline{\mathbf{TIX}}_{-3}$ such that we get a $C_{0,1,2}$ code for $\overline{\mathbf{SMIX}}_{-4}$. The injected differences into the words S_0^{-3} and S_4^{-3} must be the same (i.e, $\delta S_0^{-3} = \delta S_4^{-3}$) and hence, we refer to only δS_0^{-3} in this analysis.

Round -4**:** The input word S_3^{-4} to $\overline{\mathbf{SMIX}}_{-5}$ is the same as the word S_{15}^0 and hence, $\delta S_3^{-4} = 0$ as $\delta S_{15}^0 = 0$. The words S_1^{-4} and S_2^{-4} have a non-zero difference. Hence, we inject differences through $\overline{\mathbf{TIX}}_{-4}$ such that we get a $C_{0,1,2}$ code for $\overline{\mathbf{SMIX}}_{-5}$. The injected differences must satisfy the condition $\delta S_0^{-4} = \delta S_4^{-4}$. We refer to only δS_0^{-4} in our analysis.

Round -5**:** At the start of round -5, we have differences in all words of S^{-5} except the three words S_{11}^{-5}, S_{15}^{-5} and S_{16}^{-5}.

Following the above analysis and the resultant state difference δS^{-5} as shown in Table 4, the following A, B and C conditions must be satisfied for a collision in **wF**-256.

1. A:
   (a) $\delta S_{11}^{-5} = 0, \delta S_{15}^{-5} = 0$ and $\delta S_{16}^{-5} = 0$
   (b) $\delta S_1^{-5} = \delta S_{12}^{-5}$
   (c) $\delta S_2^{-5} = \delta S_{13}^{-5}$
2. B:
   (a) $\delta m^{-5} = \delta S_8^{-5}$
   (b) $\delta m^{-4} = \delta S_5^{-5} \oplus \delta S_9^0$
   (c) $\delta m^{-3} = \delta S_{13}^{-5} \oplus \delta S_{17}^{-5}$
   (d) $\delta m^{-2} = \delta S_{10}^{-5}$
   (e) $\delta m^{-1} = 0$
   (f) $\delta m^0 = 0$
3. C:
   (a) $\delta \hat{x}_{0\cdots3}^{-5} = \mathbf{N}^{-1}.(\delta S_0^{-4}, \delta S_0^{-3}, \delta x_3^{-2}, 0)$ where $\delta S_0^{-4} = \delta S_1^{-5}, \delta S_0^{-3} = \delta S_9^{-5}, \delta x_3^{-2} = \delta S_{10}^{-5}$

(b) $\delta\hat{x}^{-4}_{0\cdots3} = \mathbf{N}^{-1}.(\delta S^{-3}_0, \delta S^{-2}_4, \delta S^0_{11}, 0)$ where $\delta S^{-3}_0 = \delta S^{-5}_9, \delta S^{-2}_4 = \delta S^{-5}_6, \delta S^0_{11} = \delta S^{-5}_7$
(c) $\delta\hat{x}^{-3}_{0\cdots3} = \mathbf{N}^{-1}.(\delta S^{-2}_0, \delta S^{-1}_4, \delta S^0_8, \delta S^0_9)$ where $\delta S^{-2}_0 =?, \delta S^{-1}_4 = \delta S^{-5}_3, \delta S^0_8 =?, \delta S^0_9 = \delta S^{-5}_5 \oplus \delta m^{-4}$
(d) $\delta\hat{x}^{-2}_{0\cdots3} = \mathbf{N}^{-1}.(\delta S^{-1}_0, 0, 0, \delta S^0_6)$ where $\delta S^{-1}_0 \neq 0, \delta S^0_6 = \delta S^{-5}_{17}$.

Our adversary runs the following attack to find a collision in **wF**-256:

1. The attacker freely chooses a state difference δS^{-5} at *Round -5* such that the condition A is satisfied.
2. The state S^{-5} and the inputs $m^{-5} \sim m^{-2}$ are chosen freely at random and the words m^{-1} are m^0 are used to represent the length encoding of the 4-word message $m^{-5} \sim m^{-2}$. Then, S'^{-5} and input message words $m'^{-5} \sim m'^0$ are chosen in such a way that $S'^{-5} = \delta S^{-5} \oplus S^{-5}$ and the condition B are satisfied.
3. The attacker is successful if the resulting state at the start of *Round 0* satisfies $\delta S^0 \neq 0$ and $\delta S^0_{0\sim5} = 0$ and $\delta S^0_{12\sim29} = 0$.

This collision attack in which an adversary can freely decide δS^{-5} is called a free-start collision attack. In Theorem 1, we show the complexity of the free-start differential characteristic on **wF**-256 together with some other internal conditions required for the collision.

Theorem 1. *The success probability of the adversary to mount a free-start collision attack on* **wF**-*256 is not less than* 2^{-70} *provided other favorable internal conditions hold.*

Proof. Since the adversary can choose δS^{-5}, it can select it in such a way that conditions A and B are satisfied with a probability of 1. Hence, the success probability of the adversary is bounded by its ability to satisfy condition C. We estimate this probability by referring to the designers' analysis [6, Table 3,§7] of computation of the rank and minimum weight for all the linear codes used in their differential analysis of **F**-256 and **wF**-256. The success probability to satisfy conditions C is as follows:

C.a: An adversary must hit a code word in $C_{0,1,2}$. The code $C_{0,1,2}$ has full rank for 1 byte. That means it is impossible to get one *active* byte in $\overline{\mathbf{N}}$ transform. It will have at least 2 *active* bytes. Hence, the probability of receiving the desired difference after $\overline{\mathbf{SBox}}$ for the word S_3 is 2^{-14}.

C.b: An adversary must hit a code word in $C_{0,1,2}$ with at least 2^{-14} probability.

C.c: An adversary must hit a code word in $C_{0,1,2,3}$. We consider only one byte difference at the code word. Hence we have one *active* **SBox** and the probability would be 2^{-7}.

C.d: To satisfy this condition, an adversary must hit a code word in $C^*_{0,3}$ where not only do words 1 and 2 have to be zero but the 3rd word S^{-2}_3 in the $\mathbf{SMIX}_2$ transform (forward direction) must match $S^{-2}_{14} = \delta S^0_6$. Hence, the adversarial success probability to satisfy the condition C.d is 2^{-35} provided this additional internal constraint is satisfied.

Finally, the lower bound of the probability, Pr_C, to satisfy the condition C is $Pr_C \geq 2^{-14} \times 2^{-14} \times 2^{-7} \times 2^{-35} = 2^{-70}$ which is also a lower bound probability for the free-start collision differential characteristic on **wF**-256. Hence, the success probability of the adversary to mount a free-start collision attack on **wF**-256 is not less than 2^{-70} provided other favorable internal conditions as noted in C.d hold. □

9 Concluding remarks

In this paper, we analysed the security of **F**-256 and **wF**-256 hash functions and improved some of the previous analytical results. Although our new analytical results do not compromise the security claims of the designers for finding collisions and (second) preimages in these designs they show several interesting properties of these designs such as weak diffusion for the full **F**-256 due to the pseudo differentiability attack, ability to control message and state words to improve the meet-in-the-middle attack on **F**-256 and free-start collision attack on **wF**-256 under some constraints on the internal conditions.

Acknowledgments: We would like to thank Charanjit Jutla and Shai Halevi for their many valuable discussions and comments on the analysis presented in this paper. Praveen Gauravaram is supported by the Danish Council for Independent Research: Technology and Production Sciences (FTP) and Natural Sciences (FNU) grant number 274-09-0096. The work in this paper has been supported in part by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II.

References

1. J.-P. Aumasson and R. C.-W. Phan. Distinguisher for Full Final Round of Fugue-256. Presented at second NIST SHA-3 conference, 2010.
2. J.-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgård Revisited: How to construct a Hash Function. In V. Shoup, editor, *Advances in Cryptology—CRYPTO '05*, volume 3621 of *LNCS*, pages 430–448. Springer, 2005.
3. J. Daemen and V. Rijmen. *The design of Rijndael: AES — the Advanced Encryption Standard*. Springer, 2002.
4. I. Damgård. A Design Principle for Hash Functions. In G. Brassard, editor, *Advances in Cryptology: CRYPTO'89*, volume 435 of *LNCS*, pages 416–427. Springer-Verlag, 1989.
5. S. Halevi, W. E. Hall, and C. S. Jutla. Analysis of Fugue-256. Comment to NIST's hash function forum on April 4, 2010.
6. S. Halevi, W. E. Hall, and C. S. Jutla. The Hash Function Fugue. Submission to NIST (updated), 2009.
7. D. Khovratovich. Cryptanalysis of hash functions with structures. In M. J. J. Jr., V. Rijmen, and R. Safavi-Naini, editors, *Selected Areas in Cryptography*, volume 5867 of *Lecture Notes in Computer Science*, pages 108–125. Springer, 2009.
8. L. Knudsen and D. Wagner. Integral cryptanalysis. *Lecture Notes in Computer Science*, 2365:112–127, 2002.
9. L. R. Knudsen, C. Rechberger, and S. S. Thomsen. The Grindahl Hash Functions. In A. Biryukov, editor, *FSE*, volume 4593 of *Lecture Notes in Computer Science*, pages 39–57. Springer, 2007.
10. S. Lucks. A Failure-Friendly Design Principle for Hash Functions. In B. Roy, editor, *Advances in Cryptology - ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 474–494. Springer-Verlag, 2005.
11. U. M. Maurer, R. Renner, and C. Holenstein. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In M. Naor, editor, *First Theory of Cryptography Conference, TCC*, volume 2951 of *LNCS*, pages 21–39. Springer, 2004.
12. R. Merkle. One way Hash Functions and DES. In G. Brassard, editor, *Advances in Cryptology: CRYPTO 89*, volume 435 of *LNCS*, pages 428–446. Springer-Verlag, 1989.
13. NIST. Announcing the Development of New Hash Algorithms for the Revision of Federal Information Processing Standard (FIPS) 180-2, Secure Hash Standard, January 2007. This notice by NIST is available at `http://www.csrc.nist.gov/pki/HashWorkshop/timeline.html` with the Docket No: 061213336-6336-01. (Accessed on 2/11/2010).
14. NIST. Second Round Candidates. Official notification from NIST, 2009. Available at `http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/submissions_rnd2.html` (Accessed on 2/11/2010).

A Examples for the pseudo differentiator on unpadded F-256

The two pairs of *pseudo initial states* for the unpadded **F**-256 hash function noted in Table 2 would produce a digest difference of
0x $d6c75ee1$ $5773d022$ $2b886ddd$ $3485f8c2$ $12d821dd$ $c5e69851$ $1363d2d2$ $8d0b3459$ when the message word is $9f18d9d8$ and $f5821e47$ respectively. The first pair of states differ in 79 bits and the second pair of states differ in 83 bits which is close to the average approximation of 80 bits input state difference.

Table 2. Two pairs of *pseudo initial states* that produce the same digest difference

Word index i	S_i	S_i^*	δS_i	S_i	S_i^*	δS_i
0	00000000	00000000	00000000	00000000	00000000	00000000
1	f72586a6	f72586a6	00000000	80aec8c3	80aec8c3	00000000
2	cdf80275	cdf80275	00000000	c53ef91c	c53ef91c	00000000
3	447f68b7	447f68be	00000009	8f962fa4	8f962fad	00000009
4	71e74696	21a592b1	5042d427	3fdec35a	6f9c177d	5042d427
5	f3d2fde1	f3d2fde1	00000000	6c677fc1	6c677fc1	00000000
6	185ea4fa	185ea4fa	00000000	b82cdc7c	b82cdc7c	00000000
7	40ba2add	40ba2add	00000000	eca2a5a9	eca2a5a9	00000000
8	44aeeae4	44aeeae4	00000000	3b7f33da	3b7f33da	00000000
9	84dbc7c0	84dbc7c0	00000000	e8f8316f	e8f8316f	00000000
10	0a944675	0a944675	00000000	fdfb60fe	fdfb60fe	00000000
11	b082bccf	b082bccf	00000000	614fbd89	614fbd89	00000000
12	bd254e62	bd254e62	00000000	4d3e3233	4d3e3233	00000000
13	c8442168	c8442168	00000000	aab0153f	aab0153f	00000000
14	504fcbff	504fcbf6	00000009	dab86888	dab86881	00000009
15	e0842a26	e0842a26	00000000	ee576228	ee576228	00000000
16	6c55cbb5	6c55cbb5	00000000	90826388	90826388	00000000
17	8a4ee3a1	8a4ee3a1	00000000	151c8030	151c8030	00000000
18	0afaa726	0afaa72f	00000009	3a5aae42	3a5aae4b	00000009
19	42f9cf1e	12bb1b39	5042d427	89268f9a	d9645bbd	5042d427
20	78b1c37a	78b1c37a	00000000	8d7cdce1	8d7cdce1	00000000
21	b4f6f7b7	244bdabc	90bd2d0b	a297636e	9b8d8a8f	391ae9e1
22	71b019fc	93aa923a	e21a8bc6	be289273	895dad7b	37753f08
23	8bcd6ca5	fd547b2e	7699178b	a8a66154	d714dddf	7fb2bc8b
24	f15f542c	f15f542c	00000000	0190a25b	0190a25b	00000000
25	4b9729b1	4b9729b1	00000000	895989cd	895989cd	00000000
26	cd8f75fa	cd8f75fa	00000000	b137416b	b137416b	00000000
27	f76c7a15	f76c7a15	00000000	314c1a96	314c1a96	00000000
28	b3b63d8f	b3b63d8f	00000000	4d6740dd	4d6740dd	00000000
29	7229ffbd	7229ffb4	00000009	8a0d14ab	8a0d14a2	00000009

B Improved pseudo differentiator for F-256

We can combine the pseudo differentiability attack on the unpadded **F**-256 and techniques used in the improved meet-in-the-middle attack on **F**-256 to improve the pseudo differentiability attack on full **F**-256 of §5. Recall that $0, -1, -2, \ldots$ are the rounds of **R** before the **G** transform and $m^0, m^{-1}, \ldots$ are their respective message blocks. Recall that *State-i'* is the 29-word internal state (from word 1 to 29) in any round i of the **R** transform after the step $S_{10}^i = S_0^i \oplus S_{10}^i$ in the *forward process*. A pair of states at *State-i'* of **R** are denoted by $(S[i], S[i]')$ where $S[i] = S[i]_{1\sim 29}$ and $S[i]' = S[i]'_{1\sim 29}$. The attack is outlined below:

1. Use the freedom available in the state words $S_{8\sim 14}$ and $S_{22\sim 29}$ of the pseudo differentiator for the unpadded **F**-256 to find a pair of states $(S[0], S[0]')$ such that their common message block is $m^0 = 0x00000200$, which is the last length encoded word of a 32-bit message. The cost of this step is 2^{32} calls to the **Backwards from G2**$_{11}$ step of the pseudo differentiator for the unpadded **F**-256. Note that by now the words $S[0]_{1\sim 29}$ and $S[0]'_{1\sim 29}$ in these pair of states are fixed. This implies the words $S^0_{1\sim 9}$ and $S^0_{11\sim 29}$ are also fixed and the words S^0_0 and S^0_{10} are not fixed.
2. Now we use the freedom in the word S^0_0 to find a pair of states (S^{-1}, S'^{-1}) that have the same message block $m^{-1} = 0x00000000$ in the -1^{th} **R** transform. We do this as follows:
   (a) The word S^0_4 which has the difference δ_2 will also be the 1st word input to the second $\overline{\textbf{SMIX}}$ transform (in the *backward process*). This $\overline{\textbf{SMIX}}$ transform also has the non-zero

difference at input word 0 which is the same the output of word 3 of the first $\overline{\textbf{SMIX}}$ transform of -1 round **R** transform. Moreover, the actual values of the input words 2 and 3 to the second $\overline{\textbf{SMIX}}$ transform are already fixed and they are zero difference words. Now we find the difference in the input word 0 (say Δ) for the second $\overline{\textbf{SMIX}}$ transform such that we obtain the 3rd output word of this second $\overline{\textbf{SMIX}}$ to be $0x00000000$. This can be done by solving a system of equations.

(b) Having determined the difference Δ for the input word 0 to the second $\overline{\textbf{SMIX}}$ transform of -1 round **R** transform, we can use the freedom in the word input S_0^0 of the first $\overline{\textbf{SMIX}}$ transform of -1 round **R** transform to force its output 3rd word to Δ difference. Note that the word S_3^0 has δ_1 difference and the words $S_{1\sim2}^0$ have zero difference with their actual values already determined. This can be done by solving a system of equations.

3. Having obtained a pair of 29-word states $(S[-1], S[-1]')$ with the common message $m^{-1} = 0x00000000$, we choose a pair $(S_0^1, S_0^{1'})$ and $(S_{10}^1, S_{10}^{1'})$ such that $S_0^1 \oplus S_{10}^1 = S[-1]_{10}$ and $S_0^{1'} \oplus S_{10}^{1'} = S[-1]'_{10}$. Note that since $\delta S_{10}^1 = 0$, $S_{10}^1 = S_{10}^{1'}$ and $S_0^1 = S_0^{1'}$. Now we have determined a pair of 30-word states $(S^1, S^{1'})$.
4. Invert the 30-word states $(S^1, S^{1'})$ for one more round of the **R** transform and obtain the message block m^{-2} and the states (S^{-2}, S'^{-2}).

In summary, we have shown a method which can find a pair of states that differ on average in 332 bits for the *initial state* of **F**-256 and producing the fixed digest difference. We can find another pair of *initial states* by repeating the attack using the freedom available in the words in the Step 1 of this attack and this pair will also produce the same digest difference as the former. The cost of the attack is about 2^{33} hashing operations.

C Integrals for the 16.5 rounds of G transform of F-256 and differential characteristics for free-start collision attack on wF-256

In Table 3, we have shown the integrals for the 16.5 rounds of the **G** transform of **F**-256 whose analysis was presented in §7. In Table 4, we have shown how the internal state of **wF**-256 gets affected between rounds 0 and -6 in the free-start collision attack on **wF**-256 whose analysis was shown in §8.

Table 3. The Integral propagation over 16.5 rounds of **F**-256 **G**-function. In this table "$.$", "$,$" and "!" denote $\mathcal{C}$, $\mathcal{A}$ and $\mathcal{S}$ respectively and blank cells are those cells that we do not consider.

	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}	S_{13}	S_{14}	S_{15}	S_{16}	S_{17}	S_{18}	S_{19}	S_{20}	S_{21}	S_{22}	S_{23}	S_{24}	S_{25}	S_{26}	S_{27}	S_{28}	S_{29}
G_1^1	.	.	.	.	.,..	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
$G_1^{1.5}$	.	.	.	.	.	.	.	.,..	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
G_1^2	.	.	.	.	.	.	.	.	.	.	.,..	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
$G_1^{2.5}$	.	.	.	.	.	.	.	.	.	.	.	.	.	.,..	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
G_1^3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.,..	.	.	.	.	.	.	.	.	.	.	.	.	.
$G_1^{3.5}$	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.,..	.	.	.	.	.	.	.	.	.	.
G_1^4	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.,..	.	.	.	.	.	.	.
$G_1^{4.5}$	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.,..	.	.	.	.
G_1^5	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.,..	.
$G_1^{5.5}$	.,..	,...	...,	..,.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
G_2^1	,!,!	,!,!	,,,,	,!,!	,...	...,	..,.	.	.	.	.	.	.	.	.	,...	...,	..,.	.	.	.	.	.	.	.	.	.	.	.	.
$G_2^{1.5}$	??!?	,?,?	,???	.?!?	.	.	.	.	.	.	.	.	.	.	.	,!,!	,!,!	,,,,	,!,!	!!,!	...,	..,.	.	.	.	.	.	.	.	.
G_2^2															??!?															
$G_2^{2.5}$																														??!?
G_2^3														??!?																
$G_2^{3.5}$																													??!?	
G_2^4													??!?																	
$G_2^{4.5}$																												??!?		
G_2^5												??!?																		
$G_2^{5.5}$																											??!?			
G_2^6											??!?																			
$G_2^{6.5}$																										??!?				
G_2^7										??!?																				
$G_2^{7.5}$																									??!?					
G_2^8									??!?																					
$G_2^{8.5}$																								??!?						
G_2^9								??!?																						
$G_2^{9.5}$																							??!?							
G_2^{10}							??!?																							
$G_2^{10.5}$																						??!?								
G_2^{11}						??!?																								
$G_2^{11.5}$																					??!?									
G_2^{12}					??!?																									
$G_2^{12.5}$																				??!?										

Table 4. Differential characteristic demonstrating a free-start collision attack on the wF-256 hash function.

Input of	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
$\mathbf{TIX}_0^{\delta m^0=0}$	0	0	0	0	0	0	δS_6^0	δS_7^0	δS_8^0	δS_9^0	δS_{10}^0	δS_{11}^0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\mathbf{SMIX}_{-1}$	0	0	0	0	0	0	δS_6^0	δS_7^0	δS_8^0	δS_9^0	δS_{10}^0	δS_{11}^0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\mathbf{CMIX}_{-1}$	0	0	δS_6^0	0	0	0	δS_6^0	δS_7^0	δS_8^0	δS_9^0	δS_{10}^0	δS_{11}^0	0	0	0	0	0	δS_6^0	0	0	0	0	0	0	0	0	0	0	0	0
$\mathbf{ROR3}_{-1}$	0	0	0	δS_6^0	δS_7^0	δS_8^0	δS_9^0	δS_{10}^0	δS_{11}^0	0	0	0	0	0	δS_6^0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	δS_6^0
$\mathbf{TIX}_{-1}^{\delta m^{-1}=0}$	δS_0^{-1}	0	0	δS_6^0	δS_4^{-1}	δS_8^0	δS_9^0	δS_{10}^0	δS_{11}^0	0	0	0	0	0	δS_6^0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	δS_6^0
$\mathbf{SMIX}_{-2}$	δx_0^{-2}	δx_1^{-2}	δx_2^{-2}	δx_3^{-2}	δS_4^{-1}	δS_8^0	δS_9^0	δS_{10}^0	δS_{11}^0	0	0	0	0	0	δS_6^0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	δS_6^0
$\mathbf{CMIX}_{-2}$	$\delta x_0'^{-2}$	$\delta x_1'^{-2}$	$\delta x_2'^{-2}$	δx_3^{-2}	δS_4^{-1}	δS_8^0	δS_9^0	δS_{10}^0	δS_{11}^0	0	0	0	0	0	δS_6^0	δS_4^{-1}	δS_8^0	δS_9^0	0	0	0	0	0	0	0	0	0	0	0	δS_6^0
$\mathbf{ROR3}_{-2}$	δx_3^{-2}	δS_4^{-1}	δS_8^0	δS_9^0	δS_{10}^0	δS_{11}^0	0	0	0	0	0	δS_6^0	δS_4^{-1}	δS_8^0	δS_9^0	0	0	0	0	0	0	0	0	0	0	0	δS_6^0	$\delta x_0'^{-2}$	$\delta x_1'^{-2}$	$\delta x_2'^{-2}$
$\mathbf{TIX}_{-2}^{\delta m^{-2}=\delta x_3^{-2}}$	δS_0^{-2}	δS_4^{-1}	δS_8^0	δS_9^0	δS_4^{-2}	δS_{11}^0	0	0	δx_3^{-2}	0	0	δS_6^0	δS_4^{-1}	δS_8^0	δS_9^0	0	0	0	0	0	0	0	0	0	0	0	δS_6^0	$\delta x_0'^{-2}$	$\delta x_1'^{-2}$	$\delta x_2'^{-2}$
$\mathbf{SMIX}_{-3}$	δx_0^{-3}	δx_1^{-3}	δx_2^{-3}	δx_3^{-3}	δS_4^{-2}	δS_{11}^0	0	0	δx_3^{-2}	0	0	δS_6^0	δS_4^{-1}	δS_8^0	δS_9^0	0	0	0	0	0	0	0	0	0	0	0	δS_6^0	$\delta x_0'^{-2}$	$\delta x_1'^{-2}$	$\delta x_2'^{-2}$
$\mathbf{CMIX}_{-3}$	$\delta x_0'^{-3}$	$\delta x_1'^{-3}$	δx_2^{-3}	δx_3^{-3}	δS_4^{-2}	δS_{11}^0	0	0	δx_3^{-2}	0	0	δS_6^0	δS_4^{-1}	δS_8^0	δS_9^0	δS_4^{-2}	δS_{11}^0	0	0	0	0	0	0	0	0	0	δS_6^0	$\delta x_0'^{-2}$	$\delta x_1'^{-2}$	$\delta x_2'^{-2}$
$\mathbf{ROR3}_{-3}$	δx_3^{-3}	δS_4^{-2}	δS_{11}^0	0	0	δx_3^{-2}	0	0	δS_6^0	δS_4^{-1}	δS_8^0	δS_9^0	δS_4^{-2}	δS_{11}^0	0	0	0	0	0	0	0	0	0	δS_6^0	$\delta x_0'^{-2}$	$\delta x_1'^{-2}$	$\delta x_2'^{-2}$	$\delta x_0'^{-3}$	$\delta x_1'^{-3}$	δx_2^{-3}
$\mathbf{TIX}_{-3}^{\delta m^{-3}=\delta x_3^{-3}}$	δS_0^{-3}	δS_4^{-2}	δS_{11}^0	0	δS_0^{-3}	δx_3^{-2}	0	0	δS_8^{-3}	δS_4^{-1}	δS_8^0	δS_9^0	δS_4^{-2}	δS_{11}^0	0	0	0	0	0	0	0	0	0	δS_6^0	$\delta x_0'^{-2}$	$\delta x_1'^{-2}$	$\delta x_2'^{-2}$	$\delta x_0'^{-3}$	$\delta x_1'^{-3}$	δx_2^{-3}
$\mathbf{SMIX}_{-4}$	δx_0^{-4}	δx_1^{-4}	δx_2^{-4}	δx_3^{-4}	δS_0^{-3}	δx_3^{-2}	0	0	δS_8^{-3}	δS_4^{-1}	δS_8^0	δS_9^0	δS_4^{-2}	δS_{11}^0	0	0	0	0	0	0	0	0	0	δS_6^0	$\delta x_0'^{-2}$	$\delta x_1'^{-2}$	$\delta x_2'^{-2}$	$\delta x_0'^{-3}$	$\delta x_1'^{-3}$	δx_2^{-3}
$\mathbf{CMIX}_{-4}$	$\delta x_0'^{-4}$	$\delta x_1'^{-4}$	δx_2^{-4}	δx_3^{-4}	δS_0^{-3}	δx_3^{-2}	0	0	δS_8^{-3}	δS_4^{-1}	δS_8^0	δS_9^0	δS_4^{-2}	δS_{11}^0	0	δS_0^{-3}	δx_3^{-2}	0	0	0	0	0	0	δS_6^0	$\delta x_0'^{-2}$	$\delta x_1'^{-2}$	$\delta x_2'^{-2}$	$\delta x_0'^{-3}$	$\delta x_1'^{-3}$	δx_2^{-3}
$\mathbf{ROR3}_{-4}$	δx_3^{-4}	δS_0^{-3}	δx_3^{-2}	0	0	δS_8^{-3}	δS_4^{-1}	δS_8^0	δS_9^0	δS_4^{-2}	δS_{11}^0	0	δS_0^{-3}	δx_3^{-2}	0	0	0	0	0	0	δS_6^0	$\delta x_0'^{-2}$	$\delta x_1'^{-2}$	$\delta x_2'^{-2}$	$\delta x_0'^{-3}$	$\delta x_1'^{-3}$	δx_2^{-3}	$\delta x_0'^{-4}$	$\delta x_1'^{-4}$	δx_2^{-4}
$\mathbf{TIX}_{-4}^{\delta m^{-4}=\delta x_3^{-4}}$	δS_0^{-4}	δS_0^{-3}	δx_3^{-2}	0	δS_0^{-4}	δS_8^{-3}	δS_4^{-1}	δS_8^0	δS_8^{-4}	δS_4^{-2}	δS_{11}^0	0	δS_0^{-3}	δx_3^{-2}	0	0	0	0	0	0	δS_6^0	$\delta x_0'^{-2}$	$\delta x_1'^{-2}$	$\delta x_2'^{-2}$	$\delta x_0'^{-3}$	$\delta x_1'^{-3}$	δx_2^{-3}	$\delta x_0'^{-4}$	$\delta x_1'^{-4}$	δx_2^{-4}
$\mathbf{SMIX}_{-5}$	δx_0^{-5}	δx_1^{-5}	δx_2^{-5}	δx_3^{-5}	δS_0^{-4}	δS_8^{-3}	δS_4^{-1}	δS_8^0	δS_8^{-4}	δS_4^{-2}	δS_{11}^0	0	δS_0^{-3}	δx_3^{-2}	0	0	0	0	0	0	δS_6^0	$\delta x_0'^{-2}$	$\delta x_1'^{-2}$	$\delta x_2'^{-2}$	$\delta x_0'^{-3}$	$\delta x_1'^{-3}$	δx_2^{-3}	$\delta x_0'^{-4}$	$\delta x_1'^{-4}$	δx_2^{-4}
$\mathbf{CMIX}_{-5}$	$\delta x_0'^{-5}$	$\delta x_1'^{-5}$	$\delta x_2'^{-5}$	δx_3^{-5}	δS_0^{-4}	δS_8^{-3}	δS_4^{-1}	δS_8^0	δS_8^{-4}	δS_4^{-2}	δS_{11}^0	0	δS_0^{-3}	δx_3^{-2}	0	δS_0^{-4}	δS_8^{-3}	δS_4^{-1}	0	0	δS_6^0	$\delta x_0'^{-2}$	$\delta x_1'^{-2}$	$\delta x_2'^{-2}$	$\delta x_0'^{-3}$	$\delta x_1'^{-3}$	δx_2^{-3}	$\delta x_0'^{-4}$	$\delta x_1'^{-4}$	δx_2^{-4}
$\mathbf{ROR3}_{-5}$	δx_3^{-5}	δS_0^{-4}	δS_8^{-3}	δS_4^{-1}	δS_8^0	δS_8^{-4}	δS_4^{-2}	δS_{11}^0	0	δS_0^{-3}	δx_3^{-2}	0	δS_0^{-4}	δS_8^{-3}	δS_4^{-1}	0	0	δS_6^0	$\delta x_0'^{-2}$	$\delta x_1'^{-2}$	$\delta x_2'^{-2}$	$\delta x_0'^{-3}$	$\delta x_1'^{-3}$	δx_2^{-3}	$\delta x_0'^{-4}$	$\delta x_1'^{-4}$	δx_2^{-4}	$\delta x_0'^{-5}$	$\delta x_1'^{-5}$	$\delta x_2'^{-5}$
$\mathbf{TIX}_{-5}^{\delta m^{-5}=\delta x_3^{-5}}$	δS_0^{-5}	δS_0^{-4}	δS_8^{-3}	δS_4^{-1}	δS_4^{-5}	δS_8^{-4}	δS_4^{-2}	δS_{11}^0	δx_3^{-5}	δS_0^{-3}	δx_3^{-2}	0	δS_0^{-4}	δS_8^{-3}	δS_4^{-1}	0	0	δS_6^0	$\delta x_0'^{-2}$	$\delta x_1'^{-2}$	$\delta x_2'^{-2}$	$\delta x_0'^{-3}$	$\delta x_1'^{-3}$	δx_2^{-3}	$\delta x_0'^{-4}$	$\delta x_1'^{-4}$	δx_2^{-4}	$\delta x_0'^{-5}$	$\delta x_1'^{-5}$	$\delta x_2'^{-5}$
Input of	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	s_{16}	s_{17}	s_{18}	s_{19}	s_{20}	s_{21}	s_{22}	s_{23}	s_{24}	s_{25}	s_{26}	s_{27}	s_{28}	s_{29}
$\mathbf{SMIX}_{-6}$	δx_0^{-6}	δx_1^{-6}	δx_2^{-6}	δx_3^{-6}	δS_4^{-5}	δS_8^{-4}	δS_4^{-2}	δS_{11}^0	δx_3^{-5}	δS_0^{-3}	δx_3^{-2}	0	δS_0^{-4}	δS_8^{-3}	δS_4^{-1}	0	0	δS_6^0	$\delta x_0'^{-2}$	$\delta x_1'^{-2}$	$\delta x_2'^{-2}$	$\delta x_0'^{-3}$	$\delta x_1'^{-3}$	δx_2^{-3}	$\delta x_0'^{-4}$	$\delta x_1'^{-4}$	δx_2^{-4}	$\delta x_0'^{-5}$	$\delta x_1'^{-5}$	$\delta x_2'^{-5}$
$\mathbf{CMIX}_{-6}$	$\delta x_0'^{-6}$	$\delta x_1'^{-6}$	$\delta x_2'^{-6}$	δx_3^{-6}	δS_4^{-5}	δS_8^{-4}	δS_4^{-2}	δS_{11}^0	δx_3^{-5}	δS_0^{-3}	δx_3^{-2}	0	δS_0^{-4}	δS_8^{-3}	δS_4^{-1}	δS_4^{-5}	δS_8^{-4}	δx_{17}^{-5}	$\delta x_0'^{-2}$	$\delta x_1'^{-2}$	$\delta x_2'^{-2}$	$\delta x_0'^{-3}$	$\delta x_1'^{-3}$	δx_2^{-3}	$\delta x_0'^{-4}$	$\delta x_1'^{-4}$	δx_2^{-4}	$\delta x_0'^{-5}$	$\delta x_1'^{-5}$	$\delta x_2'^{-5}$
$\mathbf{ROR3}_{-6}$	δx_3^{-6}	δS_4^{-5}	δS_8^{-4}	δS_4^{-2}	δS_{11}^0	δx_3^{-5}	δS_0^{-3}	δx_3^{-2}	0	δS_0^{-4}	δS_8^{-3}	δS_4^{-1}	δS_4^{-5}	δS_8^{-4}	δx_{17}^{-5}	$\delta x_0'^{-2}$	$\delta x_1'^{-2}$	$\delta x_2'^{-2}$	$\delta x_0'^{-3}$	$\delta x_1'^{-3}$	δx_2^{-3}	$\delta x_0'^{-4}$	$\delta x_1'^{-4}$	δx_2^{-4}	$\delta x_0'^{-5}$	$\delta x_1'^{-5}$	$\delta x_2'^{-5}$	$\delta x_0'^{-6}$	$\delta x_1'^{-6}$	$\delta x_2'^{-6}$
$\mathbf{TIX}_{-6}^{\delta m^{-6}=\delta x_3^{-6}}$	δS_0^{-6}	δS_4^{-5}	δS_8^{-4}	δS_4^{-2}	δS_4^{-6}	δx_3^{-5}	δS_0^{-3}	δx_3^{-2}	δx_3^{-6}	δS_0^{-4}	δS_8^{-3}	δS_4^{-1}	δS_4^{-5}	δS_8^{-4}	δx_{17}^{-5}	$\delta x_0'^{-2}$	$\delta x_1'^{-2}$	$\delta x_2'^{-2}$	$\delta x_0'^{-3}$	$\delta x_1'^{-3}$	δx_2^{-3}	$\delta x_0'^{-4}$	$\delta x_1'^{-4}$	δx_2^{-4}	$\delta x_0'^{-5}$	$\delta x_1'^{-5}$	$\delta x_2'^{-5}$	$\delta x_0'^{-6}$	$\delta x_1'^{-6}$	$\delta x_2'^{-6}$