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

Abstract

We define ideal functionalities that are weaker than ideal functionalities traditionally used in realizing variable input length (VIL) random oracles (RO) in the indifferentiability or universal-Composability (UC) model. We also show realization of VIL-RO using these weaker ideal functionalities, with applications to proving Fugue and CubeHash hash functions to be VIL-RO. We argue that components of Fugue realize this weaker ideal functionality using techniques employed in proving resistance of Fugue to differential collision-attacks. This should be contrasted with other hash functions that are proven VIL-RO assuming the components are extremely ideal, e.g. random permutations.

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 [7]. 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 potentially be securely realized by fixed input length (FIL) components that themselves do not give adversarial access to any inversion oracles. If one tries to build hash functions using FIL components which are required to be secure with adversarial access to inversion oracles, one may not get an optimal speed/security trade-off. 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 (VIL) hash function to be resistant to known cryptanalytic techniques like differential

*Contact author: csjutla@us.ibm.com

†Now at Fujitsu Labs of America.

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 about FIL components).

In this paper, we formalize this intuition, and show that one can build variable input length (VIL) hash functions, which are indifferentiable [8] from VIL Random Oracles (VIL-RO), from ideal functionalities that do not have inversion oracles. Currently, this ideal functionality is *itself* a VIL functionality, but in a forthcoming work, we will show how to do this with a FIL ideal functionality. Normally, this would not be a difficult task as traditionally these FIL ideal functionalities are assumed to be ideal to the extreme (e.g. a random permutation on 128 bits, or a random oracle on 128-bits, or an ideal cipher on 128 bits). But, we want to make these component ideal functionalities *as weak as possible*, so that one can actually argue that Fugue's (or some other hash function's) components 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 the component idealized functionality.

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. As mentioned earlier, we would like not to assume something strong like a FIL-RO. Given the focus of Fugue to avoid un-necessarily trying to defend against internal properties that are not needed in the full realization of a secure VIL-RO, one needs to take a different approach to proving 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, we do 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 [6] 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.

5. Finally, assuming the previous point, it follows from the UC Composition Theorem [5], or the composition theorem based on indifferentiability ([8]), that Fugue realizes VIL-RO.

1.1 Alternative Approach using One-way Functions

The above formulation of a weak ideal functionality implicitly assumes that there are no pre-image attacks, because it requires that the final transformation be fed with states which are produced by the input stage acting on some adversarially chosen message. If pre-image attacks were possible, then an adversary can always come up with an input message to achieve the required intermediate state, and hence it may not be possible to realize such an ideal functionality. Of course, pre-image attacks are usually not possible to achieve (and in fact most broken hash functions still tend to resist pre-image attacks), and that is usually the weak assumption that most hash functions make.

However, it may be interesting to base the security of a hash function on an even weaker assumption, which is the one-way function assumption. One-way function assumption is weaker than the pre-image assumption in that in the former the adversary must find an inverse on a random hash value (to be precise, in case of one-way permutations; but for treatment in this paper we will work with this definition even for one-way functions), as opposed to in the latter where an adversary may just need to show an inverse on some distribution of the hash value.

However, there are two difficulties here. First is a formulation problem, as it is difficult to define a stand alone ideal functionality for one-way functions. Thus, as before we must define a novel composite functionality. Second is a design in-efficiency, as this composite functionality would require a stronger ideal functionality for the final transformation part, i.e. that the final transformation be a random permutation by itself (i.e. without stealing some input rounds as is done in the case of Fugue, where one could do this as it came with a proof of improbability of obtaining partial collisions – in other words, even though Fugue's final transformation is not an ideal random permutation, it can still be proven secure in the model described in the Introduction).

This approach is described in detail in Section 6, and maybe applicable to designs such as CubeHash [2].

1.2 Related Work

. This work in this exact form was first submitted to the NIST SHA-3 website in 2010. Since that time, there have been other works, in particular due to Andreeva et al [1] and Bhattacharyya and Mandal [4], showing that Fugue can be seen as a variant sponge design [3]. However, these works [1, 4] continue to assume that the components have strong ideal functionalities.

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,
  - if t-fake is not supplied, let t be a random 960-bit value, otherwise let t be t-fake. 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 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, (i.e. if prefix-len is not zero) if $|m| \neq 8$, return $\perp$. Otherwise, set state to s, and iterate Fugue's input round R on input m (word by word), followed by the final Fugue transformation G, and return the final 256 bits.

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. Note that Fugue-$\mathcal{I}$ itself does not perform any padding. The padding is done by function *Hash*. Thus, a more precise description above would be to say that F-256 is used in the above formalization of both Fugue-$\mathcal{I}$-FINAL and Fugue-$\mathcal{I}$-PREFIX.

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

In the indifferentiability paradigm [8], 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 unless real Fugue returns a collision with a previous intermediate state. 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 [7] 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 outputs for messages of length less than 8, and starting from the Fugue IV, are random and (jointly) 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 final round stage G2 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, ($D = t_1 \oplus t_2 \neq 0$), 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 behavior. Note here that the adversary must know a message prefix p_1 which starting from the Fugue IV leads to an intermediate state t_1, and similarly for p_2 and t_2. One way to achieve this is to launch a forward differential attack, but such attacks are shown improbable. The other possibility is to do a pre-image attack, but that is usually considered to be even more difficult, and is usually taken as a weak assumption.

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.

6 Composite Functionality VIL-CROW with FIL-RP yields VIL-RO

In this section, we define a composite functionality using a variable input length function modeling collision resistant one-way functions (VIL-CROW), and a fixed input length random permutation (FIL-RP). We then show that VIL-RO (section 3) can be realized using this composite functionality.

Before we get into formalizing this, we remark that the indifferentiability result of Bertoni et al [3] about Sponge constructions claims that if all input rounds are random permutations, and the final round is a random permutation, and the input rounds do not yield internal collisions then one can realize a VIL-RO from these functionalities.

The claim here is stronger, as it requires the input round (though admittedly as a monolithic variable input length entity) to be just collision resistant and one-way. These are much easier to attain (or even prove under reasonable assumptions) than proving each input round to be random (which can come at the cost of relinquishing efficiency).

The random Oracle (RO) ideal functionality keeps a store T, as described in section 3.

A collision-resistant compression function (VIL-CR) ideal functionality keeps a store T as well, which is a table of entries, each a pair consisting of arbitrary length messages and (say) a 1024-bit intermediate state. On any input m of arbitrary length, it first checks to see if $\langle m, s\rangle$ is in T, and if so it returns s. Otherwise, it calls the adversary with the input m, who returns a value s-fake. If s-fake is equal to any other s (i.e. second component of any entry) in T, the functionality picks a random value s', stores $\langle m, s'\rangle$ in T and returns s'. Otherwise, it stores $\langle m, s\text{-fake}\rangle$ in T and returns s-fake.

The one-way function ideal functionality is tricky, as the usually understood definition is that on a randomly chosen output value, no adversary can come up with an input which would yield that value. Defining ideal functionality with this "randomly chosen" notion requires a composite functionality.

As a warm-up exercise, we first define the following ideal functionality called (VIL-OW-FIL-RO) where it displays two functions in its interface. The first function is called "Generate Random", and the second is called "Generate Oneway". The functionality maintains two stores: R and T.

- On the first function (Generate Random) invocation, on input m of length only 1024 bit or less, it behaves like a FIL-RO, that is returns a random value s of 1024-bits on new inputs, and saves $\langle m, s\rangle$ in R, .
- On the second function (Generate Oneway) invocation, on input m of variable input length, it calls the adversary and let's the adversary decide the output as long as it is not in table R. To be more precise, on input m, the functionality first checks to see if m is in store T, and if so returns the associated value. Otherwise, it calls the adversary with m who returns a 1024-bit value s-fake. If s-fake is the second component of any entry in R, then the functionality sets a temporary variable s to a new random 1024-bit value (i.e. ignore s-fake), otherwise s is set to s-fake. Next, it records $\langle m, s\rangle$ in T, and returns s.

Thus, the first function is just a FIL-RO, and the second is linked with the first, and thats how it models a one-way function. Essentially, in the oneway function security game, the adversary is given a random value and is asked to invert it. Since that random value must have come from some random oracle, we built it into the functionality.

Now we are ready to define the ideal functionality VIL-CROW-FIL-RP. This has three functions called invert-final, input-sponge, and finalize. It also has three tables, S, R, and O.

- Invert-final: on the first type of input x of length only 1024-bit or less it behaves like a FIL-RO but this time generates 1024 bit random value s (s for internal state) and saves it as R[x] = s, as well as O[s] =x...this models the inversion function of the final round random permutation.
- finalize: a finalize function takes 1024-bit input s (s for internal state), and ignores the input if s was not set as internal state before, i.e. it is not in table S. Else, if O[s] is defined it outputs that, else it picks a new 1024-bit random value x and outputs that. It saves O[s] = x, as well as R[x] =s.
- input-sponge: on the third type of input m of variable input length, it calls the adversary and lets it decide a 1024-bit value *s*-fake (for internal state), as long as it is not already in S or in R. If it is not (i.e. not in either of S or R), it saves S[m]=*s*-fake, otherwise it sets S[m] to a random new value. It then outputs S[m]. This makes it model both collision resistance and oneway-ness.

Now, it is easy to see that this is a random oracle when calls are made as follows: With input P, call input-sponge which returns, say s. Then call finalize with s, and get back r.

It is easy to see that this is a VIL-RO: On the first call adversary gets to set s, but it can never be set to something it set before as internal state (collision resistance) nor to some value one obtained by calling invert-final (onewayness). Now, before it calls finalize on s, an adversary can make several invert-final queries, but those will be random and unrelated to s with high probability. When the finalize call is made on s, it will not be in S or R with high probability, and hence a random value will be output.

Finally, for a real-world hash function, say CubeHash, one needs to show that one can define a real-world functionality with the above interface such that it is indifferentiable from the above ideal functionality VIL-CROW-FIL-RP. If one can make such a claim, then the hash function defined using this real world functionality is indifferentiable from VIL-RO. Thus, it entails showing that the final round of the real-world design is a random permutation (from the ideal definition of invert-final and finalize), and the input rounds are collision resistant and one-way (from the ideal definition of input-sponge).

References

[1] Elena Andreeva, Bart Mennink, Bart Preneel, The parazoa family: generalizing the sponge hash functions. Int. J. Inf. Sec., volume 11, number 3, pages 149–165, year 2012.

[2] D. Bernstein, CubeHash: a simple hash function. `http://cubehash.cr.yp.to`

[3] G. Bertoni, J. Daemen, M. Peeters and G. Van Assche, On the Indifferentiability of the Sponge Construction. Proc. Eurocrypt 2008.

[4] Bhattacharyya, R., Mandal, A.: On the indifferentiability of Fugue and Luffa. In 9th ACNS, vol. 6715, pp. 479-497. Jun 7-10, 2011.

[5] R. Cannetti, Universally Composable Security: A new Paradigm for Cryptographic Protocols, Proc. FOCS 2001.

[6] 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.

[7] Shai Halevi, Eric W. Hall, Charanjit S. Jutla, The Hash Function "Fugue", IACR Cryptology ePrint Archive 2014: 423 (2014).
`http://domino.research.ibm.com/comm/research_projects.nsf/pages/fugue.index.html`

[8] 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.