A Keyed Sponge Construction with Pseudorandomness in the Standard Model

Donghoon Chang[1], Morris Dworkin[1], Seokhie Hong[2], John Kelsey[1], and Mridul Nandi[3]

[1] National Institute of Standards and Technology (NIST), USA
pointchang@gmail.com, morris.dworkin@nist.gov, john.kelsey@nist.gov
[2] Center for Information Security Technologies (CIST), Korea University, Korea
shhong@korea.ac.kr
[3] Indian Statistical Institute (ISI), Kolkata, India
mridul.nandi@gmail.com

Abstract. The sponge construction, designed by Bertoni, Daemen, Peeters, and Asscheis, is the framework for hash functions such as Keccak, PHOTON, QUARK, and SPONGENT. The designers give a keyed sponge construction by prepending the message with key and prove a bound on its pseudorandomness in the ideal permutation model . In this paper we give a different keyed sponge construction that is based on the Even-Mansour permutation and prove its pseudorandomness in the standard model.
Key Words : Sponge Construction, Pseudorandomness, Indifferentiability.

1 Introduction

A hash function is a crucial component for cryptographic primitives such as message authentication codes, pseudorandom functions, pseudorandom-bit generators, and digital signatures. Different uses require different security properties from the hash function, such as preimage resistance, second-preimage resistance, collision resistance, pseudorandomness of output distribution, and so on.

A popular methodology for designing hash functions is to construct a domain extension for an underlying fixed input length component, with the goal of reducing the desired security properties of the larger construction to properties of the component, so that the designers can focus on achieving the necessary properties in the component. For example, the designers of Skein, one of the SHA-3 finalists, give pseudorandom and indifferentiable security proofs for the Skein domain extension when its underlying tweakable block cipher is pseudorandom or ideal, respectively [2]. Another example is that Merkle-Damgård (MD) construction [23, 16] with message length padding (also called Merkle-Damgård Strengthening); this domain extension, used in MD5 and SHA-1, preserves the collision resistance and the preimage resistance of the underlying compression function.

There are several ways to construct pseudorandom functions from MD hash functions, i.e., hash functions based on the MD construction with message length padding : HMAC [10], the Sandwich method [24] and the H^2 construction method [25]. In each case the underlying keyed compression function of the hash function is assumed to be pseudorandom.

The sponge construction [3] is a hash domain extension, designed by Bertoni, Daemen, Peeters, and Assche, that has influenced hash functions such as Keccak [4], PHOTON [18], QUARK [1] and SPONGENT[12]. Recently, the Keccak team provided pseudorandom security proofs of a keyed sponge construction and an authenticated-encryption scheme based on thesSponge construction, in which the key is located in the prefix of the message [5, 6]. It is preferable that the security assumptions be as practical as possible.

In this paper, we propose a new, efficient PRF construction that is based on the sponge construction, called the E-M keyed Sponge construction (EMKSC), and we give a proof of pseudorandomness in the standard model. In particular, we assume that for a given permutation f underlying the sponge construction, the permutation $F_K(\cdot) = f(\cdot \oplus K) \oplus K$ is pseudorandom. The latter permutation is a specialization of the Even-Mansour permutation construction [17], $EM_{K_1,K_2}(M) = f(M \oplus K_1) \oplus K_2$, where $|M| = |K_1| = |K_2| = n$.
. In Section 3 we describe three variants of EMKSC, with different effective key sizes, called EMKSC1, EMKSC2, and EMKSC3. Table 1 shows that, when the underlying hash function is based on the sponge construction, this PRF compares favorably with other constructions that have proofs in the standard model.

Algorithm	# of Hash Calls	# of Underlying function Calls	Final Output Key Masking
HMAC [10]	2	at least ℓ+3	No
Sandwich [24]	1	at least ℓ+2	No
H^2 with one key [25]	2	at least ℓ+2	No
EMKSC1	1	ℓ+1	Yes
EMKSC2	1	ℓ	Yes
EMKSC3	1	ℓ	No

Table 1. Comparison of standard model PRF constructions based on a sponge hash function: ℓ is the number of the underlying function calls in EMKSC2 and EMKSC3.

2 Preliminaries

Let $Func(Dom, Range)$ be the set of all functions from $Dom \to Range$. Let $f : \mathcal{K} \times Dom \to Range$. We say that f is ϵ-prf if for any efficient adversary A, the following holds:

$$Adv_f^{\mathsf{prf}}(A) = |\Pr[K \leftarrow_r \mathcal{K} : A^{f(K,\cdot)} = 1] - \Pr[u \leftarrow_r Func(Dom, Range) : A^{u(\cdot)} = 1]| \leq \epsilon.$$

Let $Perm(Dom, Range)$ be the set of all permutations from $Dom \to Range$, where $|Dom| = |Range|$. We say that f is ϵ-prp if for any efficient adversary A, the following holds:

$$Adv_f^{\mathsf{prp}}(A) = |\Pr[K \leftarrow_r \mathcal{K} : A^{f(K,\cdot)} = 1] - \Pr[u \leftarrow_r Perm(Dom, Range) : A^{u(\cdot)} = 1]| \leq \epsilon.$$

The maximum prf or prp advantages for all the adversaries with at most q-queries are defined as follows:

$$Adv_f^{\mathsf{prf}}(q) := MAX_A Adv_f^{\mathsf{prf}}(A) \text{ and } Adv_f^{\mathsf{prp}}(q) := MAX_A Adv_f^{\mathsf{prp}}(A).$$

The Sponge Construction [3]. The sponge construction, denoted $SPONGE_f$ here, is a domain-codomain extension for a hash function that is based on a permutation or function f, with a fixed input and output length, n. The construction has two other parameters that we omit from our notation: a positive integer r less than n, called the bitrate, and an injective padding function, denoted pad. For any input string M, called the message, the length of $pad(M)$ is a multiple of r, and the last r bits of $pad(M)$ are not all 0. The quantity $n-r$ is called the capacity, dentoed c. The two inputs to the construction are the length of the desired output, denoted ℓ, and the message. It will be convenient to slightly generalize the construction to allow an initial n-bit string, denoted IV, as a third input. The generalized construction, denoted $Sponge_f$, is defined in Fig. 1. $SPONGE_f(M, \ell)$ is defined to be $Sponge_f(0^{r+c}, M, \ell)$, as illustrated in Fig. 2.

The Sponge Construction With Initial Value: $Sponge_f(IV, M, \ell)$
Let $pad(M) = (M_1\|\|....\|\|M_t)$, for some positive t where each $\|M_i\| = r$.
Requirement : $M_t \neq 0^r$ and pad is injective.
100 $s_a = first_r(IV)$ and $s_b = last_c(IV)$
200 for $i = 1$ to t,
201 $(s_a\|\|s_b) = f((s_a \oplus M_i)\|\|s_b)$.
300 for $i = 1$ to $\lceil \frac{\ell}{r} \rceil$,
301 $Z_i = s_a$.
302 $c = f(s_a\|\|s_b)$.
400 return $first_(s_a\|\|s_b)$ bits.

Fig. 1. The Sponge Construction With Initial Value.

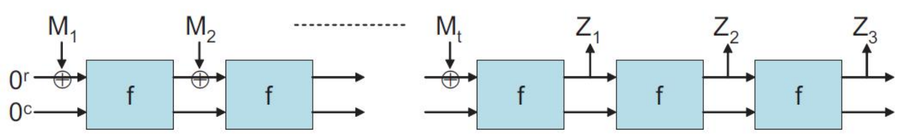

Fig. 2. The Sponge Construction: $SPONGE_f(M, \ell) = first_\ell(Z_1||Z_2||Z_3...)$.

The Keyed Sponge Construction [5]. The keyed sponge construction (called $keySPONGE$) is defined as follows; for a message M, a secret key K of any size, and the desired bit-size ℓ of output, $keySPONGE_f(K, M, \ell) = SPONGE_f(K||M, \ell)$. The keyed sponge construction is proven to be pseudorandom, under the assumption that f is an ideal permutation [5].

The EMKSC. The E-M keyed Sponge construction (called EMKSC) based on the Sponge construction is defined as follows. (See Fig. 3). For a message M, a $(r+c)-bit$ secret key $K_1||K_2$, and the desired bit-size ℓ of output, $EMKSC_f(K_1||K_2, M, \ell) = Sponge_f(K_1||K_2, M, \ell) \oplus first_\ell(K_1||K_1||...)$.

Indifferentiability [22]. Maurer *et al.* [22] defined the indifferentiable security of a target system TS, when the adversary can have access to a tuple of additional oracles $AO = (AO_1, ..., AO_i)$. If there exists a tuple of efficient simulators $S = (S_1, ..., S_i)$ such that ϵ is negligible for any adversary D, we say that TS is (ϵ, AO, S)-indifferentiable from the compared oracle CO if

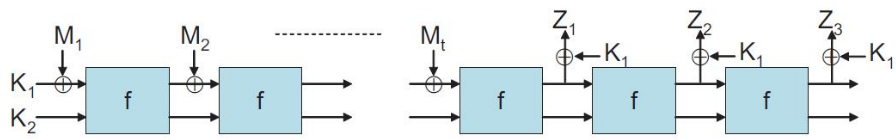

Fig. 3. $EMKSC_f(K_1||K_2, M, \ell) = Sponge_f(K_1||K_2, pad(M), \ell) \oplus first_\ell(K_1||....||K_1)$.

$$Adv^{\mathsf{indiff}}_{TS,AO,S}(D) = |\Pr[D^{TS,AO} = 1] - \Pr[D^{CO,S} = 1] \le \epsilon,$$

where TS and AO may have access to each other by the definitions of TS and AO, and S may have access to CO by the definition of a protocol based on CO.

3 Security of EMKSC

Depending on how the key is generated, we consider the following three variants of the EMKSC in the standard model:

1. EMKSC1: K_1 is a r-bit random string and $K_2 = 0^c$. This case is applied to Keccak without any modification if K_1 is considered as the first block of message of Keccak and K_1 is again xored to the output of Keccak as long as its size of hash output is less than or equal to r.
2. EMKSC2: K_1 and K_2 are r-bit and c-bit random strings, respectively.
3. EMKSC3: $K_1 = 0^r$ and K_2 is a c-bit random string. Note that there is no key masking of the final output because $K_1 = 0^c$).

3.1 Pseudorandomness of the EMKSC

EMKSC in Fig. 3 can be described in a different way as shown in Fig. 4. So, we will give a PRF security proof of the EMKSC, using the alternative description in Fig. 4, when F_K is pseudorandom.

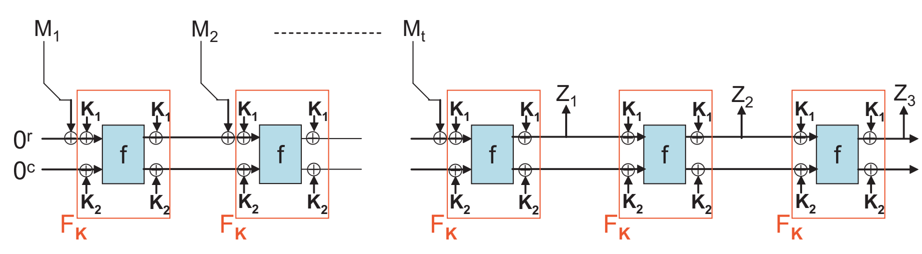

Fig. 4. Alternative Description of the EMKSC, where $K = K_1||K_2$.

Theorem 1. *For any key K, let $F_K(\cdot) = f(\cdot \oplus K) \oplus K$ be a permutation from $(c+r)$-bit strings to $(c+r)$-bit strings. Let pad be a padding function such that the function is injective and the final r-bit block of its output is not 0^r. Let $EMKSC_f$ be the EMKSC based on f, where the output size is fixed as ℓ. Let $M^1, M^2, \cdots, M^q$ be q distinct inputs of the sponge construction. Let $\sigma_1 = \sum_{j=1}^{q} t_j$ and $\sigma_2 = q \cdot \lceil \frac{\ell}{r} \rceil$, where for each i $pad(M^i) = (M_1^i, M_2^i, ..., M_{t_i}^i)$. Then, for any prf attacker A with at most q-queries, $Adv^{\mathsf{prf}}_{EMKSC_f}(A) \leq \frac{(\sigma_1+\sigma_2+1)^2}{2^{c+1}} + \frac{(\sigma_1+\sigma_2+1)^2}{2^{r+c+1}} + Adv^{\mathsf{prp}}_{F}(\sigma_1+\sigma_2)$.*

Proof. For $K = K_1||K_2$, we consider three cases; 1) $K_1 \xleftarrow{\$} \{0,1\}^r$ and $K_2 = 0^c$, 2) $K_1||K_2 \xleftarrow{\$} \{0,1\}^{r+c}$, and 3) $K_1 = 0^r$ and $K_2 \xleftarrow{\$} \{0,1\}^c$. It is clear that the game $G0$ exactly simulates $EMKSC_f(K, \cdot, \ell)(= Sponge_{F_K}(0^{r+c}, pad(\cdot), \ell))$ for the three cases. Also, the game $G1$ exactly simulates $Sponge_u(0^{r+c}, pad(\cdot), \ell)$, where u is a random permutation from $\{0,1\}^{r+c} \to \{0,1\}^{r+c}$. Since other functionalities are same except u and F_K, it is clear that $|\Pr[A^{G0} = 1] - \Pr[A^{G1} = 1]| \leq Adv^{\mathsf{prp}}_{F}(\sigma_1 + \sigma_2)$, where $\sigma_1 + \sigma_2$ is the maximum number of calls of u or F_K.

Claim 1. $|\Pr[A^{G1} = 1] - \Pr[A^{G2} = 1]| \leq \Pr[A^{G_1}\ sets\ BAD] \leq \frac{(\sigma_1+\sigma_2+1)^2}{2^{c+1}} + \frac{(\sigma_1+\sigma_2+1)^2}{2^{r+c+1}}$.
Proof of Claim 1. There are three types of bad events, BAD_1, BAD_2 and BAD_3. The maximum number of the subroutine g calls in Fig. 6 is $\sigma_1 + \sigma_2$. So, it is clear that $\Pr[A^{G_1}\ sets\ BAD_1] \leq \frac{(\sigma_1+\sigma_2+1)^2}{2^{r+c+1}}$ and $\Pr[A^{G_1}\ sets\ BAD_2] \leq \frac{(\sigma_1+\sigma_2+1)^2}{2^{c+1}}$. In case of the event BAD_3, as long as BAD_1 doesn't occur, there is no way that BAD_2 occurs, because "$g(x)$ is already defined and $\varepsilon \notin Pre$" means that there should be internal collision on the last c-bit, which is not allowed in line 303. Therefore, the Claim 1 holds. □

Claim 2. The game $G2$ exactly simulates a random function from $\{0,1\}^* \to \{0,1\}^\ell$.
Proof of Claim 2. We have to prove that for each query M^i its output distribution should be random. In Fig. 6, the output value z in line 215 is defined from the first r-bit output values of the function g. So, it is sufficient to show that for all different queries all the first r-bit output values of the function g, which determine the final output value z in line 215, are independently random. In the game $G2$, for a new query x of the function g, in line 301, the first r-bit output y_1 is randomly chosen, and in line 303, the last c-bit output y_2 is chosen differently from all the last c-bits of previous outputs. Moreover, since the last c-bit of the output of the function g becomes again the last c-bit of the input of the function g, for any $M^i \neq M^j$, $Sponge_g(0^{r+c}, pad(M^i), \ell)$ is independent from $Sponge_g(0^{r+c}, pad(M^j), \ell)$. Therefore, the Claim 2 holds. □

Therefore, $Adv^{\mathsf{prf}}_{PRF_f}(A) \leq |\Pr[A^{G0} = 1] - \Pr[A^{G2} = 1]| \leq \frac{(\sigma_1+\sigma_2+1)^2}{2^{c+1}} + \frac{(\sigma_1+\sigma_2+1)^2}{2^{r+c+1}} + Adv^{\mathsf{prp}}_{F}(\sigma_1 + \sigma_2)$.

■

3.2 Security Analysis of F_K

Here, we show that it can be "*reasonably assumed*" that $F_K(\cdot) = f(\cdot \oplus K) \oplus K$ is pseudorandom, where $K = K_1||K_2$, and f is the underlying permutation of the sponge construction. We provide the two security analyses of F_K: one suggests that there is no structural weakness F_K, using the notion of indifferentiability, and the other gives a security bound of F_K against known attack techniques. The results of this subsection suggest that the design of F_K is sound to use in practice.

Game $G0$
Initialize : f is a fixed permutation, and $K = K_1\|\|K_2$. For the first case, $K_1 \xleftarrow{\$} \{0,1\}^r$ and $K_2 = 0^c$. For the second case, $K_1\|\|K_2 \xleftarrow{\$} \{0,1\}^{r+c}$. For the third case, $K_1 = 0^r$ and $K_2 \xleftarrow{\$} \{0,1\}^c$.
100 On i-th query M^i, 101 $\quad z^i = Sponge_{F_K}(0^{r+c}, pad(M^i), \ell)$, 102 $\quad$ return z^i. 200 ROUTINE $Sponge_{F_K}$-query $(0^{r+c}, x, \ell)$, 201 $\quad$ Let $x = (m_1\|\|....\|\|m_t)$, where each $\|m_i\| = r$. 202 $\quad (s_a, s_b) = (0^r, 0^c)$. 203 $\quad \varepsilon$ is initialized as the empty string. 204 $\quad$ for $i = 1$ to t, 205 $\qquad \varepsilon = \varepsilon\|\|m_i$. 206 $\qquad (s_a\|\|s_b) = F_K(s_a \oplus m_i\|\|s_b)$. 207 $\quad z$ is initialized as the empty string. 208 $\quad$ for $i = 1$ to $\lceil \frac{\ell}{r} \rceil$, 209 $\qquad Z_i = s_a$ and $z = (z\|\|Z_i)$. 210 $\qquad \varepsilon = \varepsilon\|\|0^r$. 211 $\qquad (s_a\|\|s_b) = F_K(s_a\|\|s_b)$. 212 $\quad z$:= the first ℓ-bit of z. 213 $\quad$ return z. 300 SUBROUTINE F_K-query x where $\|x\| = r + c$, 301 $\quad$ define $y := f(x \oplus K) \oplus K$. 302 $\quad$ return y.

Fig. 5. $G0$ perfectly simulates $EMKSC_f(K, \cdot, \ell)(= Sponge_{F_K}(0^{r+c}, pad(\cdot), \ell))$.

Structural Soundness of F_K. In Fig. 2, Fig. 3, and Fig. 4, we want to show the security bound of F_K for general distinguishing attacks which do not use the internal structure of f. For this, we assume that f is an easy-to-invert ideal permutation, where any attacker can access f. Then, we show that F_K is indifferentiable from the invertible random permutation oracle $\mathcal{F}$ on $\{0,1\}^{r+c}$.

Theorem 2 **(K_1 is random and $K_2 = 0$).** $F_K(x_1\|\|x_2) := y_1\|\|y_2 = f((x_1 \oplus K_1)\|\|x_2) \oplus (K_1\|\|0^c)$, *where* $|x_1| = |y_1| = |K_1| = r$, $|x_2| = |y_2| = c$, *and* f *is an easy-to-invert permutation on* $\{0,1\}^{r+c}$. *Then, we can construct a simulator* $S = (S_f, S_{f^{-1}})$ *such that for any indifferentiability adversary* A *making at most* (q_1, q_2, q_3) *queries to its three oracles the following holds;*

$$Adv^{\mathsf{indiff}}_{F_K,(f,f^{-1}),S}(A) \leq \frac{3 \cdot q^2}{2^r - q}, \text{ where } q = q_1 + q_2 + q_3.$$

Proof. Fig. 7 shows how $F_K, (f, f^{-1})$ and the simulator S work. It is easy to check that Games $G2$ and $G3$ in Fig. 8 perfectly simulate (F_K, f, f^{-1}) and $(\mathcal{F}, S_f, S_{f^{-1}})$ respectively. So, we have only to compute the bound of probability that bad events, BAD_1, BAD_2, and BAD_3 occur. Since K is randomly chosen from $\{0,1\}^r$ and the remaining c bits of input can be controlled by the attacker A, it is clear that for each i, $\Pr[A^{G_2} \text{ sets } BAD_i] \leq \frac{q^2}{2^r - q}$. Therefore, the above theorem holds. ■

Theorem 3 **(K_1 and K_2 are both random).** *Let $F_{K_1,K_2}(x_1||x_2) := y_1||y_2 = f((x_1 \oplus K_1)||(x_2 \oplus K_2)) \oplus (K_1||K_2)$, where $|x_1| = |y_1| = |K_1| = r$, $|x_2| = |y_2| = |K_2| = c$, and f is an easy-to-invert permutation on $\{0,1\}^{r+c}$. Then, we can construct a simulator $S = (S_f, S_{f^{-1}})$ such that for any indifferentiability adversary A making at most (q_1, q_2, q_3) queries to its three oracles the following holds;*

$$Adv^{\mathsf{indiff}}_{F_{K_1,K_2},(f,f^{-1}),S}(A) \leq \frac{3 \cdot q^2}{2^{r+c}-q}, \text{ where } q = q_1 + q_2 + q_3.$$

Proof. This can be proven in the same way as Theorem 2. ∎

Theorem 4 **($K_1 = 0$ and K_2 is random).** *$F_K(x_1||x_2) := y_1||y_2 = f(x_1||(x_2 \oplus K_2)) \oplus (0^r||K_2)$, where $|x_1| = |y_1| = r$, $|x_2| = |y_2| = |K_2| = c$, and f is an easy-to-invert permutation on $\{0,1\}^{r+c}$. Then, we can construct a simulator $S = (S_f, S_{f^{-1}})$ such that for any indifferentiability adversary A making at most (q_1, q_2, q_3) queries to its three oracles the following holds;*

$$Adv^{\mathsf{indiff}}_{F_K,(f,f^{-1}),S}(A) \leq \frac{3 \cdot q^2}{2^{c}-q}, \text{ where } q = q_1 + q_2 + q_3.$$

Proof. This can be proven in the same way as Theorem 2. ∎

Security of F_K against Key-recovery Attack Even and Mansour first proposed a block-cipher construction from a publicly known permutation f, called the Even-Mansour construction [17], which is $EM_{K_1,K_2}(M) = f(M \oplus K_1) \oplus K_2$, where $|M| = |K_1| = |K_2| = n$. Informally, they proved that the Even-Mansour construction is secure when f is an ideal permutation, where an attacker can freely access f, and the number of queries to f and F_K is bounded by $O(2^{n/2})$ [17]. There are several known key-recovery attacks on the Even-Mansour construction where the input and output masks are chosen independently at random. Daemen [15] described known and chosen plaintext-based key recovery attacks on the Even-Mansour construction with complexity abot 2^{n-1} and $2^{n/2}$, respectively. But our specialization of the construction differs in two ways: 1) the input and output masking keys are the same and 2) the masking key for EMKSC1 and EMKSC3 affects only part of the input and output of the underlying public permutation f. Here, we modify Daemen's attack a little bit to give the security of F_K against the chosen plaintext attack.

[K_1 is random and $K_2 = 0$] $F_K(x_1||x_2) := y_1||y_2 = f((x_1 \oplus K_1)||x_2) \oplus (K_1||0^c)$, where $|x_1| = |y_1| = |K_1| = r$, $|x_2| = |y_2| = c$, and f is an easy-to-invertible permutation on $\{0,1\}^{r+c}$. The complexity of our key recovery attack is $2^{r/2}$ queries and $2^{r/2}$ memory as follows.

The Attacker A works as follows:

1. An attacker A makes $2^{r/2}$ queries (M_a^i, M_b^i) (where $1 \leq i \leq 2^{r/2}$) to F_K and obtains $2^{r/2}$ (C_a^i, C_b^i), where for all i, $|M_a^i| = |M_b^i| = r + c$, $M_a^i \oplus M_b^i = v||0^c$ for some r-bit constant v, and the substring of the last c bits of both M_a^i and M_b^i is a constant, w.
2. A repeats the following step at most $2^{r/2}$ times: For an r-bit random value X, A computes $\Delta W = f(X||w) \oplus f((X \oplus v)||w)$ and checks if there exists an i such that $\Delta C^i = \Delta W$, where $\Delta C^i = C_a^i \oplus C_b^i$. If X is the same as the first r-bit $M_a^i \oplus (K||0^c)$ or $M_b^i \oplus (K||0^c)$ for some i, then ΔW should be the same as ΔC^i. But, for each trial for X, the probability that $\Delta W = \Delta C^i$ for some i is about $2^{r/2}$. So, we can expect at least one X among $2^{r/2}$ trials which satisfies the condition, that is, we can find the key K with $2^{r/2}$ queries and $2^{r/2}$ memory.

Remark. The above attack shows that the indifferentiable security bound of F_K shown in Theorem 2 is tight as $O(2^{r/2})$. The result of this section can be applied to Keccak [4], because we don't need to change any initial value and internal structure except for the input and output values of Keccak.

[K_1 **and** K_2 **are both random**] $F_{K_1,K_2}(x_1||x_2) := y_1||y_2 = f((x_1 \oplus K_1)||(x_2 \oplus K_2)) \oplus (K_1||K_2)$, where $|x_1| = |y_1| = |K_1| = r$, $|x_2| = |y_2| = |K_2| = c$, and f is an easy-to-invertible permutation on $\{0,1\}^{r+c}$. The complexity of our key recovery attack is $2^{(r+c)/2}$ queries and $2^{(r+c)/2}$ memory, which can be shown in a similar way.

Remark. The above attack shows that the indifferentiable security bound of F_{K_1,K_2} shown in Theorem 3 is $O(2^{(r+c)/2})$, which is much bigger than the security bound $O(2^{r/2})$ of our first construction.

[$K_1 = 0$ **and** K_2 **is random**] In the same way of above attacks, we can define an key recovery attacker A on F_K of the third construction with $2^{c/2}$ queries and $2^{c/2}$ memory.

4 Conclusion

In this paper, we have shown how to prove the pseudorandomness of a new keyed sponge construction in the standard model. Our proof technique may be applied to prove the pseudorandom security of keyed constructions based on domain extensions of the hash functions: Luffa [13], CubeHash [8], Fugue [19], and an authenticated-encryption scheme based on the sponge construction [6].

References

1. J. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia, *Quark: A Lightweight Hash*, CHES'10, LNCS 6225, Springer-Verlag, pp. 1-15, 2010.
2. M. Bellare, T. Kohno, S. Lucks, N. Ferguson, B. Schneier, D. Whiting, J. Callas, J. Walker, *Provable Security Support for the Skein Hash Family*, http://www.skein-hash.info/sites/default/files/skein-proofs.pdf.
3. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, *On the Indifferentiability of the Sponge Construction*, Advances in Cryptology – EUROCRYPT'08, LNCS 4965, Springer-Verlag, pp. 181-197, 2008.
4. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, *The Keccak sponge function family*, Submission to NIST, 2008. (http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/submissions_rnd2.html)
5. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, *On the security of the keyed sponge construction*, Submission to the NIST second SHA-3 workshop, 2010. (http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/Aug2010/documents/papers/VANASSCHE_SpongeKeyed.pdf)
6. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, *Duplexing the sponge: single-pass authenticated encryption and other applications*, Submission to the NIST second SHA-3 workshop, 2010. (http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/Aug2010/documents/papers/DAEMEN_DuplexSponge.pdf)
7. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, *Duplexing the sponge: single-pass authenticated encryption and other applications*, SAC'11, to apear, 2011. (http://sac2011.ryerson.ca/SAC2011/BDPVA.pdf)

8. D. Bernstein, *CubeHash: a simple hash function*, Submission to NIST, 2008. (http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/submissions_rnd2.html)
9. M. Bellare and P. Rogaway, *The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs*, Advances in Cryptology – EUROCRYPT'06, LNCS 4004, Springer-Verlag, pp. 409-426, 2006.
10. M. Bellare, R. Canetti, and H. Krawczyk, *Keying hash functions for message authentication*, Advances in Cryptology – EUROCRYPT'96, LNCS 1109, Springer-Verlag, pp. 1-15, 1996.
11. M. Bellare and T. Ristenpart, *Multi-Property-Preserving Hash Domain Extension and the EMD Transform*, Advances in Cryptology – ASIACRYPT'06, LNCS 4284, pp. 299–314, 2006.
12. A. Bogdanov, M. Knežević, G. Leander, D. Toz, K. Varici, and I. Verbauwhede, *spongent: A Lightweight Hash Function*, CHES'2011, LNCS 6917, pp. 312–325, 2011.
13. C. D. Cannière, H. Sato, and D. Watanabe, *The Hash Function Family Luffa*, Submission to NIST, 2008. (http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/submissions_rnd2.html)
14. J. S. Coron, Y. Dodis, C. Malinaud and P. Puniya, *Merkle-Damgard Revisited: How to Construct a Hash Function*, Advances in Cryptology-CRYPTO'05, LNCS 3621, Springer-Verlag, pp. 430-448, 2005.
15. J. Daemen, *Limitations of the Even-Mansour Construction*, Advances in Cryptology – ASIACRYPT'91, LNCS 739, Springer-Verlag, pp. 495-498, 1991.
16. I. B. Damgard, *A design principle for hash functions*, Advances in Cryptology-CRYPTO'89, LNCS 435, Springer-Verlag, pp. 416-427, 1990.
17. S. Even and Y. Mansour, *A Construction of a Cipher From a Single Pseudorandom Permutation*, Advances in Cryptology-ASIACRYPT'91, LNCS 739, Springer-Verlag, pp. 210-224, 1992.
18. J. Guo, T. Peyrin, and A. Poschmann, *The PHOTON Family of Lightweight Hash Functions*, Advances in Cryptology-CRYPTO'11, LNCS 6841, Springer-Verlag, pp. 222-239, 2011.
19. S. Halevi, W. E. Hall, and C. S. Jutla, *The Hash Function Fugue*, Submission to NIST, 2008. (http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/submissions_rnd2.html)
20. J. Kelsey and B. Schneier, *Second preimages on n-bit hash functions for much less than 2^n work*, Advances in Cryptology – EUROCRYPT'05, LNCS 3494, Springer-Verlag, pp. 474–490, 2005.
21. M. Liskov, R. L. Rivest, and D. Wagner, *Tweakable Block Ciphers*, Advances in Cryptology – CRYPTO'02, LNCS 2442, Springer-Verlag, pp. 31-46, 2002.
22. U. Maurer, R. Renner and C. Holenstein, *Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology*, TCC'04, LNCS 2951, Springer-Verlag, pp. 21-39, 2004.
23. R. C. Merkle, *One way hash functions and DES*, Advances in Cryptology-CRYPTO'89, LNCS 435, Springer-Verlag, pp. 428-446, 1990.
24. K. Yasuda, *"Sandwich" Is Indeed Secure: How to Authenticate a Message with Just One Hashing*, ACISP'07, LNCS 4586, Springer-Verlag, pp. 355-369, 2007.
25. K. Yasuda, *HMAC without the "Second" Key*, ISC'09, LNCS 5735, Springer-Verlag, pp. 443-458, 2009.

Game $\boxed{G1}$ and $\boxed{G2}$

Initialize : $g : \{0,1\}^{r+c} \rightarrow \{0,1\}^{r+c}$ is everywhere-undefined, $V = \{0^c\}$, and $W = Pre = \emptyset$.

On i-th query M^i,
    $z^i = Sponge_g(0^{r+c}, pad(M^i), \ell)$,
    return z^i.

ROUTINE $Sponge_g$-query $(0^{r+c}, x, \ell)$,
    Let $x = (m_1||....||m_t)$, where each $|m_i| = r$.
    $(s_a, s_b) = (0^r, 0^c)$, where 0^j is the j-bit zeros.
    ε is initialized as the empty string.
    for $i = 1$ to t,
        $\varepsilon = \varepsilon||m_i$.
        $(s_a||s_b) = g(s_a \oplus m_i||s_b)$.
        $Pre = Pre \cup \{\varepsilon\}$.
    z is initialized as the empty string.
    for $i = 1$ to $\lceil \frac{\ell}{r} \rceil$,
        $Z_i = s_a$ and $z = (z||Z_i)$.
        $\varepsilon = \varepsilon||0^r$.
        $(s_a||s_b) = g(s_a||s_b)$.
        $Pre = Pre \cup \{\varepsilon\}$.
    z:= the first ℓ-bit of z.
    return z.

SUBROUTINE g-query x where $|x| = r + c$,
    $y := y_1||y_2 \xleftarrow{\$} \{0,1\}^{r+c}$.
    if $y \in W$, then $\mathsf{BAD}_1 \leftarrow \mathsf{true}$ $\boxed{\text{and } y := y_1||y_2 \xleftarrow{\$} \{0,1\}^{r+c} \setminus W}$.
    if $y_2 \in V$ then $\mathsf{BAD}_2 \leftarrow \mathsf{true}$ $\boxed{\text{and } y_2 \xleftarrow{\$} \{0,1\}^c \setminus V}$.
    if $g(x)$ is already defined and $\varepsilon \in Pre$, then $y = g(x)$.
    $V = V \cup \{y_2\}$.
    if $g(x)$ is already defined and $\varepsilon \notin Pre$, then $\mathsf{BAD}_3 \leftarrow \mathsf{true}$ $\boxed{\text{and } y = g(x)}$.
    define $g(x) := y$ and $W = W \cup \{y\}$.
    return y.

Fig. 6. $G1$ executes with the non-dotted boxed statement and without the dotted boxed statement, whereas $G2$ executes with the dotted boxed statement and without the non-dotted boxed statement. Clearly $G1$ and $G2$ are identical-until-BAD. $G1$ perfectly simulates $Sponge_u(0^{r+c}, pad(\cdot), \ell)$, where u is a random permutation from $\{0,1\}^{r+c} \rightarrow \{0,1\}^{r+c}$. $G2$ perfectly simulates a random function from $\{0,1\}^* \rightarrow \{0,1\}^\ell$.

(F_K, f, f^{-1})	$(\mathcal{F}, S_f, S_{f^{-1}})$
Initialize : $K_1 \xleftarrow{\$} \{0,1\}^r$.	Initialize : $W = X = Z = \emptyset$.
100 On i-th F_K query $x^i = x_1^i \| x_2^i$,	100 On i-th $\mathcal{F}$ query $x^i = x_1^i \| x_2^i$,
101 $\quad y^i := y_1^i \| y_2^i = f((x_1^i \oplus K_1) \| x_2^i) \oplus (K_1 \| 0^c)$.	101 $\quad y^i := y_1^i \| y_2^i \xleftarrow{\$} \{0,1\}^{r+c} \setminus W$.
102 $\quad$ return y^i.	102 $\quad W = W \cup \{y^i\}$ and return y^i.
200 On i-th f query $a^i = a_1^i \| a_2^i$,	200 On i-th S_f query $a^i = a_1^i \| a_2^i$,
201 $\quad b^i := b_1^i \| b_2^i = f(a^i)$.	201 $\quad b^i := b_1^i \| b_2^i \xleftarrow{\$} \{0,1\}^{r+c} \setminus X$.
202 $\quad$ return b^i.	202 $\quad Z = Z \cup \{a^i\}$, $X = X \cup \{b^i\}$, and return b^i.
300 On i-th f^{-1} query $b^i = b_1^i \| b_2^i$,	300 On i-th $S_{f^{-1}}$ query $b^i = b_1^i \| b_2^i$,
301 $\quad a^i := a_1^i \| a_2^i = f^{-1}(b^i)$.	301 $\quad a^i := a_1^i \| a_2^i \xleftarrow{\$} \{0,1\}^{r+c} \setminus Z$.
302 $\quad$ return a^i.	302 $\quad Z = Z \cup \{a^i\}$, $X = X \cup \{b^i\}$, and return a^i.

Fig. 7. (F_K, f, f^{-1}) and $(\mathcal{F}, S_f, S_{f^{-1}})$. $F_K(x_1||x_2) := y_1||y_2 = f((x_1 \oplus K_1)||x_2) \oplus (K_1||0^c)$, where $K = K_1||K_2$, $K_2 = 0$, $|x_1| = |y_1| = |K_1| = r$, $|x_2| = |y_2| = c$, and f is an easy-to-invertible permutation on $\{0,1\}^{r+c}$.

Game $\boxed{G2}$ and $G3$
Initialize : $K_1 \xleftarrow{\$} \{0,1\}^r$ and $U = V = W = X = Z = \emptyset$.
100 On i-th O_1 query $x^i = x_1^i \| x_2^i$ where $\|x_1^i\| = r$ and $\|x_2^i\| = c$,
101 $\quad y^i := y_1^i \| y_2^i \xleftarrow{\$} \{0,1\}^{r+c} \setminus W$.
102 $\quad$ if $(x_1^i \oplus K_1) \| x_2^i \in \{x \mid (x,*) \in U\}$, then $\mathsf{BAD}_1 \leftarrow \mathsf{true}$ $\boxed{\text{and } y^i = (K \| 0^c) \oplus f((x_1^i \oplus K_1) \| x_2^i)}$.
103 $\quad$ define $f'((x_1^i \oplus K_1) \| x_2^i) := y^i$, $W = W \cup \{y^i\}$ and $V = V \cup \{((x_1^i \oplus K_1) \| x_2^i, y^i \oplus (K_1 \| 0^c))\}$.
104 $\quad$ return y^i.
200 On i-th O_2 query $a^i = a_1^i \| a_2^i$ where $\|a_1^i\| = r$ and $\|a_2^i\| = c$,
201 $\quad b^i := b_1^i \| b_2^i \xleftarrow{\$} \{0,1\}^{r+c} \setminus X$.
202 $\quad$ if $a^i \in \{a \mid (a,*) \in V\}$, then $\mathsf{BAD}_2 \leftarrow \mathsf{true}$ $\boxed{\text{and } b^i = f'(a^i)}$.
203 $\quad$ define $f(a^i) := b^i$, $Z = Z \cup \{a^i\}$, $X = X \cup \{b^i\}$ and $U = U \cup \{(a^i, b^i)\}$.
204 $\quad$ return b^i.
300 On i-th O_3 query $b^i = b_1^i \| b_2^i$ where $\|b_1^i\| = r$ and $\|b_2^i\| = c$,
301 $\quad a^i := a_1^i \| a_2^i \xleftarrow{\$} \{0,1\}^{r+c} \setminus Z$.
302 $\quad$ if $b^i \in \{b \mid (*,b) \in V\}$, then $\mathsf{BAD}_3 \leftarrow \mathsf{true}$ $\boxed{\text{and } a^i = f'^{-1}(b^i)}$.
303 $\quad$ define $f(a^i) := b^i$, $Z = Z \cup \{a^i\}$, $X = X \cup \{b^i\}$ and $U = U \cup \{(a^i, b^i)\}$.
304 $\quad$ return a^i.

Fig. 8. $G2$ executes with boxed statements whereas $G3$ executes without these. Clearly $G2$ and $G3$ are identical-until-BAD. $G2$ and $G3$ perfectly simulate (F_K, f, f^{-1}) and $(\mathcal{F}, S_f, S_{f^{-1}})$ respectively. In this games, we assume that there is no repetition query. Note that the simulator $S = (S_f, S_{f^{-1}})$ works without access to $\mathcal{F}$.