On the security of the keyed sponge construction

Guido Bertoni[1], Joan Daemen[1], Michaël Peeters[2], and Gilles Van Assche[1]

[1] STMicroelectronics
[2] NXP Semiconductors

Abstract. The advantage in differentiating the sponge construction from a random oracle is upper bounded by $N^2 2^{-(c+1)}$, with N the number of calls to the underlying transformation or permutation and c the capacity, resulting in an expected time complexity of $N \sim 2^{c/2}$. In this paper we prove that the advantage in distinguishing a keyed sponge from a random oracle is much smaller in typical use cases. In particular, when the data complexity is limited to $M \ll 2^{c/2}$, the expected time complexity is about $N \sim \min(2^c/M, 2^{|K|})$, with $|K|$ the length of the key. This improvement over the indifferentiability bound allows decreasing the capacity (and thus the permutation width) for a given required security level or achieving a higher security level for a given capacity. This new bound has positive implications for all applications in which a sponge function is used for encryption and/or authentication, or generally in conjunction with a key, including on platforms with limited resources.

Keywords: sponge construction, generic attacks, keyed sponge, key recovery, stream cipher, encryption, authentication

1 Introduction

Sponge functions are versatile cryptographic primitives that can be used for hashing, message authentication code (MAC) computation, stream encryption, authenticated encryption, pseudo-random sequence generation and other applications. We refer to [6,5,7] for description of such modes of use.

A sponge function consists of the application of the sponge construction to a fixed-length permutation or transformation f. The sponge construction is sound in the sense that attacks that do not exploit specific properties of f, i.e., generic attacks, are very unlikely to have success. More exactly, the advantage in differentiating the sponge construction calling a random permutation or transformation f from a random oracle is upper bounded by $N^2 2^{-(c+1)}$ with N the number of calls to the underlying transformation or permutation and c the capacity [4]. This implies that with respect to generic attacks of complexity N, the sponge construction offers the same level of security as a random oracle, as long as $N^2 2^{-(c+1)}$ is negligible. Once N approaches $2^{c/2}$ this is no longer the case and the indifferentiability bound does not give any guarantees. As pointed out in [3,4], this is due to the existence of *inner collisions* in a sponge function.

It was noted in [3] that the expected workload of determining the inner state of a sponge function from its output only is probably much higher than $2^{c/2}$. However, no lower bound for this workload had yet been proven. In modes of use where a sponge function is used in conjunction with a key, the difficulty of state recovery is crucial and the amount of sponge function processing with a given key is usually too small for inner collisions to actually occur. Hence, while the indifferentiability bound suggests that the capacity must be twice the size of the key, the attacks in [3] suggest that a capacity slightly larger than the key may suffice. In other words, the security level of a sponge function in a keyed mode is probably much higher than $2^{c/2}$.

The success probability of state recovery was studied in [6], for which an upper bound was proven. In this paper we generalize this to an upper bound for the advantage of distinguishing a keyed sponge from a random oracle. Indistinguishability is a very strong notion of security as it implies the inability of generating collisions, retrieving the state or guessing the key. Hence, this upper bound allows keyed sponges to claim a higher security level for a given capacity or to take a smaller capacity for a required security level in any application. Note in a sponge function the sum of bitrate and capacity is equal to the width of the underlying permutation (or transformation) and hence decreasing capacity results either in an immediate increase of bitrate and hence efficiency or in a smaller permutation width and hence a smaller implementation.

In this paper we concentrate on the case that f is a permutation because of its practical relevance. A similar bound can be proven for the case that f is a transformation and we leave it as an exercise for the reader.

The remainder of this paper is organized as follows. First we provide the reader with the definition of the sponge construction and of the keyed sponge in Section 2. This is followed by a quick introduction of indistinguishability in Section 3. In Section 4 we prove the new bound in a number of steps. Finally, in Section 5 we discuss the implications of this bound.

2 The sponge construction

The sponge construction [3] builds a function $\textsc{sponge}[f, \text{pad}, r]$ with variable-length input and arbitrary output length using a fixed-length permutation (or transformation) f, a padding rule "pad" and a parameter *bitrate* r. A sponge function, that is, a function implementing the sponge construction, provides a particular way to generalize hash functions and has the same interface as a random oracle.

2.1 Definition

The permutation f operates on a fixed number of bits, the *width* b. The sponge construction has a state of b bits. First, all the bits of the state are initialized to zero. The input message is padded and cut into blocks of r bits. The padding rule shall be invertible and the last r-bit block of the padded string shall be different from zero. We use the following notation: the padding of a message M to a sequence of x-bit blocks is denoted by $M||\text{pad}[x](|M|)$.

After padding it proceeds in two phases: the *absorbing phase* followed by the *squeezing phase*:

Absorbing phase The r-bit input message blocks are XORed into the first r bits of the state, interleaved with applications of the function f. When all message blocks are processed, the sponge construction switches to the squeezing phase.

Squeezing phase The first r bits of the state are returned as output blocks, interleaved with applications of the function f. The number of iterations is determined by the requested number of bits.

Finally the output is truncated to the requested length ℓ. The sponge construction is illustrated in Figure 1 and Algorithm 1 provides a formal definition. The notation $\lfloor x \rfloor_n$ means that the string x is truncated after its first n bits.

The value $c = b - r$ is called the *capacity*. The first r bits of a state s are called its *outer part* $\overline{s}$ and the last c bits its *inner part* $\widehat{s}$. The inner part of the state is never directly affected by the input blocks and never output during the squeezing phase. As said, the capacity c actually determines the attainable security level of the construction [4].

2.2 Graph representation of sponge operations

We adopt the graph representation used in [3] for describing the knowledge of the adversary. We consider the transformation f as a directed graph whose vertex set (called *nodes*) is $\mathbb{Z}_2^b = \mathbb{Z}_2^r \times \mathbb{Z}_2^c$ and whose edges are $(s, f(s))$. It has both 2^b nodes and edges. From the node graph, we derive

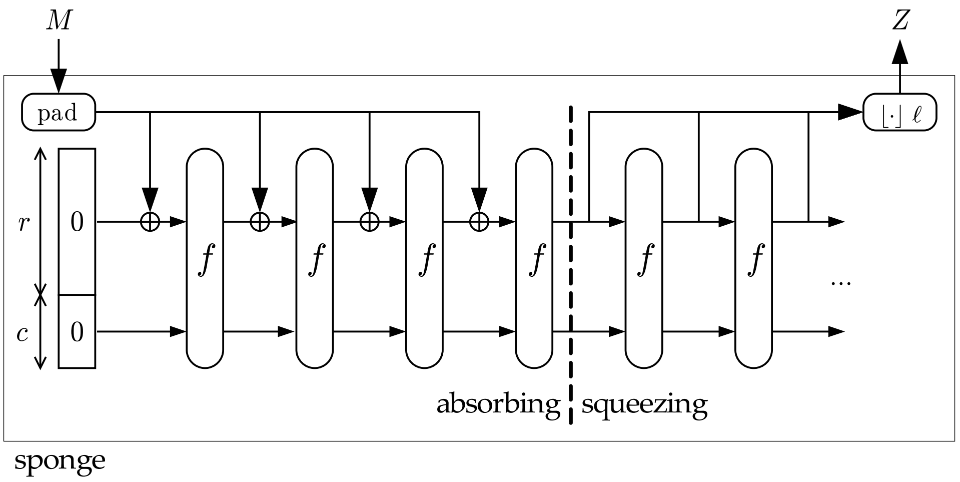

Fig. 1. The sponge construction

Algorithm 1 The sponge construction SPONGE$[f, \text{pad}, r]$

```
**Require:** $r < b$

  **Interface:** $Z = \text{sponge}(M, \ell)$ with $M \in \mathbb{Z}_2^*$, integer $\ell > 0$ and $Z \in \mathbb{Z}_2^\ell$
  $P = M||\text{pad}[r](|M|)$
  Let $P = P_0||P_1||\ldots||P_w$ with $|P_i| = r$
  $s = 0^b$
  **for** $i = 0$ to $w$ **do**
    $s = s \oplus (P_i||0^{b-r})$
    $s = f(s)$
  **end for**
  $Z = \lfloor s \rfloor_r$
  **while** $|Z| < \ell$ **do**
    $s = f(s)$
    $Z = Z||\lfloor s \rfloor_r$
  **end while**
  **return** $\lfloor Z \rfloor_\ell$
```

the (directed) supernode graph, with vertex set (called *supernodes*) equal to $\mathbb{Z}_2^c$. In this graph, an edge $(\widehat{s},\widehat{t})$ is in the edge set if and only if $\exists \overline{s},\overline{t}$ such that (s,t) with $s=(\overline{s},\widehat{s})$ and $t=(\overline{t},\widehat{t})$ is an edge in the node graph. The set of supernodes is a partition of the nodes where a supernode contains the 2^r nodes with the same inner part.

The sponge construction operates on a chaining variable s and its operation can be seen as a walk through the node graph of the chaining variable. We denote the chaining variable before absorbing an input block p_i by s_i. Its initial value is $s_0 = 0^b = (0^r, 0^c)$. Then for each block p_i, it performs a two-step transition. First, it moves to the node s' within the same supernode with $\overline{s'} = \overline{s_i} \oplus p_i$, and then it follows the edge starting from s', arriving in s_{i+1}. After absorbing all blocks of p it arrives in node $s_{|p|}$ with $|p|$ denoting the number of blocks in p. Then it gives the outer part of $s_{|p|}$ as output z_0. For each additional block z_i squeezed it follows the edge from $s_{|p|+i-1}$ arriving in $s_{|p|+i}$ and gives out the outer part of the latter as z_i. Note that this can be considered a special case of the above two-step transition if we extend p with blocks $p_{|p|+i} = 0^r$ for all $i \geq 0$.

Clearly, the chaining variable s_i is completely determined by the first i blocks of p. We call this a *path* to s_i. Or more exactly:

Definition 1 ([3]). *First, the empty string is a path to the node* $0^b = (0^r, 0^c)$. *Then, if* p *is a path to node* s *and there is an edge* $((\overline{s}\oplus a, \widehat{s}), t)$ *in the node graph,* $p' = p||a$ *is a path to node* t.

Note that although a path completely determines a node, there may be many paths to a node. A pair of different paths to the same supernode is called an *inner collision.*

It follows from the above that the j-th output block z_j of $z = \text{SPONGE}[f,\text{pad},r](p)$ is the outer part of the node with path $p||0^{rj}$. And so, given a path p (different from 0^{rj}) to a node s, one can find its outer part by a call to the sponge construction. We have $\overline{s} = z_j$ with $z = \text{SPONGE}[f,\text{pad},r](p')$ and p' and j given by $p = p'||0^{rj}$ such that p' is a valid sponge input, i.e., $|p'| > 0$ and $p'_{|p'|-1} \neq 0^r$. For a path of form 0^{rj} there is no such p' and hence the sponge construction cannot be queried to obtain $\overline{s}$.

2.3 The keyed sponge

We define a keyed sponge as a sponge function in which every input presented to the sponge is first pre-pended with a secret key K:

$$\text{KeyedSponge}[f,\text{pad},r,K](M,\ell) = \text{SPONGE}[f,\text{pad},r](K||M,\ell).$$

We write $Z = \text{KeyedSponge}(M,\ell)$ and assume the parameters are clear from the context.

3 Indistinguishability from a random oracle

Indistinguishability deals with the ability of telling apart two systems, typically a concrete construction and an ideal system. An adversary can query the two systems, one at the left and one at the right, and a priori does not know which system is which. Based on the responses to these queries it shall then decide whether the concrete construction is at the left or at the right. If this is hard, the concrete construction can replace the ideal system in application without any significant loss of security [11].

In this paper we prove upper bounds for the success probability of distinguishing a keyed sponge from a random oracle. We use the definition of random oracle from [2]. A random oracle, denoted $\mathcal{RO}$, takes as input binary strings of any length and returns for each input a random infinite string, i.e., it is a map from $\mathbb{Z}_2^*$ to $\mathbb{Z}_2^\infty$, chosen by selecting each bit of $\mathcal{RO}(x)$ uniformly and independently, for every x.

The success probability depends on the guessing rule used by the adversary. Let R_{KS} be the set of observations for which she guesses that the system she is interacting with is the keyed sponge. The success probability reads

$$\Pr(\text{success}) = \frac{1}{2} + \frac{1}{2} \sum_{x \in R_{\mathrm{KS}}} (\Pr(x \text{ observed}|\textsc{KeyedSponge}) - \Pr(x \text{ observed}|\mathcal{RO})). \tag{1}$$

To maximize the success probability, the adversary has to choose

$$R_{\mathrm{KS}} = \{x \ : \ \Pr(x \text{ observed}|\textsc{KeyedSponge}) \geq \Pr(x \text{ observed}|\mathcal{RO})\}.$$

In that case

$$\begin{aligned}&\sum_{x \in R_{\mathrm{KS}}} (\Pr(x \text{ observed}|\textsc{KeyedSponge}) - \Pr(x \text{ observed}|\mathcal{RO})) \\ &= \frac{1}{2} \sum_{x} |\Pr(x \text{ observed}|\textsc{KeyedSponge}) - \Pr(x \text{ observed}|\mathcal{RO})|.\end{aligned}$$

The quantity $\frac{1}{2}\sum_x |\Pr(x \text{ observed}|\textsc{KeyedSponge}) - \Pr(x \text{ observed}|\mathcal{RO})|$ is called the *distinguishing advantage*.

4 Upper bounds for distinguishing a keyed sponge from a random oracle

In this section we prove an upper bound for the success probability of distinguishing a keyed sponge from a random oracle below the indifferentiability bound. We start by defining the setting, the way we model the knowledge of the adversary and the cost of queries. Then we prove that in the absence of certain events ($\widehat{\mathcal{R}}$-collisions and inner clashes, see Sections 4.3 and 4.6), the advantage of telling apart the output of a sponge function and that of a random oracle is zero. Hence the event x observed in Section 3 is an $\widehat{\mathcal{R}}$-collision or an inner clash. We then provide upper bounds for the probability that a query leads to an $\widehat{\mathcal{R}}$-collision or inner clash assuming no $\widehat{\mathcal{R}}$-collisions or inner clashes occurred in previous queries. Finally, we combine these to prove an upper bound on the advantage of an attack of given data and time complexity.

4.1 The setting

In the context of our proof the adversary shall distinguish between a keyed sponge and a random oracle using their responses to queries. We assume that the adversary has no a priori knowledge about f and that she can learn about f by making queries to it. Hence, the adversary shall distinguish between two systems that each have two components, as illustrated in Figure 2:

– The system at the left is the combination of the function f and the keyed sponge function $x = \textsc{KeyedSponge}(M, \ell)$. The adversary can make queries to both components separately, where the latter in turn calls the former to construct its responses. These are the two different interfaces to the system at the left.
– The system at the right consists of a random oracle $\mathcal{RO}$ and the function f, working independently.

Clearly $\mathcal{RO}$ is independent of f, so the goal of this setting is to show that it is hard to see the dependence of a keyed sponge on its underlying permutation f for an adversary who does not know the key.

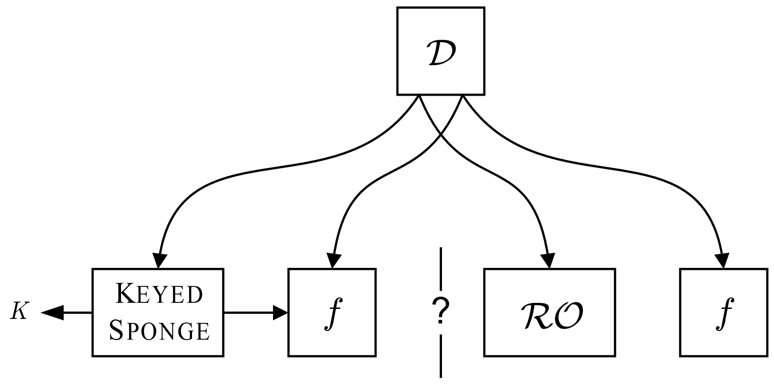

Fig. 2. The distinguishing setup

We give a proof of indistinguishability for the case that f is a random permutation. A random permutation operating on a certain domain is a permutation selected randomly and uniformly from all permutations operating on that domain.

The keyed sponge function KEYEDSPONGE(x, ℓ) provides one interface denoted by $\mathcal{H}$, taking a binary string $x \in \mathbb{Z}_2^*$ and an integer ℓ and returning a binary string $y \in \mathbb{Z}_2^\ell$, the sponge output truncated to ℓ bits. For its execution it can make calls to f and has access to a key K. The permutation f has an interface f that takes as input an element s of $\mathbb{Z}_2^b$ and returns $t = f(s)$ and an interface f^{-1} that takes as input an element s of $\mathbb{Z}_2^b$ and returns $t = f^{-1}(s)$. The union of the two interfaces f and f^{-1} is denoted by $f^{\pm 1}$. Note that the sponge construction in Algorithm 1 only uses the interface f.

At the right is the system with $\mathcal{RO}$ and f. It offers the same interface as the left system, i.e., $\mathcal{RO}$ provides the interface $\mathcal{H}$ and returns an output truncated to the requested length.

4.2 The knowledge of the adversary

The adversary can keep track of what she learns from the responses received to the queries she sent. We represent this knowledge in two graphs: one representing her knowledge on the keyed sponge or random oracle: the so-called *$\mathcal{H}$-knowledge graph*, and the other representing her knowledge on f: the *f-knowledge graph*.

The f-knowledge graph is a subgraph of the one discussed in Section 2.2. It has a number of edges corresponding to the past queries to $f^{\pm 1}$ and for each edge the adversary knows the value of the node where it starts and the node where it arrives. We denote the set of nodes (and the corresponding set of values) with an outgoing edge by $\mathcal{F}_o$ and the set of nodes with an incoming edge by $\mathcal{F}_i$.

We denote the set of supernodes (and the corresponding set of inner values) with an outgoing edge by $\widehat{\mathcal{F}_o}$. Similarly, we denote the set of supernodes (and the corresponding set of inner values) with an incoming or outgoing edge by $\widehat{\mathcal{F}}$. We say a node s is in $\widehat{\mathcal{F}}$ if the supernode $\widehat{s}$ is in $\widehat{\mathcal{F}}$.

Figure 3 gives an example of an f-knowledge graph. In this example every supernode has 4 nodes. The nodes in $\mathcal{F}_o$ or in $\mathcal{F}_i$ are represented with gray circles. All the nodes displayed in the figure are in $\widehat{\mathcal{F}}$.

The $\mathcal{H}$-knowledge graph can be seen as a *blinded* subgraph of the one discussed in Section 2.2. Here blinded means that the adversary does not know the inner part of its (super-)nodes except the root. Also, the adversary only knows the outer parts of the nodes that are traversed by the sponge function in the squeezing phase.

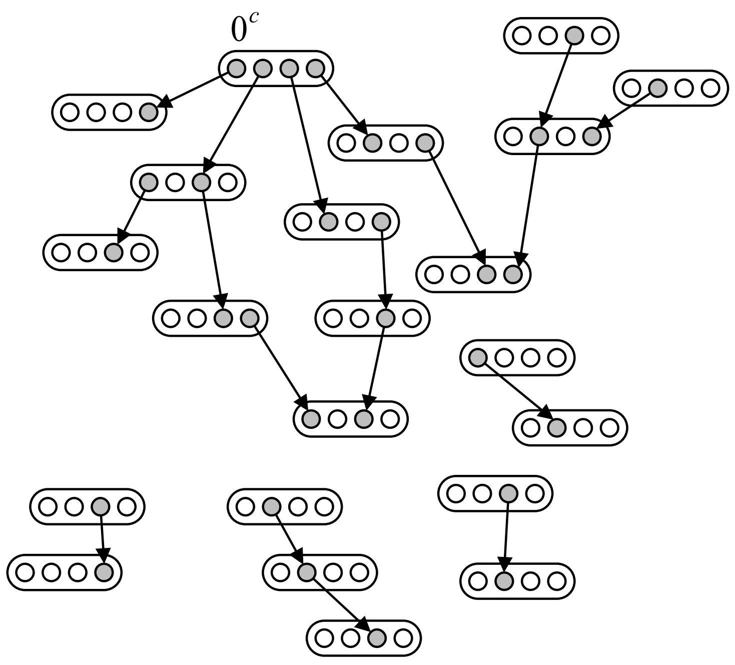

Fig. 3. Example of f-knowledge graph

In the $\mathcal{H}$-knowledge graph, we distinguish between nodes before or during the absorbtion of the key K and nodes after the absorbtion of K. For this, we need the following definition.

Definition 2. *The state s_K is the state after absorbing the key K. We distinguish two cases, depending whether $|K|$ is a multiple of the rate r.*

- *If $|K|$ is a multiple of r, s_K is the node obtained after absorbing K. (Hence K is a path to s_K.) Then, $\widehat{s_K}$ is the set of nodes with the same inner value as that of s_K.*
- *Otherwise, let us define $K = K_0||K_1$, where $|K_0|$ is a multiple of r and $|K_1| < r$. If we note s_{K_0} the state of the sponge function after absorbing K_0, then $s_K = s_{K_0} \oplus (K_1||0^{r+c-|K_1|})$ and $\widehat{s_K}$ is the set of nodes with the same inner value and the same first $|K_1|$ bits as that of s_K.*

Since the key is fixed, any query to $\mathcal{H}$ starts from s_K. So, starting from edges coming out of $\widehat{s_K}$ and going downstream in the $\mathcal{H}$-knowledge graph, we denote the set of nodes in the $\mathcal{H}$-knowledge graph with an incoming edge by $\mathcal{R}_i$ and the set of nodes with an outgoing edge by $\mathcal{R}_o$. Additionally, we denote the set of supernodes containing a node in $\mathcal{R}_i$ by $\widehat{\mathcal{R}_i}$. We say a node s is in $\widehat{\mathcal{R}_i}$ if the supernode $\widehat{s}$ is in $\widehat{\mathcal{R}_i}$.

We define $\widehat{\mathcal{R}}$ as the union of $\widehat{s_K}$ and the set of nodes with inner value equal to those in $\mathcal{R}_i$, or equivalently

$$\widehat{\mathcal{R}} = \widehat{s_K} \cup \widehat{\mathcal{R}_i}.$$

Figure 4 gives an example of a $\mathcal{H}$-knowledge graph. The nodes in $\mathcal{R}_o$ or in $\mathcal{R}_i$ are represented with gray circles. The nodes in $\widehat{\mathcal{R}}$ are explicitly delimited. In this example, the key spans exactly two blocks.

The definitions of $\widehat{s_K}$ and $\widehat{\mathcal{R}}$ are chosen to cover both the cases where the key length is a multiple of the rate r and when it is not. In the latter case, a block containg both the last bits of the key (K_1 in Definition 2) and the first bits of the input message M must be absorbed. We can think of this in two steps.

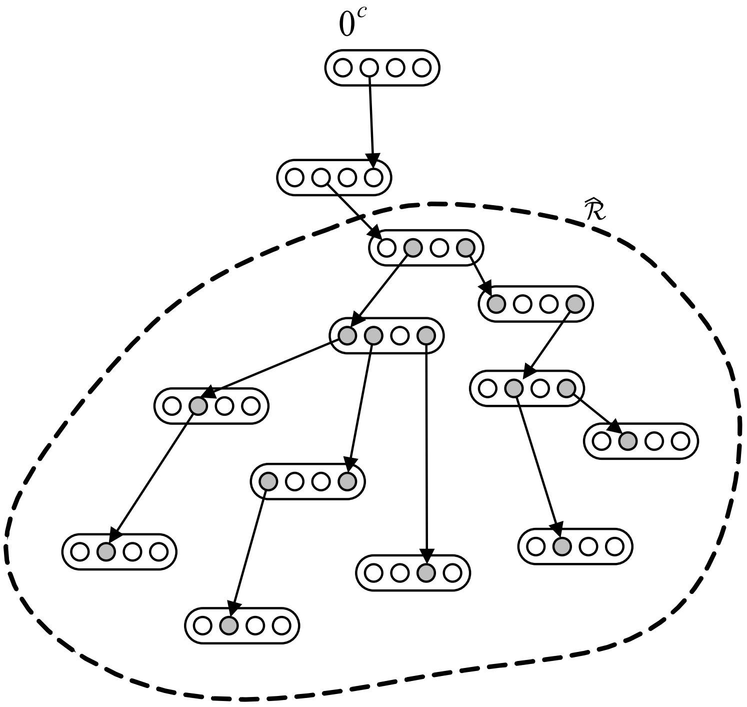

Fig. 4. Example of $\mathcal{H}$-knowledge graph

1. The last $|K_1|$ bits of the key are XORed into the first bits of the state, namely $s_K = s_{K_0} \oplus (K_1||0^{r+c-|K_1|})$ as in Definition 2.
2. The first block of the message has a length of only $r - |K_1|$ bits, which are XORed starting from position $|K_1|$ in the state s_K. The underlying permutation f is then applied on this state.

The definition of $\widehat{s_K}$ incorporates this by considering not only the inner part of s_K but also the first $|K_1|$ bits. One can think of this as a special definition of the inner value in this particular case. If a node s has as last c bits the last c bits of s_K, then the inner parts are considered equal if and only if the first $|K_1|$ bits are also equal.

As an example, assume that $|K| < r$ so that the key does not fill even a single block. In this case, $s_K = K||0^{r-|K|}||0^c$ and $\widehat{s_K} = \{K||u||0^c, u \in \mathbf{Z}_2^{r-|K|}\}$.

4.3 When the adversary is successful

We now define two events, for which we consider the adversary to be successful if one of them occurs.

Definition 3. *An* $\widehat{\mathcal{R}}$-collision *occurs if* $\mathcal{H}$ *is the keyed sponge and two edges of the* $\mathcal{H}$*-knowledge graph within* $\widehat{\mathcal{R}}$ *arrive in two nodes with equal inner values.*

An $\widehat{\mathcal{R}}$-collision is a sub-case of an inner collision, which does not occur with a random oracle. Hence, this allows the distinguisher to detect a keyed sponge.

Definition 4. *An* inner clash *occurs if* $\mathcal{H}$ *is the keyed sponge and if due to a query we get* $\widehat{\mathcal{R}} \cap \widehat{\mathcal{F}} \neq \emptyset$.

An inner clash does not happen in a random oracle. If an inner clash occurs in a keyed sponge, the adversary can start comparing the $\mathcal{H}$- and f-knowledge graphs. This allows her to determine the inner part of a node in $\mathcal{R}_i$ and known outer part, and to distinguish the keyed sponge from a random oracle quickly with some queries to f.

We neglect the cost of the adversary necessary to detect an $\widehat{\mathcal{R}}$-collision or an inner clash. Although we do not explicitly do this in the sequel, we could model this by saying that the interface $\mathcal{H}$ is augmented with a Boolean piece of information telling that an $\widehat{\mathcal{R}}$-collision in the $\mathcal{H}$-knowledge graph has occured. In the ideal world, $\mathcal{RO}$ always answers negatively and the distinguisher is successful as soon as the keyed sponge acknowledges the $\widehat{\mathcal{R}}$-collision. Similary, the $f^{\pm 1}$ interface could tell if $s \in \widehat{\mathcal{R}}$ or if $t \in \widehat{\mathcal{R}}$ for a query $t = f(s)$ or $s = f^{-1}(t)$. In the ideal world, it would always be negative.

4.4 Key guessing

Unless the distinguisher correctly guesses s_K, the inner values in $\widehat{\mathcal{R}}$ are unknown to the distinguisher.

On the one hand, the distinguisher can have an a priori knowledge of s_K. In a typical case, $|K| < c$, hence $\widehat{s_K}$ is not uniformly distributed over all 2^c possible inner values.

On the other hand, all the inner values in $\widehat{\mathcal{R}_i}$ are completely unknown to the distinguisher unless she makes a query to f (resp. to f^{-1}) with an input $s \in \widehat{\mathcal{R}}$ (resp. $t \in \widehat{\mathcal{R}}$). If $\widehat{\mathcal{R}} \cap \widehat{\mathcal{F}} = \emptyset$, the probability from her point of view that a given inner value of a node in $\widehat{\mathcal{R}_i}$ has a given value $\widehat{s}$ is only determined by the number of queries made to $f^{\pm 1}$. This follows immediately from the fact that she cannot determine the output values of $f^{\pm 1}$ until she actually makes a query to it.

From this, we deduce two main approaches for the distinguisher: either she tries to determine s_K or she tries to distinguish the keyed sponge not knowing the inner values in the $\mathcal{H}$-knowledge graph.

An important point to note is that a correct key guess implies an inner clash. The adversary can verify that her guess is correct via queries to f. If so, she will simulate the keyed sponge and hit a node in $\widehat{s_K}$. Referring to Section 4.3, we assume that the adversary wins as soon as she tests a state within $\widehat{s_K}$ (i.e., causing an inner clash), so this automatically includes a correct key guess. (And this includes whether or not $|K|$ is a multiple of r.)

In this subsection, we treat the case the distinguisher tries to determine s_K and produce an inner clash. Afterwards, the rest of the proof covers the case where s_K is unknown (indicated by "UK").

Since f is a publicly known function, we assume that $s_K = g(K)$ is a function known to the adversary. We allow her to evaluate $g(K)$ directly (i.e., not to query f specifically for this), but we do require that she tests if her guess is correct through the f interface. Referring to the next section, this means that evaluating $g(K)$ has no cost but checking whether the guess is correct has a cost 1. Counting the cost of evaluating $g(K)$ would depend on the number of blocks of K. In the case $|K| < r$, $g(K) = K||0^{r-|K|}||0^c$ and there would be no query necessary to evaluate it. Hence, leaving this cost for free to the adversary allows to have a bound that does not depend on r.

Unless $|K| < r$, the function $\widehat{s_K} = \widehat{g}(K)$ that returns the inner value of s_K can have collisions. The range of $\widehat{g}(K)$ may have a size smaller than $2^{|K|}$. Let $P_{\text{key}}(i)$ be the probability of guessing the key after i queries, or more precisely

$$P_{\text{key}}(i) = \Pr[\widehat{s_K} \cap \widehat{\mathcal{F}_o} \neq \emptyset \text{ after } i \text{ queries}].$$

The adversary can test first the values for which multiple values of K collide to the same $\widehat{s_K}$. So even if K is uniformly distributed over $\mathbf{Z}_2^{|K|}$, we can have $P_{\text{key}}(i) - P_{\text{key}}(i-1) = m_i 2^{|K|}$ with

m_i the multiplicity of the i-th query. In the end, the security will be limited by the complexity of the exhaustive key search $2^{|K|}$. Hence, in the case that $|K| < c$, the proportion of queries for which $m_i > 1$ will be small and we nevertheless assume that the expected time workload of key guessing is $2^\kappa/2$ for $\kappa \approx |K|$.

In the case $|K| < r$, we have $s_K = K||0^{r-|K|}||0^c$ and there is no collision, so $\kappa = |K|$ if K is chosen uniformly.

4.5 The cost of queries

The bound we provide in this paper is on the success probability of a distinguishing attack as a function of the cost function that models the effort from the adversary. We define a cost function similar to the one used in the indifferentiability bounds [4]. However, here the cost consists of two variables, which model different aspects of an attack:

Data complexity This models queries sent to the keyed instance. In a setup attacking a concrete cryptosystem, this models the amount of access to the keyed instance. In practice one can limit the adversary's access to the instance. This is sometimes also referred to as the online complexity.

Time complexity This models computations requiring no access to the keyed instance. In a concrete setup the only limitation of the time complexity is the computing power and time available to the adversary. This is sometimes also referred to as the offline complexity.

The unit of our cost function is the number of calls to $f^{\pm 1}$:

– The time complexity N is the number of queries to $f^{\pm 1}$, where only fresh queries are counted. A query is not fresh if its response is already known due to previous queries to $f^{\pm 1}$.
– The data complexity M is the number of fresh calls to f due to queries to $\mathcal{H}$ as if $\mathcal{H} =$ KeyedSponge. Here a call to $f(s)$ is not fresh if it has already been made due to a prior query to $\mathcal{H}$.

Clearly, the number of edges in the f-knowledge graph is equal to N. Per definition, after a number of queries to $\mathcal{H}$ with a total cost of M, the $\mathcal{H}$-knowledge graph has M edges within $\widehat{\mathcal{R}}$.

The convention of only counting fresh queries/calls only benefits to the adversary and is thus on the safe side regarding security.

4.6 Conditions for uniform and independent output of the keyed sponge

We now prove the following lemma.

Lemma 1. *In the absence of $\widehat{\mathcal{R}}$-collisions in the $\mathcal{H}$-graph and of inner clashes, the response to a query to $\mathcal{H}$ is uniformly and independently distributed.*

Proof. If $\mathcal{H}$ is the random oracle the lemma is true. Otherwise, the query results in a number of new edges in the $\mathcal{H}$-knowledge graph and its response is the concatenation of outer parts of nodes traversed in the squeezing phase. The following reasoning applies for each of the edges that arrives in such a node. Let s be the node where such an edge starts and $t = f(s)$ where it arrives. We compute the probability that the outer part of t has a particular value a, on the condition that the new node t does not introduce an $\widehat{\mathcal{R}}$-collision nor an inner clash. This probability must be taken over the set F of functions f that are compatible with the knowledge on f from all prior queries to $f^{\pm 1}$ and the sponge function. We will prove that this probability is uniform: equal to 2^{-r} independently of the value a.

Due to the requirements related to $\widehat{\mathcal{R}}$-collisions and inner clashes, the set of possible values for t is $\mathbb{Z}_2^b \setminus (\mathbb{Z}_2^r \times (\widehat{\mathcal{R}_i} \cup \widehat{\mathcal{F}})) = \mathbb{Z}_2^r \times (\mathbb{Z}_2^c \setminus (\widehat{\mathcal{R}_i} \cup \widehat{\mathcal{F}}))$. Clearly, this set contains an equal number of elements for each given outer part and hence the outer part of t has a uniform distribution. □

Without the requirement that t shall not result in an $\widehat{\mathcal{R}}$-collision or inner clash, its possible values are given by the set $\mathbb{Z}_2^b \setminus (\mathcal{R}_i \cup \mathcal{F}_i)$. This follows immediately from the fact that s has not been queried before and that f is a permutation. Note that the outer part of t is in general not distributed uniformly. Hence, it is the requirement on the absence of $\widehat{\mathcal{R}}$-collisions and inner clashes that gives the sponge its uniformity in this setting.

From Lemma 1 it follows that the set of observations R_{KS} in Equation (1) coincide with the occurrence of an $\widehat{\mathcal{R}}$-collision or an inner clash. As a random oracle has no finite inner state, these cannot occur for a random oracle and we have:

$$\Pr(\text{success}) = \frac{1}{2} + \frac{1}{2}\Pr(\widehat{\mathcal{R}}\text{-collision or inner clash}|\textsc{KeyedSponge}).$$

4.7 Success probabilities of queries

From Lemma 1 it follows immediately that an adversary can only have non-zero advantage in distinguishing a keyed sponge function from a random oracle if she succeeds in generating an $\widehat{\mathcal{R}}$-collision in the sponge function or an inner clash between queries to $f^{\pm 1}$ and queries to $\mathcal{H}$.

Assume the adversary has performed i fresh queries to $f^{\pm 1}$ and a number of queries to $\mathcal{H}$ with a total cost of j. We now provide bounds on the success probability of the next query as a function of i and j, for the three types of query, assuming that no $\widehat{\mathcal{R}}$-collision or inner clash has occurred yet (indicated by “NRC,NIC”).

Queries to $\mathcal{H}$ In the $\mathcal{H}$-knowledge graph, a query results in the addition of a number of edges and nodes. Rather than computing an upper bound for the success probability for a full query, we compute it for the success probability for each edge added in the $\mathcal{H}$-knowledge graph, i.e., per increment of the cost by 1. More specifically, we compute an upper bound on the probability that an $\widehat{\mathcal{R}}$-collision or an inner clash occurs when adding an edge to the $\mathcal{H}$-knowledge graph.

Consider the addition of an edge from s to a new node t. In $\widehat{\mathcal{R}}$, there are up to $(2^r-1)j+2^r$ nodes in which t can create an $\widehat{\mathcal{R}}$-collision, including all the nodes in the supernode $\widehat{s_K}$. The total number of values that t can take, compatible with f being a permutation, is at least $2^{r+c}-(i+j)$. Hence,

$$\Pr(\widehat{\mathcal{R}}\text{-collision}|\text{NRC,NIC,UK}) \leq \frac{(2^r-1)j+2^r}{2^{r+c}-(i+j)}. \tag{2}$$

We start from a situation where there is no inner clash, so $s \notin \widehat{\mathcal{F}}$. For the success probability of an inner clash, we need to evaluate the probability that t ends up in $\widehat{\mathcal{F}}$. In the f-knowledge graph, there are up to 2^r nodes per supernode in which t can create an inner clash, for up to $2i$ different inner values. The total number of values that t can take, compatible with f being a permutation, is at least $2^{r+c}-(i+j)$. Hence,

$$\Pr(\text{inner clash}|\text{NRC,NIC,UK}) \leq \frac{2^{r+1}i}{2^{r+c}-(i+j)}. \tag{3}$$

Queries to $f^{\pm 1}$ A query to $f^{\pm 1}$ cannot result in an $\widehat{\mathcal{R}}$-collision. In the following lemma we provide an upper bound to the probability of an inner clash of queries to f and f^{-1}.

For f, if $\widehat{s_K}$ is unknown, either the adversary by chance chooses $s \in \widehat{\mathcal{R}}$ or she does not. In the first case, we compute the probability of choosing one of the inner values in $\widehat{\mathcal{R}}$. There are up to $j+1$ such values and $2i$ values are known not to be in this set, so we get

$$\Pr(s \in \widehat{\mathcal{R}}|\text{NRC,NIC,UK}) = \frac{j+1}{2^c-2i}.$$

For the second case, in the $\mathcal{H}$-knowledge graph there are up to 2^r nodes per supernode in which t can create an inner clash, for up to $j+1$ different inner values. The total number of values that t can take, compatible with f being a permutation, is at least $2^{r+c}-(i+j)$. Hence,

$$\Pr(t \in \widehat{\mathcal{R}} | s \notin \widehat{\mathcal{R}}, \text{NRC,NIC,UK}) = \frac{2^r(j+1)}{2^{r+c}-(i+j)}.$$

Hence,

$$\Pr(s \in \widehat{\mathcal{R}} \vee t \in \widehat{\mathcal{R}} | \text{NRC,NIC,UK}) \leq \frac{2j+2}{2^c - 2(i+j)}. \tag{4}$$

For f^{-1}, the reasoning is the same, with the roles of s and t inverted. So,

$$\Pr(t \in \widehat{\mathcal{R}} \vee s \in \widehat{\mathcal{R}} | \text{NRC,NIC,UK}) \leq \frac{2j+2}{2^c - 2(i+j)}.$$

4.8 Total success probability

We can now use Eq. (2), (3) and (4) to obtain an upper bound on the success probability of distinguishing for given values of N and M.

The probability of an $\widehat{\mathcal{R}}$-collision is given by the following lemma.

Lemma 2. *The probability of an $\widehat{\mathcal{R}}$-collision after a sequence of queries to $\mathcal{H}$ with cost M and queries to $f^{\pm 1}$ with cost N is upper bounded by:*

$$\Pr(\widehat{\mathcal{R}}\textit{-collision}) \leq 1 - \exp\left(\frac{-M^2}{2^{c+1}}\right)$$

Proof. Let i_j be the number of $f^{\pm 1}$ queries performed before query j to $\mathcal{H}$. The probability that an $\widehat{\mathcal{R}}$-collision occurs in the $\mathcal{H}$-knowledge graph is given by:

$$\begin{aligned}\Pr(\widehat{\mathcal{R}}\text{-collision}) &= 1 - \prod_{j=0}^{M-1}\left(1 - \frac{(2^r-1)j + 2^r}{2^{r+c}-(i_j+j)}\right)\\ &= 1 - \prod_{j=0}^{M-1}\left(1 - \frac{\frac{j+1}{2^c} - \frac{j}{2^{r+c}}}{1 - \frac{i_j+j}{2^{r+c}}}\right)\end{aligned}$$

Using the approximation $\log(1-\epsilon) \approx -\epsilon$ for $\epsilon \ll 1$ and neglecting the last term in the denominator by exploiting $i_j + j \ll 2^{r+c}$ yields

$$\Pr(\widehat{\mathcal{R}}\text{-collision}) = 1 - \exp\left(-\sum_{j=0}^{M-1} \frac{j+1}{2^c} - \frac{j}{2^{r+c}}\right) = 1 - \exp\left(-\frac{M(M+1)}{2^{c+1}} + \frac{M(M-1)}{2^{r+c+1}}.\right)$$

This is the same expression as the success probability for generating inner collisions in a random P-sponge obtained in [3] and the advantage of differentiating a random P-sponge from a random oracle in [4]. For $M \gg 1$ and $2^r \gg 1$, this simplifies to

$$\Pr(\widehat{\mathcal{R}}\text{-collision}) \leq 1 - \exp\left(\frac{-M^2}{2^{c+1}}\right).$$

□

An upper bound to the probability of an inner clash is given by the following lemma.

Lemma 3. *The probability of an inner clash in a sequence of queries to $\mathcal{H}$ with cost M and queries to $f^{\pm 1}$ with cost N for an adversary with no knowledge on the inner parts of the nodes in $\widehat{\mathcal{R}}$ is upper bounded by:*

$$\Pr(\textit{inner clash}|\,UK) \leq 1 - \exp\left(\frac{-2(M+1)(N+1)}{2^c}\right).$$

Proof. For generating inner clashes the success probability of the k-th query depends on whether it is a query to $\mathcal{H}$ or to $f^{\pm 1}$. Let $\delta_k = 1$ if the k-th query is one to $f^{\pm 1}$ and $\delta_k = 0$ otherwise. Let now i_k be the number of f-queries performed before the k-th query and j_k the number of $\mathcal{H}$-queries performed before the k-th query. We have $i_k = \sum_{\ell<k} \delta_\ell$ and $j_k = \sum_{\ell<k}(1-\delta_\ell)$. We can combine Eq. (3) and (4) into:

$$\Pr(\text{inner clash in } k\text{-th query}|\text{UK}) \leq \frac{2(1-\delta_k)i_k + 2\delta_k j_k + 2}{2^c - 2k}.$$

The total inner clash probability is now upper bounded by:

$$\Pr(\text{inner clash}|\text{UK}) \leq 1 - \prod_{k=1}^{M+N}\left(1 - \frac{2(1-\delta_k)i_k + 2\delta_k j_k + 2}{2^c - 2k}\right).$$

Using the approximation $\log(1-\epsilon) \approx -\epsilon$ for $\epsilon \ll 1$ and assuming $k \ll 2^c$ yields

$$\Pr(\text{inner clash}|\text{UK}) \leq 1 - \exp\left(-\frac{2}{2^c}\sum_{k=1}^{M+N}(1-\delta_k)i_k + \delta_k j_k + 1\right). \tag{5}$$

Working out the sum in the righthand member of equation (5) yields

$$\sum_k (1-\delta_k)i_k + \delta_k j_k = \sum_k \sum_{\ell<k}(1-\delta_k)\delta_\ell + \sum_k \sum_{k<\ell} \delta_\ell(1-\delta_k) = MN.$$

Filling this in in equation (5) yields the result. □

The success probability in Lemma 3 is under the condition that the adversary has no knowledge on the inner parts of nodes in $\widehat{\mathcal{R}}$. We must therefore add to this value the probability of correctly guessing $\widehat{s_K}$ after N queries to $f^{\pm 1}$. Referring to Section 4.4, this is $P_{\text{key}}(N)$.

Combining this with Lemma 2 and Lemma 3 results in the main theorem of this paper.

Theorem 1. *The advantage of distinguishing* KeyedSponge$[f, K]$ *from a random oracle, with f a random permutation and K uniformly distributed, is upper bounded by:*

$$1 - \exp\left(-\frac{\frac{M^2}{2} + 2(M+1)(N+1)}{2^c}\right) + P_{key}(N)\ ,$$

where M is the data complexity, N the time complexity and $P_{key}(N)$ the probability of guessing the key after N queries.

Inner collisions are likely to occur only if the data complexity M approaches $2^{c/2}$. If the data complexity is limited to a value $M \ll 2^{c/2}$, the expected time complexity to distinguish a keyed sponge from a random oracle is about $N \sim \min(2^c/M, 2^\kappa)$.

5 Discussion and conclusions

Theorem 1 has implications for all use cases where a keyed sponge function is used. In fact it imposes an upper bound to the success probability of attacks that are generic in f. We discuss some interesting use cases in the following subsections.

Note that upper bounds for the success probability of state retrieval attacks for keyed sponges have been proved in [6]. Those bounds are tighter but are less generic than that of Theorem 1.

5.1 A safety margin beyond the indifferentiability bound

The bound provided in Theorem 1 only holds when the keyed sponge makes use of a random permutation. When a concrete permutation is taken, no such bounds can be given. See for example [9] and also [11] for discussions on this subject. However, it guarantees that using the sponge construction excludes generic attacks with a success probability higher than the sum of our bound and the success probability the attack would have for a random oracle.

The results of this paper show that one can expect a higher security against generic attacks from a keyed sponge than from a sponge function with all its possible uses. In the latter case, the worst-case success probability must be considered, which is determined by the sponge indifferentiability result of [4] and a bound of $N^2/2^{c+1}$. The bound in this paper shows that when being used in conjunction with a key, in many use cases the success probability of generic attacks is much smaller. Hence, this can be seen as an extra safety margin.

5.2 A smaller size or an increased bitrate

A keyed sponge can be used for message authentication code (MAC) computation, stream encryption, authenticated encryption and reseedable pseudorandom sequence generators [7,6]. In these use cases, the results of this paper allow to decrease the capacity for the same level of security. This benefit can be exploited in two possible ways. First, one can keep the bitrate constant and design a permutation f with a smaller state size ($r + c$ bits). This primarily benefits to the implementation of sponge functions on resource-constrained platforms. Second, one can keep the permutation (and its size) unchanged but increase the bitrate when used in a keyed mode.

In many applications we can assume the implementation imposes an upper limit for M with a given key K, e.g., $M < 2^a$ with a the *usage exponent* with a value typically between 20 and 40. This may be due to physical constraints or imposed by the protocol. Assuming such a limit with $a \ll c/2$, the probability of $\widehat{\mathcal{R}}$-collisions becomes negligible and the expected time complexity for distinguishing without key guessing is 2^{c-a-1}. In symmetric primitives that have a key, a typical security requirement is that there should be no attacks faster than exhaustive key search. For a keyed sponge this requirement translates to a lower bound for the capacity:

$$c \geq |K| + a + 1. \tag{6}$$

This is especially relevant in applications with constrained resources. Two examples of sponge functions that are suitable for constrained resources are the sponge function family Quark [1] and the small-width members of Keccak [8]. We illustrate the implications of the new bound with Quark used in application requiring a keyed sponge with a required security level of 80-bits against all attacks, hence with an 80-bit key. According to the $2^{c/2}$ indifferentiability bound this would require a capacity of at least 160 bits, and one would choose d-Quark ($r = 16$ and $c = 160$). Instead, if we assume a usage exponent of at most 39, Equation (6) requires a capacity of only 120 bits. One can then opt either for a faster variant of d-Quark using a higher rate or for a smaller member of the family, namely u-Quark ($r = 8$ and $c = 128$). In addition, one could also define a faster variant of u-Quark using the same permutation with $r = 16$ and $c = 120$.

References

1. J.-P. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia, *Quark: A lightweight hash*, in Mangard and Standaert [10], pp. 1–15.
2. M. Bellare and P. Rogaway, *Random oracles are practical: A paradigm for designing efficient protocols*, ACM Conference on Computer and Communications Security 1993 (ACM, ed.), 1993, pp. 62–73.
3. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, *Sponge functions*, Ecrypt Hash Workshop 2007, May 2007, also available as public comment to NIST from `http://www.csrc.nist.gov/pki/HashWorkshop/Public_Comments/2007_May.html`.
4. ———, *On the indifferentiability of the sponge construction*, Advances in Cryptology – Eurocrypt 2008 (N. P. Smart, ed.), Lecture Notes in Computer Science, vol. 4965, Springer, 2008, `http://sponge.noekeon.org/`, pp. 181–197.
5. ———, *Duplexing the sponge: single-pass authenticated encryption and other applications*, Second SHA-3 candidate conference, August 2010.
6. ———, *Sponge-based pseudo-random number generators*, in Mangard and Standaert [10], pp. 33–47.
7. ———, *Cryptographic sponge functions*, January 2011, `http://sponge.noekeon.org/`.
8. ———, *The* Keccak *reference*, January 2011, `http://keccak.noekeon.org/`.
9. R. Canetti, O. Goldreich, and S. Halevi, *The random oracle methodology, revisited*, Proceedings of the 30th Annual ACM Symposium on the Theory of Computing, ACM Press, 1998, pp. 209–218.
10. S. Mangard and F.-X. Standaert (eds.), *Cryptographic hardware and embedded systems, CHES 2010, 12th international workshop, Santa Barbara, CA, USA, August 17-20, 2010*, Lecture Notes in Computer Science, vol. 6225, Springer, 2010.
11. U. Maurer, R. Renner, and C. Holenstein, *Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology*, Theory of Cryptography - TCC 2004 (M. Naor, ed.), Lecture Notes in Computer Science, no. 2951, Springer-Verlag, 2004, pp. 21–39.