Permutation-based encryption, authentication and authenticated encryption

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

[1] STMicroelectronics
[2] NXP Semiconductors

Abstract. While mainstream symmetric cryptography has been dominated by block ciphers, we have proposed an alternative based on fixed-width permutations with modes built on top of the sponge and duplex construction, and our concrete proposal KECCAK. Our permutation-based approach is scalable and suitable for high-end CPUs as well as resource-constrained platforms. The latter is illustrated by the small KECCAK instances and the sponge functions Quark, Photon and Spongent, all addressing lightweight applications. We have proven that the sponge and duplex construction resist against generic attacks with complexity up to $2^{c/2}$, where c is the capacity. This provides a lower bound on the width of the underlying permutation. However, for keyed modes and bounded data complexity, a security strength level above $c/2$ can be proven. For MAC computation, encryption and even authenticated encryption with a passive adversary, a security strength level of almost c against generic attacks can be attained. This increase in security allows reducing the capacity leading to a better efficiency. We argue that for keyed modes of the sponge and duplex constructions the requirements on the underlying permutation can be relaxed, allowing to significantly reduce its number of rounds. Finally, we present two generalizations of the sponge and duplex constructions that allow more freedom in tuning the parameters leading to even higher efficiency. We illustrate our generic constructions with proposals for concrete instantiations calling reduced-round versions of the KECCAK-f[1600] and KECCAK-f[200] permutations.

1 Introduction

In the last decades, mainstream symmetric cryptography has been dominated by block ciphers: block cipher modes of use have been employed to perform encryption, MAC computation and authenticated encryption. Moreover, most hash functions internally call a compression function with a block cipher structure at its kernel. From a design perspective these hash functions merely consist of block ciphers in some dedicated mode of use. One could argue that the "swiss army knife" title usually attributed to the hash function belongs to the block cipher.

In the last five years, we have proposed new modes of use for, a.o., hashing, MAC computation and (plain or authenticated) encryption that make use of a fixed-width permutation instead of a block cipher [5,7,6]. These modes make use of the sponge and duplex constructions, illustrated in Figures 1 and 2.

In both constructions the width b of the underlying permutation is split in two: an outer part with size r and an inner part with size c. The rate r determines the efficiency of the construction and the capacity c the attainable security strength, so for a given permutation with width b, the equation $b = c + r$ expresses a trade off between security and efficiency.

The first concrete instantiation of such a permutation-based sponge function has been our design KECCAK [10]. With its seven associated permutations, it goes from a toy primitive to a wide sponge function. In the meanwhile several other sponge functions have been proposed: Quark [2], Photon [20] and Spongent [12]. Remarkably, the latter three were all put forward as lightweight hash functions. All four designs are based on permutation families covering multiple widths rather than a single width. All together, these permutations cover a large number of widths ranging from 25 to 1600 bits.

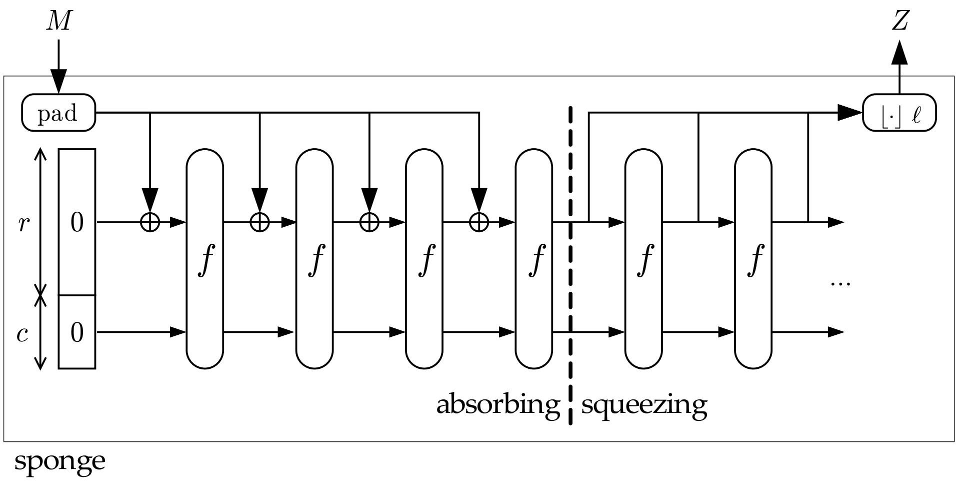

Fig. 1. The sponge construction

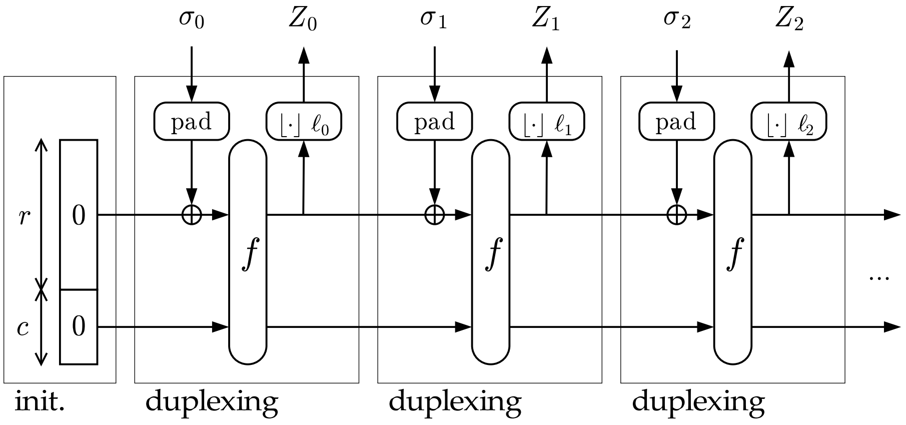

Fig. 2. The duplex construction

Additionally, numerous other cryptographic designs have iterated permutations at their core, such as the hash function Grøstl [19] whose specification counts two 512-bit permutations P_{512} and Q_{512} and two 1024-bit permutations P_{1024} and Q_{1024}, JH [27] that makes use of a 1024-bit permutation called E8, or the stream ciphers Salsa and ChaCha [3] that are based on 512-bit permutations. Moreover, most block ciphers can be converted into an iterated permutation by fixing the key to some constant. All these permutations can be used in sponge functions and duplex objects.

In this note, we focus on the intuition behind our parameter choices rather than formal proofs. The outline is as follows. In Section 2, we look at the generic security of keyed sponge and duplex modes and exploit the known results to increase the efficiency for authentication and (plain or authenticated) encryption. In section 3 we argue that cryptanalytic results provide evidence that primitives are much harder to attack in keyed modes than in un-keyed modes and use that observation to propose keyed modes based on reduced-round variants of Keccak-f. Finally, we propose in Section 4 variants dedicated to authentication (based on Alred) and (authenticated or plain) encryption.

Throughout the text, we use the concept of *security strength* as used by NIST in its cryptography standards. It is defined in [22] as *a number associated with the amount of work (that is, the number of operations) that is required to break a cryptographic algorithm or system.* Security strength levels are expressed in bits and a level of n bits implies that the amount of work to break the system is of the order 2^n operations. As exhaustive key search of an n-bit key takes an amount of work 2^n, we will adopt for a target security strength of n, keys of length n. In its standards NIST targets five specific security strength levels: 80, 112, 128, 192 and 256 bits. The concrete instances we propose in this note address 80 and 128 bits for the lightweight instances and 128 and 256 bits for the other ones.

2 Increasing the rate in the sponge and duplex constructions

Using the indifferentiability framework we have proven that the sponge and duplex constructions are secure against generic attacks with complexity below $2^{c/2}$ [4]. With this bound, a desired security strength level of 80 bits implies a capacity of 160 bits and hence imposes a minimum width for the permutation, which may hinder lightweight applications. Moreover, permutations with a width just slightly above 160 bits will inevitably lead to small rates.

When a sponge function or duplex object is used in conjunction with a key, one can prove more refined bounds taking into account the data complexity. In [8] we have proven that if the data complexity is limited to 2^a r-bit blocks, the keyed mode withstands generic attacks with time complexity up to 2^{c-a} calls of the underlying permutation. If $a < c/2$, this results in an increase of the security strength from $c/2$ to $c - a$.

The bound $c - a$ assumes a very powerful adaptive adversary and the intuition behind it is the following. Given sufficient output of a keyed sponge (or duplex) object, an attacker can make guesses for the inner c bits of the state and verify for each guess whether it is consistent with the observed output. In keyed modes of the sponge or duplex construction, the knowledge of the inner part of the state is as valuable as knowing the key. The probability of success of a single guess is 2^{-c} and hence for typical values of r, i.e. $r \geq 8$, the expected workload of this attack is very close to 2^{c-1} executions of the underlying permutation f. If $r > c$, this corresponds with generically solving a constrained-input constrained output (CICO) problem [6] for f with c unknown bits at its input and c unknown bits at its input. In general an adaptive attacker can reduce this expected workload by a factor close to 2^a at the cost of 2^a adaptively chosen sponge (or duplex) input blocks by converting this CICO problem in a multi-target CICO problem. She must just apply inputs to the keyed sponge or duplex instance in such a way that the outer r-bit parts of the state at the input of f has some chosen value for multiple executions of f. In the duplex mode this is in general not difficult. The $r - 2$ outer bits can be fixed to zero by feeding as σ in a duplexing call the first $r - 2$ bits of the output Z of the previous duplexing call.

If the adversary can force M executions of f with the *same* outer r bits but *different* inner c bits, the probability of success for a guess becomes $M2^{-c}$ instead of 2^{-c}, reducing the expected workload roughly by a factor M. We call M the multiplicity of the attack. The bound 2^{c-a} is a consequence of the fact that in the worst case the multiplicity M may come very close to the data complexity 2^a.

In specific use cases an adversary may not have the possibility to enforce the r outer bits to some fixed value and hence achieving a high multiplicity may be out of reach. She can count on luck to have collisions in the r outer bits and from observed sponge or duplex output blocks extract the outer value that occurs most often. The number of times this outer value is observed is then the multiplicity. If $a < 2r$, M is expected to be only 1 or

2. Once a comes close to r this value starts to grow. Some use cases in which the attacker cannot enforce the values of the outer bits are the following:

– MAC computation with the sponge construction where the input is the key in a first sequence of blocks, followed by the message in following blocks. In this case the attacker can choose the message input blocks, but does not know the r outer bits of the state prior to their absorbing. Note that by MACing many messages with the same first message block one can enforce the outer part of the state after absorbing the first message block to be the same, but this also holds for the inner part so it does not contribute to the multiplicity.
– Keystream generation with the sponge construction where the input is a key followed by a nonce. In this case specifying the same nonce will also not contribute to the multiplicity.
– Authenticated encryption with the duplex construction (SpongeWrap), with an adversary that does not have active access to the message encryptor. Clearly, in that case even an adversary that knows the plaintext can only observe duplex inputs and outputs and not choose their values. Note that access to the message decryptor will typically only provide plaintexts upon receiving messages with a valid tag.

We have discussed this issue under the name of the passive state recovery problem and proven an upper bound for the generic success probability in [5] .

Clearly, these security bounds decrease the required capacity for a given target security strength and hence open up to smaller permutations or higher rates.

If we look at real-world constraints, we think assuming $M \leq 2^{64}$ for any attacker is reasonable. Going beyond this limit would imply an attacker that is able to present over 2^{64} chosen input blocks to keyed duplex instances under attack and manage the outputs. We will assume this limit in our choice of parameters in the remainder of this note. Generalizing to other values is however straightforward.

3 Reducing the number of rounds in the underlying permutation

In the design of Keccak we have chosen for a large safety margin by taking a number of rounds in Keccak-f that is almost the double of what we estimate to be sufficient for the absence of shortcut attacks more efficient than generic attacks.

In the published cryptanalysis of concrete primitives we observe that most primitives offer a much higher resistance against attacks in keyed modes than in unkeyed modes. The typical examples of attacks on an unkeyed hash function are the generation of collisions and second pre-images. Attacking a keyed mode ranges from key retrieval attacks to distinguishers. Note that also determining a first pre-image can be seen as an attack on a keyed mode, where the pre-image is the key. Examples that illustrate this difference in resistance between keyed and unkeyed modes include:

– MD5 [24]: despite painstaking efforts there is little progress in MD5 pre-image generation [25] while MD5 collisions [26] have been generated that are practically exploitable
– Panama [15]: the Panama stream cipher is as yet unbroken while for the Panama hash function collisions can be generated instantaneously [23,13]
– Keccak: In the Keccak crunchy crypto contest [9] collision challenges have been broken up to 4 rounds while pre-image challenges have been broken only up to 2 rounds.

Clearly, the situations for an adversary attacking an un-keyed instance of a sponge function (e.g., used for collision-resistant hashing) and that of attacking a keyed instance are very different. In a keyed instance, after the key has been absorbed, the inner c bits of the state are unknown to the attacker. In sponge-based MAC generation, during the absorbing phase even the complete state is unknown to the attacker.

In the remainder of this paper we explore how the safety margin in Keccak can be relaxed for keyed applications in the light of these facts. We compare the performance of these instances with Keccak[], the Keccak instance with default parameters $r = 1024$ and $c = 576$ and Keccak[r=40, c=160]. Of course the same exercise can be conducted for Quark, Photon, Spongent or in general any iterated permutation used in a keyed sponge or duplex mode.

In this section we limit ourselves to applying the standard sponge and duplex constructions to round-reduced versions of Keccak-f. Note that such round-reduced versions are covered by the Keccak specifications in [10] and the Keccak reference code [11]. To avoid confusion with Keccak instances in which Keccak-f has the nominal number of rounds, we denote these primitives by the name Keccup.

Definition 1. Keccup-$f[b, n]$ *is a family of permutations parameterized by its width b and its number of rounds n, where* Keccup-$f[b, n]$ *is identical to* Keccak-$f[b]$ *reduced to n rounds [10]. Similarly,* Keccup$[r, c, n]$ *is a sponge function or duplex object using* Keccup-$f[r + c, n]$.

As we do not modify the sponge or duplex constructions but just apply them to round-reduced instances of Keccak-f, the proven security bounds with respect to generic attacks still apply. However, it may well be that this reduction of the number of rounds leads to specific attacks breaking the security claims.

We do the exercise for two Keccup-f permutation widths: 1600 and 200.

3.1 Using Keccup-$f[1600, n]$

We target two security strength levels: 128 and 256. We will assume that the length k of the secret keys corresponds to the security strength.

Consider a duplex instance with a target security strength of $k = 128$ bits. Assuming the multiplicity is bound by $M = 2^a \leq 2^{64}$ results in a capacity of $c = k + a = 192$ bits, leaving 1408 bits of rate. From our experiments related to trail search [10,14], we think that the minimum weights for differential or linear trails over four rounds of Keccak-$f[1600]$ is higher than 64. On the other hand, for higher-order differential cryptanalysis, cube attacks and interpolation attacks, the algebraic degree of Keccak-f increases only by a factor two each round. If we settle for a degree of 1024, 10 rounds is the choice. Using Keccup$[r = 1408, c = 192, n = 10]$ would give a speed-up with respect to Keccak[] of $24/10 \times 1408/1024 \approx 3.3$.

Targeting a security strength of $k = 256$ bits, a similar reasoning leads to a capacity of $c = k + a = 320$ bits and 11 rounds, so Keccup$[r = 1280, c = 320, n = 11]$. Here the speed-up with respect to Keccak[] becomes $24/11 \times 1280/1024 \approx 2.7$.

3.2 Using Keccup-$f[200, n]$

Thanks to their small state size, sponge and duplex instances based on Keccak-$f[200]$ are well suited for use in resource-constrained environments. Its 25-byte state is actually smaller than AES-128, with its 16-byte data path and 16-byte round keys. On the downside, this small state limits the achievable security strength. We will address security strength levels 80 and 128 and compare their performance with Keccak$[r = 40, c = 160]$, the instance with capacity 160 bits based on the Keccak-$f[200]$ permutation.

A target security strength of $k = 80$ bits combined with $M \leq 2^{64}$ gives a capacity of $c = k + a = 144$ bits and a rate of 56 bits. This relatively small rate value severely limits degrees of freedom in differential attacks. As in the case of $\textsc{Keccup}\text{-}f[1600, n]$, our choice of the number of rounds is mostly inspired by possible weaknesses due to the limited algebraic degree. We think 9 rounds is on the safe side, so we propose $\textsc{Keccup}[r = 56, c = 144, n = 9]$. Compared to $\textsc{Keccak}[r = 40, c = 160]$, the speedup is $18/9 \times 56/40 \approx 2.8$.

A target security strength of $k = 128$ bits leads to a capacity of 192 bits and a rate of only 8 bits. Thanks to the very small value of this rate, attackers engaged in cube and similar attacks are expected to be forced applying multiple blocks. This relaxes the constraints on the number of rounds somewhat. On the other hand, clearly there must be some number of rounds between the injection of differences and the appearance of their effect at the output. Here we estimate that 6 rounds would be sufficient, so we propose $\textsc{Keccup}[r = 8, c = 192, n = 6]$. Still, the security strength of 128 bits comes at a high cost. Compared to $\textsc{Keccak}[r = 40, c = 160]$, the loss of speed is $18/6 \times 8/40 = 0.6$.

4 Variants

In this section we present two variants of the sponge and duplex constructions that do not comply to the standard definitions. In defining these variants we take the liberty of varying the rate and the number of rounds of the underlying permutation across the different phases. We are aware that variants and generalizations to the standard sponge definition have already been proposed. For example, in [20] it was observed that taking different rate/capacity pairs for the absorbing and squeezing phase can also have an effect on the security strength. Other variants are proposed as part of the Parazoa generalization of sponge functions [1], although not all are based on a permutation.

4.1 The donkeySponge construction

This mode is inspired by the Alred construction for MAC functions [16] and its instance Pelican-MAC [17,18]. The Alred construction allows building efficient MAC functions from block ciphers. Pelican-MAC is an instantiation based on AES, that is for long messages about 2.5 times faster than an AES-based CBC-MAC. Pelican-MAC takes as input a MAC key and a message and operates in three phases. In a first phase a secret state is initialized by applying AES to a 16-byte all-zero string with the MAC key as key. Then 16-byte message blocks are XORed into the secret state, interleaved by a permutation consisting of 4 unkeyed AES rounds. After applying all message blocks, the tag is obtained by applying AES to the secret state, again with the MAC key as key and (possibly) truncating the result. For the security of the Alred construction it is crucial that an adversary shall not be able to reconstruct the secret state or to generate inner collisions (pairs of partial messages leading to the same state).

Clearly the second phase resembles the absorbing phase of a sponge, but with rate equal to the width. In fact generalizing the Alred construction to support a b-bit permutation rather than a block cipher does not require much imagination. We did the exercise and call the result donkeySponge. We can summarize it as follows:

- The function takes as input a MAC key and a message and it returns a tag.
- The b-bit state is initialized with the MAC key and subject to n_{init} rounds of the permutation resulting in the secret state. The number of rounds n_{init} must be chosen such that all bits of the secret state depend on the MAC key.

– The b-bit blocks of the message are XORed into the secret state, interleaved with n_{absorb}-round permutations. The number n_{absorb} must be chosen to make the success probability of generating inner collisions negligible.
– The tag is obtained by applying a n_{squeeze}-round permutation to the secret state and truncating the result to ℓ bits. The number of rounds n_{squeeze} shall be high enough to prevent an adversary in reconstructing the inner state from outputs observed for chosen inputs. The number $b - \ell$ must be large enough to prevent state reconstruction by exhaustive search, namely, $b - \ell \geq k$.

Note that during the absorbing phase the rate becomes equal to the permutation width. This reduces the capacity to zero and precludes applying the proven generic security strength bound of the sponge construction. Alred has a provable reduction of retrieval of the MAC to the breaking of the block cipher. In donkeySponge the key can be readily computed from the secret state, so this is a security feature that donkeySponge does not have. On the other hand, Alred shares with donkeySponge that retrieval of the secret state is catastrophic for the security. In Alred, reconstruction of this state requires generating inner collisions or breaking the underlying block cipher. In donkeySponge, the former depends on the differential probability (DP) of differentials over n_{absorb} rounds. For the latter, breaking the underlying block cipher is replaced by successfully applying a chosen-input-difference attack on a truncated permutation with unknown input, whose success depends on the DP of differentials over n_{squeeze} rounds.

From an efficiency point of view, Alred requires working memory for the data path and the key schedule while donkeySponge only has a data path. For a given amount of working memory available, in donkeySponge message differences diffuse over the full memory, while in Alred the propagation of these differences is confined to the data path part. Hence in principle donkeySponge makes better use of available resources.

The donkeySponge construction is illustrated in Figure 3 and a formal specification is given in Algorithm 1. It has the following parameters:

– f: permutation family parameterized by the number of rounds, where the n-round member is denoted by $f[n]$;
– n_{init}: number of rounds applied after the key has been put in the state;
– n_{absorb}: number of rounds in between absorbing of message blocks;
– n_{squeeze}: number of rounds before squeezing of the tag (and in between squeeze operations if requested tag length exceeds rate);
– r: rate during squeezing.

We have done the exercise to see what parameter values would be reasonable for a KECCUP-based instances of this and the result is the following. For instances calling KECCUP-$f[1600, n]$ with security strength 128 and 256 and calling KECCUP-$f[200, n]$ with security strength 128 and 80, we would propose the following values:

– $n_{\text{init}} = 3$: after three rounds each state bit depends on some key bit, even for keys with very weak entropy.
– $n_{\text{absorb}} = 6$: from our differential propagation experiments [10,14] we believe that there are no 6-round trails with weight below 128 for KECCAK-$f[200]$ and below 256 for KECCAK-$f[1600]$.
– $n_{\text{squeeze}} = 12$: we take it as the double of $n_{\text{absorb}} = 6$ to accomodate for strength against variants such as higher-order differential attacks.

For KECCAK-$f[200]$ with target security strength 128 the length of the tag is limited to 72 bits. If a longer tag is desired, additional squeezing steps must be performed. For

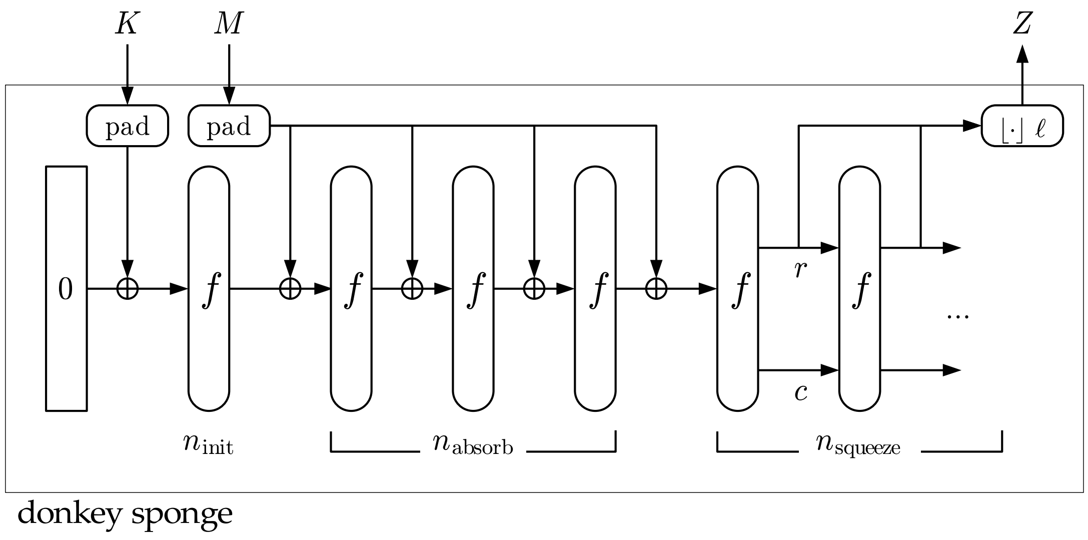

Fig. 3. The donkey sponge construction

long messages the Keccup-$f[1600, n]$-based variants are a factor $24/6 \times 1600/1024 = 6.25$ faster than Keccak[] and the Keccup-$f[200, n]$-based variants are a factor $18/6 \times 200/40 = 15$ faster than Keccak$[r = 40, c = 160]$. The fixed cost in the computation of a MAC is executing 15 rounds of Keccup-f.

4.2 The monkeyDuplex construction

The authenticated encryption with associated data (AEAD) mode SpongeWrap based on the duplex construction [7] only guarantees confidentiality if for the same key and different messages the associated data is unique. In other words, the associated data should behave as a nonce. Violating this results in the encryption of different plaintexts with the same keystream. However, it does not jeopardize the key.

In this section we propose a variant of the duplex construction called monkeyDuplex whose security requires the uniqueness of a nonce. This makes this mode more fragile and we are aware that nonce uniqueness may not be imposed in all applications. However, if it is possible, it results in a considerable security gain.

The principles underlying monkeyDuplex are the following:

– It is an object that upon creation is loaded with a MAC key and a nonce. Once created, one can make duplexing calls to it, providing it with an input string σ and requesting an output string Z. The output string depends on all previous input strings, the key and the nonce.
– Upon creation, the b-bit state is initialized with the concatenation of the key and the nonce and subject to n_{init} rounds. Informally speaking, the number of rounds n_{init} must be chosen such that an active attacker has no advantage in combining outputs of duplex objects loaded with different nonces. In other words, if the differential properties of $f[n_{\text{init}}]$ are strong enough, state values of monkeyDuplex objects with different nonce values can be considered independent. If so, state recovery by an attacker with access to multiple objects does not give an advantage over state recovery from a single object.

Algorithm 1 DONKEYSPONGE$[f, n_{\text{init}}, n_{\text{absorb}}, n_{\text{squeeze}}, r]$

```
Require: $r < b$

  Interface: $Z = \text{donkeySponge}(K, M, \ell)$ with $K$, $M$ and $Z$ bitstrings, $\ell$ an integer, $|K| < b$ and $|Z| = \ell$
  $s = K||\text{pad10}^*[b](|K|)$
  $s = f[n_{\text{init}}](s)$

  $P = M||\text{pad10}^*[b](|M|)$
  for $i = 0$ to $|P|_b - 2$ do
    $s = s \oplus P_i$
    $s = f[n_{\text{absorb}}](s)$
  end for
  $s = s \oplus P_{|P|_b - 1}$

  $Z$ = empty string
  while $|Z| < \ell$ do
    $s = f[n_{\text{squeeze}}](s)$
    $Z = Z||\lfloor s \rfloor_r$
  end while
  return $\lfloor T \rfloor_\ell$
```

– The duplexing calls are qualitatively the same as in the duplex construction. However, the number of rounds of the permutation, n_{duplex}, can be reduced significantly as compared to the plain duplex construction. We rely on the initialization phase and its nonce to make differential attacks infeasible: an attacker has no control whatsoever over state differences between pairs of monkeyDuplex objects. This limits his attack path to state reconstruction.

The monkeyDuplex construction is illustrated in Figure 4 and a formal specification is given in Algorithm 2. It has the following parameters:

– f: permutation family parameterized by the number of rounds, where the n-round member is denoted by $f[n]$;
– ℓ_{key}: length of the key;
– ℓ_{nonce}: length of the nonce;
– n_{init}: number of rounds applied after the key and nonce have been put in the state;
– n_{duplex}: number of rounds in a duplex call;
– r: rate during duplexing.

The difficulty of state recovery grows with increasing values of n_{duplex} and decreasing values of r, while the efficiency is determined by the ratio r/n_{duplex}. For achieving a given efficieny one can vary these two parameters where increasing the rate r necessitates increasing n_{duplex} and vice versa. This results in a spectrum of possible choices with at the two ends two particular approaches:

– Blockwise: $r \geq b/2$. the problem of state recovery consists in solving a CICO problem for the permutation $f[n_{\text{duplex}}]$. The choice of n_{duplex} is based on estimating the difficulty of solving this problem.
– Streaming: $n_{\text{duplex}} = 1$. The problem of state recovery consists in determining the state from the knowledge of a small part of the state for a number n_{unicity} of subsequent rounds. The difficulty of solving this problem depends on n_{unicity} and the nature of the round function. We have $n_{\text{unicity}} = \lfloor b/r \rfloor$. The value of r can hence be derived from the minimum value of n_{unicity} for which the state recovery problem is estimated to have expected workload above 2^k.

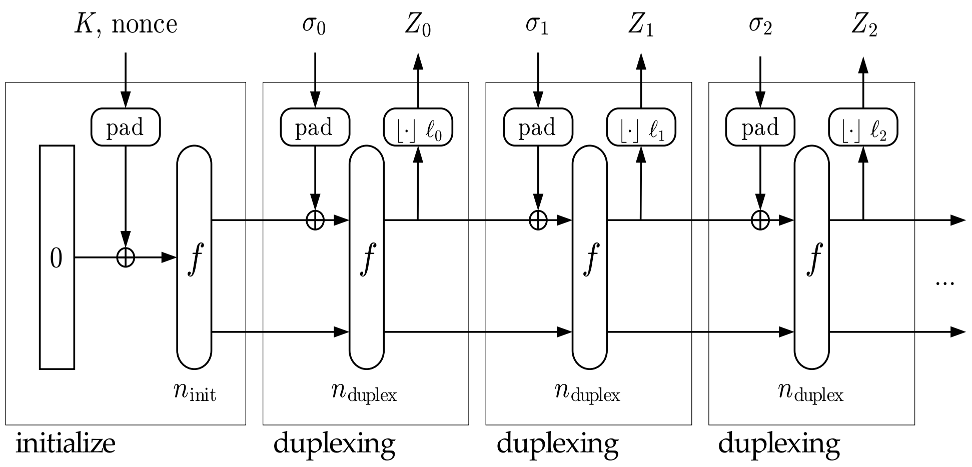

Fig. 4. The monkey duplex construction

Algorithm 2 MONKEYDUPLEX$[f, \ell_{\text{key}}, \ell_{\text{nonce}}, n_{\text{init}}, n_{\text{duplex}}, r]$

```
Require: $r < b$
Require: $\ell_{\text{key}} + \ell_{\text{nonce}} \leq b - 1$

  Interface: $D$.initialize($K$, nonce) with $|K| = \ell_{\text{key}}$ and $|\text{nonce}| = \ell_{\text{nonce}}$
  $s = K||\text{nonce}||\text{pad10}^*[b](\ell_{\text{key}} + \ell_{\text{nonce}})$
  $s = f[n_{\text{init}}](s)$

  Interface: $Z = D$.duplexing($\sigma, \ell$) with $Z$ and $\sigma$ bitstrings, integer $\ell$ satisfying $0 \leq \ell \leq r$ and $|\sigma| \leq r - 2$
  and $|Z| = \ell$
  $P = \sigma||\text{pad10}^*1[r](|\sigma|)||0^{b-r}$
  $s = s \oplus P$
  $s = f[n_{\text{duplex}}](s)$
  return $\lfloor s \rfloor_\ell$
```

We have done the exercise to see what parameter values would be reasonable for a KECCUP-based instances. We denote these instances by the name KETJE. The result is the following. For all instances we propose $n_{\text{init}} = 12$: this is motivated by the same arguments as the choice of n_{squeeze} in donkeySponge. In the choice of r and n_{duplex} we have blockwise and streaming instances.

For the streaming instances ($n_{\text{duplex}} = 1$):

– $b = 1600$, security strength 256: $r = 128$ yielding $n_{\text{unicity}} = 12$. This is a factor $24 \times 128/1024 = 3$ faster than KECCAK[].
– $b = 200$, security strength 80: $r = 16$ yielding $n_{\text{unicity}} = 12$. This is a factor $18 \times 16/40 = 7.2$ faster than KECCAK$[r = 40, c = 160]$.

We only propose a blockwise instance for $b = 1600$. A value of $b = 200$ does not allow taking $r > b/2$ and at the same time supporting a security strength of 80 and a multiplicity of 2^{64}. We have:

– $b = 1600$, security strength 256: $r = 1280$ and $n_{\text{duplex}} = 8$. This is a factor $24/8 \times 1280/1024 = 3.75$ faster than KECCAK[].

The monkeyDuplex construction can be used in different modes. The two most important ones are:

– keystream generation, where all duplexing calls are blank, i.e. with $\sigma =$ empty string;
– authenticated encryption, similar to SpongeWrap [7].

5 Conclusions

We have discussed ways to speed up keyed modes based of on permutations. These are based on the following observations:

– Constructions can reach a higher security strength level with respect to generic attack in keyed modes than in unkeyed modes.
– Concrete primitives reach a higher security strength against attacks in keyed used cases than in un-keyed use cases.

We have proposed two new constructions that provide a speed advantage over the standard sponge and duplex constructions at the expense of a reduction of generality. The donkeySponge construction is basically a keyed sponge dedicated to MAC computation. The monkeyDuplex construction is a keyed duplex construction that depends on nonces for its cryptographic security and can be used for efficient keystream generation and authenticated encryption. We have illustrated the potential of our proposals with instances based on reduced-round versions of Keccak-f[1600] and Keccak-f[200], but they can be applied to any iterated permutation.

References

1. E. Andreeva, B. Mennink, and B. Preneel, *The parazoa family: generalizing the sponge hash functions*, Int. J. Inf. Sec. **11** (2012), no. 3, 149–165.
2. J.-P. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia, *Quark: A lightweight hash*, in Mangard and Standaert [21], pp. 1–15.
3. D. Bernstein, *The Salsa20 family of stream ciphers*, The eSTREAM Finalists (M. Robshaw and O. Billet, eds.), Lecture Notes in Computer Science, vol. 4986, Springer, 2008, pp. 84–97.
4. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, *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. ———, *Sponge-based pseudo-random number generators*, in Mangard and Standaert [21], pp. 33–47.
6. ———, *Cryptographic sponge functions*, January 2011, `http://sponge.noekeon.org/`.
7. ———, *Duplexing the sponge: single-pass authenticated encryption and other applications*, Selected Areas in Cryptography (SAC), 2011.
8. ———, *On the security of the keyed sponge construction*, Symmetric Key Encryption Workshop (SKEW), February 2011.
9. ———, Keccak *crunchy crypto collision and pre-image contest*, 2011, `http://keccak.noekeon.org/crunchy_contest.html`.
10. ———, *The* Keccak *reference*, January 2011, `http://keccak.noekeon.org/`.
11. ———, *Reference and optimized implementations of* Keccak, 2012, `http://keccak.noekeon.org/`.
12. A. Bogdanov, M. Knezevic, G. Leander, D. Toz, K. Varıcı, and I. Verbauwhede, *SPONGENT: A lightweight hash function*, CHES (B. Preneel and T. Takagi, eds.), Lecture Notes in Computer Science, vol. 6917, Springer, 2011, pp. 312–325.
13. J. Daemen and G. Van Assche, *Producing collisions for PANAMA, instantaneously*, Fast Software Encryption 2007 (A. Biryukov, ed.), LNCS, Springer-Verlag, 2007, pp. 1–18.
14. ———, *Differential propagation analysis of* Keccak, Fast Software Encryption 2012, 2012.
15. J. Daemen and C. S. K. Clapp, *Fast hashing and stream encryption with PANAMA*, Fast Software Encryption 1998 (S. Vaudenay, ed.), LNCS, no. 1372, Springer-Verlag, 1998, pp. 60–74.
16. J. Daemen and V. Rijmen, *A new MAC construction ALRED and a specific instance ALPHA-MAC*, Fast Software Encryption (H. Gilbert and H. Handschuh, eds.), Lecture Notes in Computer Science, vol. 3557, Springer, 2005, pp. 1–17.
17. ———, *The Pelican MAC function*, IACR Cryptology ePrint Archive **2005** (2005), 8.
18. ———, *Refinements of the ALRED construction and MAC security claims*, IET information security **4** (2010), 149–157.

19. P. Gauravaram, L. R. Knudsen, K. Matusiewicz, F. Mendel, C. Rechberger, M. Schläffer, and S. S. Thomsen, *Grøstl – a SHA-3 candidate*, Submission to NIST (round 3), 2011.
20. J. Guo, T. Peyrin, and A. Poschmann, *The PHOTON family of lightweight hash functions*, Crypto (P. Rogaway and R. Safavi-Naini, eds.), Lecture Notes in Computer Science, vol. 6841, Springer, 2011, pp. 222–239.
21. 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.
22. NIST, *NIST special publication 800-57, recommendation for key management (revised)*, March 2007.
23. V. Rijmen, B. Van Rompay, B. Preneel, and J. Vandewalle, *Producing collisions for PANAMA*, Fast Software Encryption 2001 (M. Matsui, ed.), LNCS, no. 2355, Springer-Verlag, 2002, pp. 37–51.
24. R. Rivest, *The MD5 message-digest algorithm*, Internet Request for Comments, RFC 1321, April 1992.
25. Y. Sasaki and K. Aoki, *Finding preimages in full MD5 faster than exhaustive search*, Advances in Cryptology – Eurocrypt 2009 (A. Joux, ed.), Lecture Notes in Computer Science, vol. 5479, Springer, 2009, pp. 134–152.
26. M. Stevens, A. Sotirov, J. Appelbaum, A. Lenstra, D. Molnar, D. Osvik, and B. de Weger, *Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate*, Crypto (S. Halevi, ed.), Lecture Notes in Computer Science, vol. 5677, Springer, 2009, pp. 55–69.
27. H. Wu, *The hash function JH*, Submission to NIST (round 3), 2011.