Kirby: A Robust Permutation-Based PRF Construction

Charlotte Lefevre[1,2], Yanis Belkheyar[1] and Joan Daemen[1]

[1] Digital Security Group, Radboud University, Nijmegen, The Netherlands
firstname.lastname@ru.nl
[2] CSEM, Switzerland

Abstract. We present a construction, called Kirby, for building a variable-input-length pseudorandom function (VIL-PRF) from a b-bit permutation. For this construction we prove a tight bound of $b/2$ bits of security on the PRF distinguishing advantage in the random permutation model and in the multi-user setting. Similar to full-state keyed sponge/duplex, it supports full-state absorbing and additionally supports full-state squeezing, where the latter can at most squeeze $b - c$ bits per permutation call for a security level of c bits. This advantage is especially relevant on constrained platforms when using a permutation with small width b. For instance, for $b = 256$ at equal security strength the squeezing rate of Kirby is twice that of keyed sponge/duplex. We define a simple mode on top of Kirby that turns it into a deck function with parallel expansion. This deck function is suited for lightweight applications in the sense that it has a low memory footprint. Moreover, for short inputs it can be used for low-latency stream encryption: the time between the availability of the input and the keystream is only a single permutation call. Another feature that sets Kirby apart from other constructions is that leakage of an intermediate state does not allow recovering the key or *earlier states.*

Keywords: permutation-based cryptography · provable security · multi-user security · PRF · lightweight · deck function

1 Introduction

Permutation-based cryptography has become increasingly popular in the last years. Most of the proposed schemes make use of the sponge [BDPV07], duplex [BDPA11] and monkeyduplex constructions [BDPVA12] that are all serial in nature. Particularly noteworthy, the winner of the NIST SHA-3 competition, the XOF family Keccak [BDPV11] and the recent winner of the NIST lightweight cryptography competition, the authenticated encryption scheme Ascon [DEMS21], rely on the sponge and a variant of monkeyduplex [BDPVA12], respectively. A permutation-based construction that allows more parallelism is Farfalle and was proposed in [BDP+16] and instantiated with the Xoodoo permutation in [DHAK18]. It results in a so-called doubly extendable cryptographic keyed (deck) function that has both variable-length input and output.

In this paper, we introduce a novel permutation-based variable-input-length pseudorandom function (VIL-PRF) construction called Kirby. This construction is suitable to build efficient lightweight cryptographic primitives with low memory footprint and/or low latency. We designed our construction to provide a level of generic multi-user security that is optimal in the width of the permutation to have resilience under leakage of inner states.

Kirby takes an input of variable length, allows full-state absorption, and generates outputs of length equal to the underlying permutation width. We specify a mode that combines Kirby with injective prefix-free input encoding and including in that input a counter, resulting in a deck function.

Prior work and our contribution. Kirby is inspired by the sponge [BDPV07] and duplex [BDPA11] constructions and can be also seen as a generalization of the construction used in Salsa [Ber08]. Over sponge and duplex it has the advantage that it can do full-state squeezing and over Salsa it has the advantage that it has arbitrary-length input. The full-state squeezing is possible thanks to continuous ratcheting as in the Davies-Meyer construction employed in Merkle-Damgård hashing [Mer89, Dam89, PGV93]. A side effect of this is that leakage of internal state does not allow the recovery of the key or earlier internal states.

The squeezing phase of the Kirby deck mode is similar to that in farfalle [BDP$^+$16]. Another design that shares elements with Kirby is Muffler [BBN22]. It is a sponge-based PRF that makes use of identifier to optimize the multi-user security bound of that construction. In the same vein [DMA17] considered the multi-target security of the keyed duplex construction in the presence of a global nonce and later this was generalized in [DM23] by treating different restrictions on the initialization of the keyed duplex.

Similarly to the proof in [DMA17] for the full-state keyed duplex construction, Kirby handles arbitrary key distributions, that are characterized solely by their min-entropy and their collision probability, as formally defined in Section 2. In particular, this shows that Kirby is resilient against related-key attacks.

Summarizing, our contributions are:

- A new permutation-based VIL-PRF construction Kirby and deck mode on top of that supports full-state absorbing and squeezing; allowing a low memory footprint and low-latency operation. Moreover, it has built-in ratcheting providing a degree of leakage resilience and static identifiers allowing optimal multi-target security;

- A proof of a tight bound on the PRF distinguishing advantage in the random permutation model and in the multi-user setting supporting arbitrary key distributions.

Organization of the paper. We introduce preliminaries in Section 2. We specify the construction Kirby in Section 3 together with rationales. Section 4 is dedicated to the security proof of the construction. Finally, in Section 5, we specify the Kirby deck mode.

2 Preliminaries

Notation. We introduce here useful notation. Let $a, b \in \mathbb{N}$ such that $a \leq b$. We use $[\![a, b]\!]$ to denote the set $\{a, a+1, \ldots, b\}$. Let $d \in \mathbb{N}$. We use the letter b to denote the width of a permutation and κ for the length of the keys. The set $\{0,1\}^d$ denotes the strings of length d bits, and $\{0,1\}^*$ denotes $\bigcup_{a \in \mathbb{N}} \{0,1\}^a \cup \epsilon$. Moreover, $\{0,1\}^{d+}$ denotes the strings with length a multiple of d, i.e., $\{0,1\}^{d+} = \bigcup_{a \in \mathbb{N}} \{0,1\}^{ad}$. Let $s, s' \in \{0,1\}^*$. We denote the length of s as $|s|$. The empty string is represented by ϵ. We write $s||s'$ for the concatenation of s and s'. For two string s_1 and s_2 of equal length, we use $s_1 + s_2$ to denote the bitwise addition of s_1 and s_2. Given $a, b \in \mathbb{N}$ such that $a < 2^b$, $\langle a \rangle_b$ denotes an injective encoding of a over b bits.

Given a finite set S, $x \xleftarrow{\$} S$ means that x is the result of a uniform random sampling in S. Similarly, if $\mathcal{D}$ represents a distribution, $x \xleftarrow{\$} \mathcal{D}$ signifies that x is generated using the distribution $\mathcal{D}$. The set of permutations over b bits is denoted $\mathbf{Perms}(b)$. For $n, k \in \mathbb{N}$

such that $n \geq k$, $(n)_k$ denotes the falling factorial of n of degree k, i.e.,

$$(n)_k = \prod_{i=0}^{k-1} (n-i)\,.$$

A string $E \in \{0,1\}^*$ is a prefix of $E' \in \{0,1\}^*$, denoted by $E \prec E'$ if E' truncated to its first $|E|$ bits is equal to E. Moreover, a set $\mathcal{E} \subset \{0,1\}^*$ is prefix-free if for any $E, E' \in \mathcal{E}$ distinct, E is not a prefix of E'.

Key sampling. Similarly to the work of Daemen et al. [DMA17], we assume that the keys of the μ users $\mathrm{K}_1, \dots, \mathrm{K}_\mu \in \{0,1\}^\kappa$ are generated using an arbitrary distribution $\mathcal{D}_{\text{key}}$. This distribution is characterized by two essential parameters: the min-entropy and the maximum collision probability. The min-entropy of a distribution $\mathcal{D}_{\text{key}}$ is defined by

$$\mathcal{H}_{\min}(\mathcal{D}_{\text{key}}) = \max_{m \in [\![1,\mu]\!], x \in \{0,1\}^\kappa} -\log_2 \left(\mathbf{Pr} \left(\mathrm{K}_1, \dots, \mathrm{K}_\mu \xleftarrow{\$} \mathcal{D}_{\text{key}} : \mathrm{K}_m = x \right) \right)\,.$$

The maximum collision probability is defined by

$$\mathcal{H}_{\text{col}}(\mathcal{D}_{\text{key}}) = \max_{m, m' \in [\![1,\mu]\!], m \neq m'} -\log_2 \left(\mathbf{Pr} \left(\mathrm{K}_1, \dots, \mathrm{K}_\mu \xleftarrow{\$} \mathcal{D}_{\text{key}} : \mathrm{K}_m = \mathrm{K}_{m'} \right) \right)\,.$$

In the case where $\mathcal{D}_{\text{key}}$ denotes uniform sampling, we have $\mathcal{H}_{\min}(\mathcal{D}_{\text{key}}) = \mathcal{H}_{\text{col}}(\mathcal{D}_{\text{key}}) = \kappa$.

Indistinguishability. Let W_0, W_1 be two worlds, and consider a distinguisher $\mathcal{A}$ placed in world W_a, for $a \xleftarrow{\$} \{0,1\}$, that we denote by $\mathcal{A}^{W_a}$. Without loss of generality, we assume that $\mathcal{A}$ is a deterministic algorithm that never makes queries for which it already knows the answer.

The advantage of $\mathcal{A}$ is defined as follows:

$$\mathrm{Adv}(W_0, W_1)(\mathcal{A}) = \left| \mathbf{Pr}\left(\mathcal{A}^{W_0} = 1\right) - \mathbf{Pr}\left(\mathcal{A}^{W_1} = 1\right) \right|\,.$$

H-coefficient technique. In the proof, we will use the H-coefficient technique [Pat08,CS14]. Consider two worlds W_0 and W_1. We summarize the interaction between $\mathcal{A}$ and the world in a transcript, which contains tuples of queries-responses. Denote by τ_0 (resp., τ_1) the probability distribution of the transcript in W_0 (resp., W_1). Consider a partition of all the possible transcripts as $T = T_{\text{GOOD}} \cup T_{\text{BAD}}$. If there exists $\epsilon_1, \epsilon_2 > 0$ such that

$$\forall \tau \in T_{\text{GOOD}},\ \frac{\mathbf{Pr}(\tau_1 = \tau)}{\mathbf{Pr}(\tau_0 = \tau)} \geq 1 - \epsilon_1\,,$$
$$\text{and}\ \ \mathbf{Pr}(\tau_0 \in T_{\text{BAD}}) \leq \epsilon_2\,,$$

then

$$\mathrm{Adv}(W_0, W_1)(\mathcal{A}) \leq \epsilon_1 + \epsilon_2\,.$$

3 Specification and Design Rationale

In this section, we describe the Kirby construction specification with a simple algorithm and deliver some design rationales to link the different elements of the construction with the associated features.

3.1 Kirby Specification

Kirby operates on a b-bit state S and for its iterations it makes use of a transformation F consisting of the permutation P with a feedforward: $\mathsf{F}(S) = P(S) + S$. First, it initializes the state S with the concatenation of a κ-bit key K and a $(b - \kappa)$-bit key identifier id. Then it sequentially absorbs the blocks of E by for each block adding it to the state and then applying F to the result. After all blocks of E have been absorbed, it returns the state S as output Z.

It is instantiated by choosing the key length κ and a permutation P. We define formally the Kirby construction in Algorithm 1. We show in Figure 1 a schematic of the Kirby construction with a 3-block input.

Algorithm 1 Definition of construction Kirby$[P, \kappa]$.

```
Require: $P$ is a $b$-bit permutation
  Input: bit strings key K, identifier id, input block sequence $E = (E_1, \dots, E_{|E|})$
  Output: a $b$-bit string $Z$
  $S \leftarrow \mathrm{K}||\mathrm{id}$
  $S \leftarrow S + P(S)$
  for $i = 1$ to $|E|$ do
      $S \leftarrow S + E_i$
      $S \leftarrow S + P(S)$
  end for
  return $Z \leftarrow S$
```

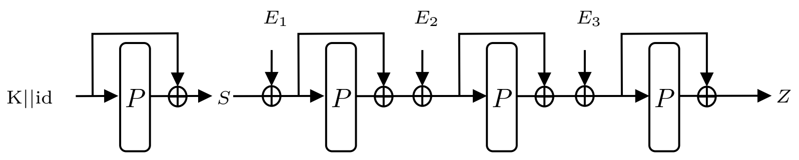

Figure 1: Example of iteration of Algorithm 1, for an input sequence E with $|E| = 3$.

3.2 Design Rationale

The main objective of this design is to provide a simple algorithm for building a VIL-PRF from a permutation that is versatile, with good provable security, and that allows for compact implementation.

For those two aspects, the overall design requires very few operations on top of the permutation and only two time the width of the permutation as memory storage for round-based implementations.

The critical path of the construction for single-block inputs E is the permutation, the input block addition and a b-bit padded string and the feedforward addition The initialization step taking the key and identifier can be pre-computed and hence does not impact the latency.

The design of Kirby also takes into account side-channel leakage considerations. First, the fact that the key and fixed-value identifier id is only used during the initialization phase reduces the attack surface. The relative similarity of our construction to the use-case study in [CLMP21] makes us think that it could be possible to prove that the leakage of intermediate state does not compromise any previously computed state.

For the provable security aspect, the main elements of our design are the identifier, the key length and the requirement that the set of input block sequences must be prefix-free.

The same identifier can be used for multiple users, with a slight degradation on the security that can be compensated by increasing the key length. We elaborate more on the impact of using the same identifier too many times in Section 4.2. We show in our security proof the importance of the requirement that the set of input sequences is prefix-free. This element plays a major role in the security to avoid collision, and length extension attacks.

Finally, we show in Section 4.2 how the combination of all those elements result in a tight security bound.

4 Security Analysis of Kirby

In this section, we provide a security proof of Kirby in the ideal permutation model similar to the proof of [DMA17]. In detail, Section 4.1, we introduce useful definitions as well as the security model. Section 4.2 is dedicated to the security bound and the proof. Finally, in Section 4.3 we discuss the tightness of the security bound.

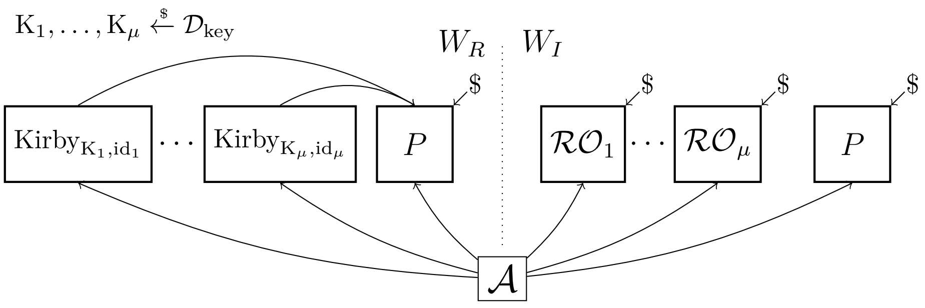
$\mathrm{K}_1, \ldots, \mathrm{K}_\mu \xleftarrow{\$} \mathcal{D}_{\text{key}}$
W_R
W_I
$\$$
$\mathrm{Kirby}_{\mathrm{K}_1,\mathrm{id}_1}$
$\cdots$
$\mathrm{Kirby}_{\mathrm{K}_\mu,\mathrm{id}_\mu}$
P
$\mathcal{RO}_1$
$\mathcal{RO}_\mu$
P
$\mathcal{A}$

Figure 2: Illustration of the security game. The symbol \$ on top of an oracle means that the underlying primitive is sampled uniformly at random, and $\mathrm{Kirby}_{\mathrm{K},\mathrm{id}}$ denotes the construction Kirby initialized with the key K and the identifier id.

4.1 Security Model

We will prove the security of Kirby in the multi-user scenario. Namely, the construction is assumed to be used simultaneously by μ users, split as $\mu = \mu_1 + \cdots + \mu_s$, where μ_i users share the same identifier. We represent this quantity by a vector $\boldsymbol{\mu} := (\mu_1, \ldots, \mu_s)$. The set of queried strings is denoted by $\mathbb{E}$, and contains elements of form (m, E), where the block sequence $E \in \{0,1\}^{b+} \cup \epsilon$ has been queried to the oracle number m. We abuse notation, and say that $\mathbb{E}$ is prefix-free whenever the following set is prefix-free:

$$\left\{ \langle m \rangle_{\lceil \log_2(\mu) \rceil} || E \;\middle|\; (m, E) \in \mathbb{E} \right\} .$$

Our goal is to prove an upper bound on the advantage of any adversary to distinguish these μ instances of Kirby based on a random permutation from μ independent random oracles, under the assumption that the set of queried strings $\mathbb{E}$ is prefix-free. We use $\mathtt{IK}[m]$ as an abbreviation for $\mathrm{K}_m, ||\mathrm{id}_m$.

Specification of the worlds. The security model is a special type of distinguishing game, akin to multi-user PRF. In the following, we specify how the two worlds to distinguish W_R and W_I are instantiated in our particular setting. In both worlds, the adversary $\mathcal{A}$ has access to a single primitive oracle $\mathcal{O}_P$ and μ construction oracles denoted by $\mathcal{O}_{C_1}, \ldots, \mathcal{O}_{C_\mu}$.

Let $1 \leq s < 2^{b-\kappa}$, and $x_1 \ldots, x_s \in \{0,1\}^{b-\kappa}$ be pairwise distinct. Let $\texttt{Expand}\boldsymbol{\mu} \in (\llbracket 1, s \rrbracket)^{\mu}$ denote the array generated by repeating in order each element $t \in \llbracket 1, s \rrbracket$ a number of times equal to the value of the t^{th} element of $\boldsymbol{\mu}$. Define the identifier of user m to be $x_{\texttt{Expand}\boldsymbol{\mu}[m]}$.

For $m \in \llbracket 1, \mu \rrbracket$, the oracle $\mathcal{O}_{C_m}$ takes as input an element $E \in \{0,1\}^{b+}$. $\mathcal{O}_P$ takes as input a string in $\{0,1\}^b$, and a bit that denotes the direction of the query. Each of the worlds is specified as follows:

- In the real world W_R, $P \xleftarrow{\$} \mathbf{Perms}(b)$, $\mathrm{K}_1, \ldots, \mathrm{K}_\mu \xleftarrow{\$} \mathcal{D}_{\text{key}}$, $\mathcal{O}_{C_m}$ gives access to Kirby construction based on the permutation P, the key K_m, and the identifier id_m, while $\mathcal{O}_P$ gives access to P;

- In the ideal world W_I, $P \xleftarrow{\$} \mathbf{Perms}(b)$, $\mathcal{O}_{C_m}$ gives access to a random oracle $\mathcal{RO}_m$, and $\mathcal{O}_P$ gives access to P. We stress that the random oracles $\mathcal{RO}_1, \ldots, \mathcal{RO}_\mu$ are independent.

Figure 2 illustrates the security game.

Metrics for queries. In this paragraph, we specify how the queries of the distinguisher to the oracles are measured. Indeed, the primitive and construction queries are counted separately, and in the real world one construction query has a practical cost which depends on the length of the input block string E. More precisely, if E has ℓ blocks of b bits, the associated construction query has a cost of $\ell + 1$ (the extra call comes from the initialization phase). Additionally, if a second construction query is made to the same oracle with input $E' \in \{0,1\}^{b\ell'}$ that has a common prefix of x blocks with E, then this construction query has cost $\ell' - x$. We define M to be the total cost in terms of minimal permutation evaluations that are required by construction queries in W_R. In the example above, $M = \ell + \ell' - x$. To accurately capture this cost in the proof, we quantify the adversarial resources as follows:

- N is the number of primitive queries;
- q is the number of construction queries;[1]
- M is the total number of permutation evaluations that would be required in the real world from construction queries without considering duplicate queries evaluations from repeated prefixes;

We now have all ingredients to define the security model. Let $\mathcal{D}_{\text{key}}$ a distribution for the key sampling procedure. Consider $1 \leq s < 2^{b-\kappa}$, and $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_s) \in \mathbb{N}^s$ be such that $\sum_{i=1}^{s} \mu_i = \mu$. We define $\mathrm{Adv}^{\mathrm{PRF}}_{\mathrm{PF\text{–}Kirby}}(N, M, \boldsymbol{\mu}, \mathcal{D}_{\text{key}})$ as the maximum advantage $\mathrm{Adv}(W_R, W_I)(\mathcal{A})$ over all distinguishers $\mathcal{A}$ such that:

- $\mathcal{A}$ has access to μ construction oracles in both the real world and the ideal world;

- In W_R, keys are sampled according to $\mathcal{D}_{\text{key}}$, identifiers are partitioned according to $\boldsymbol{\mu}$;

- The construction queries made by $\mathcal{A}$ are prefix-free set ;

- The construction queries would require in total M permutation queries in W_R;

- $\mathcal{A}$ can make and at most N primitive queries.

[1]The quantity q does not appear in the security bound, yet this quantity is useful to refer to throughout the proof.

4.2 Security Bound of Kirby

Theorem 1. *Let $\boldsymbol{\mu} = (\mu_1, \dots, \mu_s) \in \mathbb{N}^s$, and $N, M \in \mathbb{N}$ such that $NM \leq 2^{b-1}$. We have*

$$\mathrm{Adv}^{PRF}_{\text{PF–Kirby}}(N, M, \boldsymbol{\mu}, \mathcal{D}_{key}) \leq \frac{M(M-1)}{2^b} + \frac{2NM}{2^b} + \frac{\sum_{i=1}^{s} \mu_i(\mu_i - 1)}{2 \times 2^{\mathcal{H}_{col}(\mathcal{D}_{key})}} + \frac{N \max_i \mu_i}{2^{\mathcal{H}_{min}(\mathcal{D}_{key})}} .$$

Interpretation of the bound. This security bound captures a versatile number of use-cases regarding the identifiers. In particular, when all identifiers are distinct, this translates to $\boldsymbol{\mu} = (1, \dots, 1)$, and the bound simplifies to

$$\mathrm{Adv}^{\mathrm{PRF}}_{\text{PF–Kirby}}(N, M, \boldsymbol{\mu}, \mathcal{D}_{\mathrm{key}}) \leq \frac{M(M-1)}{2^b} + \frac{2NM}{2^b} + \frac{N}{2^{\mathcal{H}_{\min}(\mathcal{D}_{\mathrm{key}})}} . \tag{1}$$

Taking $\mathcal{D}_{\mathrm{key}}$ to be uniform sampling, and assuming that each user has an online complexity limited by 2^{64}, this translates to $M \leq \mu 2^{64}$, and we obtain

$$\mathrm{Adv}^{\mathrm{PRF}}_{\text{PF–Kirby}}(\mathcal{A}) \leq \frac{\mu^2}{2^{b-128}} + \frac{2\mu N}{2^{b-64}} + \frac{N}{2^{\kappa}} .$$

One can reasonably assume that an adversary $\mathcal{A}$ has a computational power limited to $N \lll 2^{128}$. Therefore, a key length satisfying $\kappa \geq 128$ allow the rightmost term to be negligible. Taking a small permutation width such as $b = 256$, the distinguishing advantage remains negligible as long as the number of users stays way below 2^{64} (note that given 2^{64} users, κ cannot be larger than 192 due to the identifiers being encoded over $256 - \kappa$ bits).

On the other hand, when no identifier is used, this means that $\boldsymbol{\mu} = (\mu)$, and the bound becomes

$$\mathrm{Adv}^{\mathrm{PRF}}_{\text{PF–Kirby}}(N, M, \boldsymbol{\mu}, \mathcal{D}_{\mathrm{key}}) \leq \frac{M(M-1)}{2^b} + \frac{2NM}{2^b} + \frac{\mu(\mu-1)}{2 \times 2^{\mathcal{H}_{\mathrm{col}}(\mathcal{D}_{\mathrm{key}})}} + \frac{N\mu}{2^{\mathcal{H}_{\min}(\mathcal{D}_{\mathrm{key}})}} .$$

Compared to (1), there is a security degradation in the key length compared to the case where all identifiers are distinct. More precisely, targeting the same security strength, the key length should be increased by $\log(\mu)$, and the key length should be larger than $2\log(\mu)$. For example, with the same assumptions as previously, if aiming for an equivalent security strength (ie., 128 bits), the key length should be increased to at least 192.

In Section 4.3 we discuss the tightness of the bound. The remainder of this section is dedicated to the security proof.

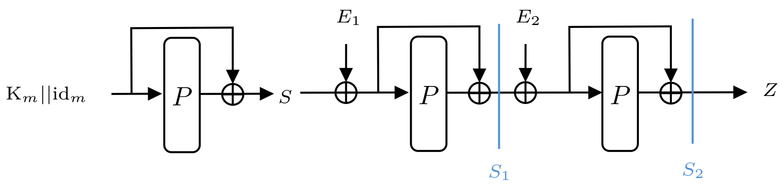

Figure 3: Illustration of the intermediate states in the real world. For this particular example, the tuples (m, S), $(m, (E_1), S_1)$, $(m, (E_1, E_2), S_2)$ belong to the extended transcript T.

Transcript notation. In the following, we define the transcript induced by the interaction between the distinguisher and the oracles. The transcript is an ordered list with $N + q$ elements. Each primitive query results in the addition of a tuple (X, Y, d) to the transcript, where $d \in \{fwd, inv\}$ denotes the direction of the query, and $Y = P(X)$. Similarly, each construction query appends to the list a tuple $(m, \mathrm{id}_m, \mathrm{path} = E, Z)$, where $1 \leq m \leq \mu$ refers to the oracle called, id_m is the identifier used, the path E is the input block sequence absorbed, and Z is the output of the construction oracle.

For the sake of the proof, we allow the oracles to release additional information at the end of the interaction, right before the distinguisher outputs its decision bit. More precisely, in the extended transcript that we call T, the elements associated to primitive queries are kept unchanged. Moreover, one construction query $(m, \mathrm{id}_m, \mathrm{path} = E, Z)$ is split into following $|E| + 1$ transcript elements:

- A construction initialization element: $(m, \mathrm{path} = \epsilon, S)$
- $|E|$ different construction absorb elements. For instance if $|E| = 4$ we have
  - $(m, \mathrm{path} = (E_1), S)$
  - $(m, \mathrm{path} = (E_1, E_2), S)$
  - $(m, \mathrm{path} = (E_1, E_2, E_3), S)$
  - $(m, \mathrm{path} = (E_1, E_2, E_3, E_4), S)$

We call a path E final if the construction has presented an output Z for it. A path which is not final is called intermediate.

Given $(m, \mathrm{path} = E, S) \in T$, the sampling method for S varies depending on the adversary's world:

- In the real world, S is the state after having absorbed the path E with the key K_m. In particular, if E is intermediate, then S is called an intermediate state. Otherwise, if E is a final path, S is the output of the construction Z.
- In the ideal world, S equals $\mathcal{RO}_m(E)$.

To simplify the notation, when the path E equals ϵ, we omit it.

There maybe duplicates among the construction absorb elements and these are removed from the transcript. There are at most μ construction initialization elements.

The construction absorb elements in the transcript can be arranged in a graph and form a forest. Each construction init element is a root of a tree and its nodes are the construction absorb element where the path is the sequence of edges one has to follow to get to the root (in reverse order). Quite naturally, the blocks of the path E form the labels of the edges. For example node $[m, \mathrm{path} = (E_1, E_2, E_3)]$ is the parent of node $[m, \mathrm{path} = (E_1, E_2, E_3, E_4)]$. The nodes are labeled with the state S and we denote the state of a node reached by following the path E in tree m by $S[m, E]$. We denote the label of the parent of the node in position $[m, E]$ by $[m, \mathrm{par}(E)]$ and the last block of a path E by E_{last}. The minimum number of blocks that must be presented as input to the construction is the number of edges in the graph. However, we also consider the identifiers loaded in the init operation as input and therefore, M is the total number of nodes in the graph.

One important remark is that every transcript that can be produced in the real world is also reachable in the ideal world. However, the converse is not true as the ideal world can produce intermediate states that do not conform to Kirby. Indeed, in the ideal world the primitive queries and construction queries are independent, and the intermediate states are generated randomly and uniformly, so that they might be incompatible with the bijectivity of a permutation. These transcripts are referred to as *permutation-inconsistent.*

Bad transcripts definition. We define here two bad events called respectively **COL** and **GUESS**, each split into sub-events. To facilitate notation, in the following, we implicitly assume the existence of the nodes listed at the beginning of each bad event.

$$\begin{aligned}
\textbf{COLkey} &: [m] \neq [m'] \text{ with } \texttt{IK}[m] = \texttt{IK}[m'] \,, \\
\textbf{COLfwd}1 &: [m,E] \neq [m',E'] \text{ with } S[m,\mathrm{par}(E)] \oplus E_{\mathrm{last}} = S[m',\mathrm{par}(E')] \oplus E'_{\mathrm{last}} \,, \\
\textbf{COLfwd}2 &: [m,E],[m'] \text{ with } \texttt{IK}[m'] = S[m,\mathrm{par}(E)] \oplus E_{\mathrm{last}} \,, \\
\textbf{COLinv}1 &: [m,E] \neq [m',E'] \text{ with } S[m,E] \oplus S[m,\mathrm{par}(E)] \oplus E_{\mathrm{last}} = S[m',E'] \oplus S[m',\mathrm{par}(E')] \oplus E'_{\mathrm{last}} \,, \\
\textbf{COLinv}2 &: [m,E],[m'] \text{ with } \texttt{IK}[m'] \oplus S[m'] = S[m,E] \oplus S[m,\mathrm{par}(E)] \oplus E_{\mathrm{last}} \,, \\
\textbf{COLinv}3 &: [m] \neq [m'] \text{ with } \texttt{IK}[m] \oplus S[m] = \texttt{IK}[m'] \oplus S[m'] \,,
\end{aligned}$$

$$\begin{aligned}
\textbf{COLfwd} &: \textbf{COLfwd}1 \vee \textbf{COLfwd}2 \,, \\
\textbf{COLinv} &: \textbf{COLinv}1 \vee \textbf{COLinv}2 \vee \textbf{COLinv}3 \,, \\
\textbf{COL} &: \textbf{COLkey} \vee \textbf{COLfwd} \vee \textbf{COLinv} \,,
\end{aligned}$$

$$\begin{aligned}
\textbf{GUESSkey} &: [m],(X,Y,\mathit{fwd}) \text{ with } X = \texttt{IK}[m] \,, \\
\textbf{GUESSfwd} &: [m,E],(X,Y,\mathit{fwd}) \text{ with } X = S[m,\mathrm{par}(E)] \oplus E_{\mathrm{last}} \,, \\
\textbf{GUESSinv}1 &: [m,E],(X,Y,\mathit{inv}) \text{ with } Y = S[m,E] \oplus S[m,\mathrm{par}(E)] \oplus E_{\mathrm{last}} \,, \\
\textbf{GUESSinv}2 &: [m],(X,Y,\mathit{inv}) \text{ with } Y = \texttt{IK}[m] \oplus S[m] \,,
\end{aligned}$$

$$\begin{aligned}
\textbf{GUESSinv} &: \textbf{GUESSinv}1 \vee \textbf{GUESSinv}2 \,, \\
\textbf{GUESS} &: \textbf{GUESSkey} \vee \textbf{GUESSfwd} \vee \textbf{GUESSinv} \,.
\end{aligned}$$

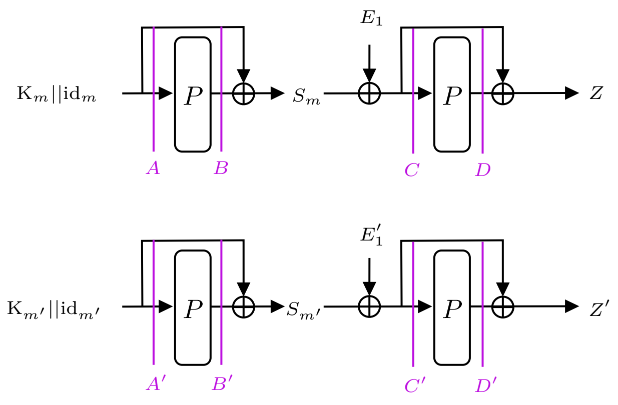

Figure 4: Illustration of the bad events in the real world. **COLkey** corresponds to $A = A'$, **COLfwd** to $C = C'$, $C = A'$, or $C = A$ (not all cases listed), **COLinv** to $D = D'$, $D = B$, $D = B'$, or $B = B'$ (not all cases listed). **GUESSkey** corresponds to (A,B,fwd) or $(A',B',\mathit{fwd}) \in T$. **GUESSfwd** corresponds to (C,D,fwd) or $(C',D',\mathit{fwd}) \in T$. Finally, **GUESSinv** is set whenever either (A,B,inv), (A',B',inv), (C,D,inv), or $(C',D',\mathit{inv}) \in T$.

In the real world **COLkey** is set when two different states are initialized with the same key and same identifier. **COLfwd** (resp., **COLinv**) concerns the state right before (resp., right after) a permutation evaluation, so that **COL** prevents collisions between intermediate states (after data absorption). At a high-level view, the goal of **COL** is twofold. First, it guarantees that every permutation call at the construction level associated to a path that is not a prefix of another path has a new permutation call. Secondly, it prevents the ideal world to release intermediate states that are permutation-inconsistent. On the other hand, **GUESS** corresponds to the adversary in the real world being able to guess a permutation evaluation that was used by the construction. More precisely, **GUESSkey** corresponds to the adversary able to guess one of the initial states, and **GUESSfwd** (resp., **GUESSinv**) corresponds to a forward (resp., inverse) successful primitive query. A transcript T is called *bad* if it sets **COL** $\vee$ **GUESS**. The bad events are illustrated in Figure 4.

Application of the H-coefficient technique. Denote by T_{Real} (resp., T_{Ideal}) the probability distribution on transcripts induced by the real (resp., ideal) world, and let τ be a good transcript. In particular the transcript is permutation-consistent, thus reachable in the real world. Moreover, in the real world, every permutation call which does not correspond to a repeated subpath is fresh. On the other side, in the ideal world, one transcript corresponds to N permutation outputs, and M random intermediate states. Therefore,

$$\frac{\mathbf{Pr}\left(T_{\text{Real}} = \tau\right)}{\mathbf{Pr}\left(T_{\text{Ideal}} = \tau\right)} = \frac{(2^b)_N \times (2^b)^M}{(2^b)_{M+N}} \geq 1\,.$$

Now, it remains to upper bound the probability to obtain a bad transcript in the ideal world. One important remark for both **COL** and **GUESS** is that the adversary is mostly non-adaptive for these events. Indeed, the proof relies on the randomness of the intermediate states, which are generated and released only at the end of the interaction. In particular, the adversary has to commit to the data to absorb before obtaining the corresponding intermediate state.

We now proceed with the probability of **COL**. The only possibility for the adversary to set this event during the interactive phase is to have two construction queries with $\text{par}(E) = \text{par}(E')$. In this case **COL** (or more precisely **COLinv**) is set if and only if $Z \oplus Z' = E_{\text{last}} \oplus E_{\text{last}}$. We can reason in a query-wise fashion, and conclude that this event is set with probability at most $\frac{q(q-1)}{2^{b+1}}$. For the non-interactive case of **COLinv**, there are $\frac{M(M-1)-q(q-1)}{2}$ candidate pairs in T, and for each pair the probability that it sets **COLinv** is $\frac{1}{2^b}$. Therefore,

$$\mathbf{Pr}\left(\mathcal{A}^{W_I} \text{ sets } \mathbf{COLinv}\right) \leq \frac{M(M-1)}{2^{b+1}}\,. \tag{2}$$

For **COLfwd**, this event can be set only after the interaction phase, and there are at most $\frac{M(M-1)}{2}$ pairs that can set this event. For each pair, **COLfwd** is set with probability $\frac{1}{2^b}$. Therefore,

$$\mathbf{Pr}\left(\mathcal{A}^{W_I} \text{ sets } \mathbf{COLfwd}\right) \leq \frac{M(M-1)}{2^{b+1}}\,. \tag{3}$$

Finally, regarding **COLkey**, a collision can only occur between keys that have the same identifier. Therefore,

$$\mathbf{Pr}\left(\mathcal{A}^{W_I} \text{ sets } \mathbf{COLkey}\right) \leq \frac{\sum_{i=1}^{s} \mu_i(\mu_i - 1)}{2 \times 2^{\mathcal{H}_{\text{col}}(\mathcal{D}_{\text{key}})}}\,. \tag{4}$$

Combining (2), (3), and (4) together, we obtain

$$\mathbf{Pr}\left(\mathcal{A}^{W_I} \text{ sets } \mathbf{COL}\right) \leq \frac{M(M-1)}{2^b} + \frac{\sum_{i=1}^{s} \mu_i(\mu_i - 1)}{2 \times 2^{\mathcal{H}_{\text{col}}(\mathcal{D}_{\text{key}})}}\,. \tag{5}$$

We now focus on the probability of **GUESS**. This event can only be set at the end of the interaction. Let N_{fwd} be the number of forward primitive queries, and N_{inv} be the number inverse primitive queries, so that $N = N_{fwd} + N_{inv}$. We start with **GUESSkey**. One forward primitive query of the adversary fixes the identifier, so that one query targets at most $\max_i \mu_i$ keys simultaneously. Therefore,

$$\mathbf{Pr}\left(\mathcal{A}^{W_I} \text{ sets } \mathbf{GUESSkey}\right) \leq \frac{N_{fwd} \max_i \mu_i}{2^{\mathcal{H}_{\min}(\mathcal{D}_{\text{key}})}} . \tag{6}$$

Now, regarding **GUESSfwd**, there are at most M states to guess. Note that these states are not necessarily independent from each other. Indeed, if the adversary makes M construction queries, each with only one block of data absorbed, then the set of states to guess are of form $S \oplus E_i$, where E_i is the i^{th} block of data. In this (worst) case, each of the states is uniformly random, and one failed attempt from the adversary eliminates at most M states from the list of candidate forward queries. Therefore,

$$\mathbf{Pr}\left(\mathcal{A}^{W_I} \text{ sets } \mathbf{GUESSfwd}\right) \leq \frac{N_{fwd} M}{2^b - MN} . \tag{7}$$

For **GUESSinv**, we obtain similarly

$$\mathbf{Pr}\left(\mathcal{A}^{W_I} \text{ sets } \mathbf{GUESSinv}\right) \leq \frac{N_{inv} M}{2^b - MN} . \tag{8}$$

Thus combining (6), (7), and (8) together, we obtain

$$\mathbf{Pr}\left(\mathcal{A}^{W_I} \text{ sets } \mathbf{GUESS}\right) \leq \frac{2NM}{2^b} + \frac{N \max_i \mu_i}{2^{\mathcal{H}_{\min}(\mathcal{D}_{\text{key}})}} , \tag{9}$$

where we used $NM \leq 2^{b-1}$. We can therefore conclude by plugging (5) and (9) together, which gives the probability that a bad transcript occurs in the ideal world.

4.3 Tightness of the Bound

The bound of Theorem 1 is tight when the number of blocks per construction query is small, and in the case of uniform key sampling. Setting **COL**, **GUESSfwd** $\vee$ **GUESSinv**, or **GUESSkey** allows in a straightforward way to mount distinguishing attacks that succeeds with high probability, and we describe them in the following.

Attack with $M^2 \approx 2^{b/2}$. The following attack exploits **COLfwd** $\vee$ **COLinv**, and uses only one construction oracle.

1. Make $\approx 2^{b/2}$ construction queries with input $(D_i||0^b||1)$, for $D_i \xleftarrow{\$} \{0,1\}^{b-1}$. In both worlds, with high probability, there exists $i \neq j$ such that $Z_i = Z_j$;
2. For every collision with $Z_i = Z_j$, make a construction query with input $(D_i||0^{2b}||1)$ and $(D_j||0^{2b}||1)$. Denote the answers by respectively Z'_i and Z'_j.
3. If there exists a collision $Z'_i = Z'_j$ from step 2, return 0, otherwise 1.

In the real world, it is likely that there exists a pair $i \neq j$ such that a collision occurs after having absorbed $D_i||0$ and $D_j||0$, resulting in a collision $Z_i = Z_j$ that carries over to the construction query described in 2. In the ideal world, it is unlikely that the collision carries over, so that the distinguisher almost always return 1 while interacting with the ideal world.

Attack with $NM \approx 2^b$. The following attack exploits directly **GUESSfwd**$\vee$**GUESSinv**, and uses only one construction oracle.

1. Make N inverse primitive queries and obtain $(X_i, Y_i, inv) \in T$, with Y_i sampled uniformly at random without repetition;
2. Make M construction queries with input $(D_i||1)$, where D_i is sampled uniformly at random without repetition;
3. If there exists i, j such that $Y_i \oplus X_i = Z_j$, let $S = (D_i||1) \oplus X_i$. Then S is a candidate for the init state in the real world.
4. Take $D \in \{0,1\}^{b-1}$ which has not been sampled before. Make one construction query with input $(D||1)$, obtain Z, and make one forward primitive query with input $S \oplus (D||1)$. Call the output Y, and if $Z = Y \oplus S \oplus (D||1)$, return 0, otherwise 1.

In the real world, with high probability the adversary will guess correctly the key, while in the ideal world, having in step item 4 the construction query matching the primitive query is highly unlikely.

Attack when $\sum_i \mu_i(\mu_i - 1) \approx 2^\kappa$. The adversary makes a constant number of construction queries to each oracle (for example 10 queries to each oracle), and if there exists $m \neq m'$ such that the outputs are all the same, then the adversary is in the real world with high probability.

Attack with $N \approx 2^\kappa / \max_i \mu_i$. One can directly exploit **GUESSkey** to mount an attack, by making forward queries with the identifier associated to the largest number of users. The attack is then similar to the attack exploiting **GUESSfwd** $\vee$ **GUESSinv**.

5 Building a Deck Function from Kirby

In this section we define a construction on top of Kirby to build a deck function [DHAK18]. This allows using Kirby for the wide variety of deck function modes [BDH+22]. The most straightforward application is the generation of a keystream for stream encryption, with a diversifier as input.

Our deck function makes use of two mappings that we specify in the following sections. In Section 5.1 we specify an injective mapping that encodes a sequence of arbitrary-length strings to single string. In Section 5.2 we specify a mapping that encodes pairs of a string and an integer to b-bit block string sequences such that its codomain forms a prefix-free set. In Section 5 we specify our deck function mode on top of Kirby using these two mappings.

We denote the length in bytes of a bytestring M by $\mathsf{bytelen}(M)$, the encoding of an integer x in the range $[0, 255]$ in a byte by $\text{encByte}(x)$ and the encoding of an integer x in the range $[0, 2^{8B} - 1]$ in a B-byte block $\text{encBlock}_B(x)$.

5.1 Injective Mapping from String Sequences to a Single String

The encoding takes a non-empty sequence of an arbitrary number of strings, each of arbitrary length and returns a single string. It does this by concatenating strings, where each string is followed by an encoding of its byte length. The latter is decodable starting from the end of the string and this makes the full string decodable.

We first specify the length encoding in Algorithm 2. It encodes a length as a sequence of bytes $\ldots b_{-2} b_{-1} b_0$. All bytes except the first, namely the one with the smallest index, have their most significant bit (MSB) set to 1. This allows determining the first byte of the length encoding string. The value represented by a string $b_{1-n} \dots b_{-2} b_{-1} b_0$ is given by $\ell = \sum_{0<i\leq n} (b_{i-n} \bmod 2^7) 2^{7i}$

Algorithm 2 Length encoding $L \leftarrow \mathsf{encodeLength}(\ell)$

```
**Input**: integer $\ell$
**Output**: byte string $L$ encoding the integer
$x \leftarrow \ell \bmod 2^7$
$\ell \leftarrow (\ell - x)/2^7$
$L \leftarrow \text{encByte}(x)$
**while** $\ell > 0$ **do**
    $x \leftarrow \ell \bmod 2^7$
    $\ell \leftarrow (\ell - x)/2^7$
    $L \leftarrow L || \text{encByte}(x + 2^7)$
**end while**
**return** $L$
```

We now specify our injective encoding function SequenceToString() in Algorithm 3. It is injective as the input strings M_i can be recovered one by one from the back. It suffices the recover the length ℓ from the end of D, and we can isolate the last input byte string. This can be done recursively for all other input byte strings.

Algorithm 3 Injective encoding $\mathsf{SequenceToString}(M_0, ..., M_{m-1})$

```
**Input**: non-empty sequence of byte strings $M_0, M_1, \dots M_{m-1}$
**Output**: string $D$
$D \leftarrow \epsilon$
**for** all strings $M_i$ **do**
    $D \leftarrow D || M_i || \mathsf{encodeLength}(\mathsf{bytelen}(M_i))$
**end for**
**return** $D$
```

5.2 Prefix-Free Encoding

We specify our prefix-free encoding function Prefix() in Algorithm 4. It pads the input byte sequence to a multiple of B bytes with $b = 8B + \ell$ and $0 < \ell \leq 8$. It then splits in B-byte blocks and appends to each block 0^ℓ forming the blocks of the output E that we call the string blocks. Subsequently, it encodes the counter value in a B-byte block, appends $10^{\ell-1}$ to form the last block of E, called the counter block. Clearly the codomain forms a prefix-free set as there is domain separation between last blocks and the other blocks.

5.3 Kirby-DECK

We define Kirby-DECK in Algorithm 5.

The deck mode can be efficiently implemented by the fact that the Kirby inputs E only differ in their last block, the counter block. The Kirby state after the absorbing the string blocks of E can be cached and then the output sequence can be computed by applying the transformation F to the bitwise sum of that state and the counter block. Hence the total number of calls to F for a deck function call is the number of string blocks plus $\lceil n/B \rceil$.

Algorithm 4 Prefix-free encoding $\mathsf{Prefix}(D, \mathrm{cnt})$

```
**Input**: string $D$ and counter value $i \in \mathbb{N}$
**Output**: sequence of $b$-bit blocks $E_0, E_1, \dots E_{t-1}$
$\ell \leftarrow b \bmod 8$
if $\ell = 0$ then
    $\ell \leftarrow 8$
end if
$B \leftarrow (b - \ell)/8$
$i \leftarrow 0$
Pad $D$ with $10^*$ padding up to a multiple of $B$ bytes
while $\mathsf{bytelen}(D) \geq 0$ do
    $E_i \leftarrow$ first $B$ bytes of $D$
    $E_i \leftarrow E_i || 0^\ell$
    Remove first $B$ bytes of $D$
    $i \leftarrow i + 1$
end while
$E_{i+1} \leftarrow \mathrm{encBlock}_B(\mathrm{cnt}) || 10^{\ell-1}$
```

Algorithm 5 Kirby-DECK(M, n)

```
**Input**: sequence of string $M = M_0, M_1, \dots, M_{m-1}$ and requested output byte length
$n \in \mathbb{N}$
**Output**: $n$-bit string $Z$
$D \leftarrow \mathsf{SequenceToString}(M_0, ..., M_{m-1})$
$Z \leftarrow \epsilon$
$\mathrm{cnt} \leftarrow 0$
while $\mathsf{bytelen}(Z) < n$ do
    $E \leftarrow \mathsf{Prefix}(D, \mathrm{cnt})$
    $Z = Z || \mathrm{Kirby}(E)$
    $\mathrm{cnt} \leftarrow \mathrm{cnt} + 1$
end while
Truncate $Z$ to its first $n$ bytes
return $Z$
```

6 Acknowledgements

We thank Bart Mennink for useful discussions. Charlotte Lefevre is supported by the Netherlands Organisation for Scientific Research (NWO) under grant OCENW.KLEIN.435. Yanis Belkheyar is supported by Intel through the Crypto FrontiersResearch Center. Joan Daemen is supported by the European Research Council under the ERC advanced grant agreement under grant ERC-2017-ADG Nr. 788980 ESCADA.

References

[BBN22] Arghya Bhattacharjee, Ritam Bhaumik, and Mridul Nandi. A sponge-based PRF with good multi-user security. IACR Cryptol. ePrint Arch., page 1146, 2022.

[BDH+22] Norica Bacuieti, Joan Daemen, Seth Hoffert, Gilles Van Assche, and Ronny Van Keer. Jammin' on the deck. In Shweta Agrawal and Dongdai Lin, editors, Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part II, volume 13792 of Lecture Notes in Computer Science, pages 555–584. Springer, 2022.

[BDP+16] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Farfalle: parallel permutation-based cryptography. IACR Cryptol. ePrint Arch., page 1188, 2016.

[BDPA11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the sponge: Single-pass authenticated encryption and other applications. In Ali Miri and Serge Vaudenay, editors, Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers, volume 7118 of Lecture Notes in Computer Science, pages 320–337. Springer, 2011.

[BDPV07] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge functions. Ecrypt Hash Workshop 2007, May 2007.

[BDPV11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak reference. SHA-3 competition (round 3), 2011. https://keccak.team/papers.html.

[BDPVA12] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Permutation-based encryption, authentication and authenticated encryption. Directions in Authenticated Ciphers, pages 159–170, 2012.

[Ber08] Daniel J. Bernstein. The salsa20 family of stream ciphers. In Matthew J. B. Robshaw and Olivier Billet, editors, New Stream Cipher Designs - The eSTREAM Finalists, volume 4986 of Lecture Notes in Computer Science, pages 84–97. Springer, 2008.

[CLMP21] Yu Long Chen, Atul Luykx, Bart Mennink, and Bart Preneel. Systematic security analysis of stream encryption with key erasure. IEEE Trans. Inf. Theory, 67(11):7518–7534, 2021.

[CS14] Shan Chen and John P. Steinberger. Tight security bounds for key-alternating ciphers. In Phong Q. Nguyen and Elisabeth Oswald, editors, Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings, volume 8441 of Lecture Notes in Computer Science, pages 327–350. Springer, 2014.

[Dam89] Ivan Damgård. A design principle for hash functions. In Gilles Brassard, editor, Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings, volume 435 of Lecture Notes in Computer Science, pages 416–427. Springer, 1989.

[DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2: Lightweight Authenticated Encryption and Hashing. J. Cryptol., 34(3):33, 2021.

[DHAK18] Joan Daemen, Seth Hoffert, Gilles Van Assche, and Ronny Van Keer. The design of xoodoo and xoofff. IACR Trans. Symmetric Cryptol., 2018(4):1–38, 2018.

[DM23] Christoph Dobraunig and Bart Mennink. Generalized initialization of the duplex construction. IACR Cryptol. ePrint Arch., page 924, 2023.

[DMA17] Joan Daemen, Bart Mennink, and Gilles Van Assche. Full-state keyed duplex with built-in multi-user support. In Tsuyoshi Takagi and Thomas Peyrin, editors, Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II, volume 10625 of Lecture Notes in Computer Science, pages 606–637. Springer, 2017.

[Mer89] Ralph C. Merkle. One way hash functions and DES. In Gilles Brassard, editor, Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings, volume 435 of Lecture Notes in Computer Science, pages 428–446. Springer, 1989.

[Pat08] Jacques Patarin. The "coefficients h" technique. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers, volume 5381 of Lecture Notes in Computer Science, pages 328–345. Springer, 2008.

[PGV93] Bart Preneel, René Govaerts, and Joos Vandewalle. Hash functions based on block ciphers: A synthetic approach. In Douglas R. Stinson, editor, Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings, volume 773 of Lecture Notes in Computer Science, pages 368–378. Springer, 1993.