Kirby: A Robust Permutation-Based PRF Construction

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, while the sponge/duplex can squeeze at most $b - c$ 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. This construction could be seen as a generalization of the construction underlying the stream cipher family Salsa. Furthermore, we define a simple mode on top of Kirby that turns it into a deck function with parallel expansion. This is similar to Farfalle but it has a much smaller memory footprint. Furthermore we prove that in the Kirby construction, the leakage of intermediate states does not allow recovering the key or *earlier states*.

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

1 Introduction

Permutation-based cryptography has become increasingly popular in the last years. Many of the proposed schemes make use of the sponge [BDPV07], duplex [BDPA11] and monkeyduplex constructions [BDPVA12]. For example, the winner of the NIST SHA-3 competition, the family of extendable output functions (XOF) Keccak [BDPV11] is based on the sponge construction. More recently, the NIST lightweight cryptography competition was won by the authenticated encryption scheme Ascon [DEMS21] that is based on a variant of monkeyduplex [BDPVA12]. These schemes have the property that part of the permutation output cannot be presented at the final output. This is called the *inner state*. Its size is called the capacity c and the security of sponge/duplex is limited to c bits. If we denote the permutation width by b, this leaves a *rate* of only $b - c$. Moreover, they are strictly serial in nature: the construction imposes a strict sequence in the execution of permutation calls. However, modern keyed application of duplex/sponge supports *full-state absorbing* [BDPVA12].

Around the time of the advent of sponge and duplex construction the stream cipher family Salsa was introduced in [Ber08], and later an improved variant called ChaCha was proposed in [B+08] In these ciphers a cryptographic permutation is applied to an input consisting of the key, a nonce, a fixed constant and the (block) counter. The corresponding keystream block is the sum of the permutation output and the input. The security goal is to achieve a pseudorandom function (PRF). Their advantage compared to sponge and duplex is that a keystream block is b bits instead of $b-c$. In sponge-vocabulary, it supports *full-state squeezing*. Moreover, it is fully parallelizable.

A more recent permutation-based construction that combines advantages of Salsa/ChaCha and sponge/duplex is Farfalle [BDP+16]. It builds a so-called doubly extendable cryptographic keyed (deck) function that support both variable-length input and output. Like Salsa and ChaCha it supports full-state squeezing and is parallelizable and like sponge it is variable-input-length (VIL) and variable output length (VOL). It has been instantiated with the Keccak permutation under the name Kravatte and with the Xoodoo permutation in [DHAK18] under the name Xoofff.

Our Contribution. In Section 3, we introduce a **novel permutation-based VIL-PRF construction called Kirby**.

Kirby is suitable to build efficient lightweight cryptographic primitives with low memory footprint and/or low latency (as shown in Fig. 1). It is inspired by the sponge construction and Salsa. Over sponge and duplex it has the advantage that it offers full-state squeezing and over Salsa it has the advantage that it has arbitrary-length input. Kirby could also be seen as a generalization of Salsa [Ber08] and XSalsa [Ber11], as discussed in Section 3.3. We also introduce in Section 8 a mode on top Kirby that uses injective prefix-free input and counter encoding, resulting in a deck function. The squeezing phase of the Kirby deck mode is similar to that of Farfalle [BDP+16] but Kirby has a significantly smaller memory footprint. Table 1 summarizes a comparison between Kirby and existing permutation-based schemes.

In Section 6, we prove a **tight bound on the PRF distinguishing advantage** in the random permutation model and in the multi-user setting supporting arbitrary key distributions. Kirby employs identifiers to enhance multi-user security, a practice observed, among others, in the sponge-based PRF Muffler [BBN22]. 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 by Dobraunig and Mennink [DM24] by treating different restrictions on the initialization of the keyed duplex.

In our construction, full-state squeezing is possible thanks to continuous feedforward as in the Davies-Meyer construction employed in Merkle-Damgård hashing [Mer89, Dam89, PGV93]. The presence of feed-forward allows us to prove a **tight bound on the leakage resilience** in the random permutation model in Section 7. The stream cipher construction from Chen et al. [CLMP21] and Kirby share the security objective that the leakage of a state does not compromise the recovery of the key or earlier states. However, Kirby is based on a public primitive, whereas the approach of Chen et al. relies on a keyed primitive, resulting in significant differences in the security models and proofs.

Our PRF and leakage resilience bounds are two special cases of one distinguishing game that is defined in Section 4 and for which we prove a bound in Section 5. We also introduce in Section 8 a mode on top Kirby that uses injective prefix-free input and counter encoding, resulting in a deck function.

Table 1: Comparison of Kirby with permutation-based designs. "Multi" means that a dedicated bound exists for multi-user security that can surpass the single-user one with generic composition.

Name	Full State		Construction Security Proof		Leakage Resilience	Ref
	Absorb	Squeeze	Single	Multi		
Farfalle	✓	✓	✗[1]	✗	✗(Not LR)	[BDP+16]
Keyed Sponges	✓	✗	✓	✓ [DMA17, DM24]	✓(Sometimes, e.g., [DM19])	[BDPV07, BDPA11]
(X)Salsa	✗	✓	✓	✗	✗(XSalsa not LR)	[Ber08, Ber11]
Kirby	✓	✓	✓	✓	✓	This work

[1]While there exists no formal security analysis of Farfalle, [ABJ+22] describes and proves the security of Farasha, which is a Farfalle-like construction.

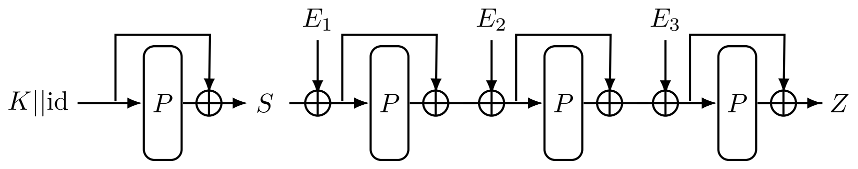

Figure 1: Example of iteration of Kirby (Algorithm 1), for an input sequence E with $n = 3$.

2 Preliminaries

Notation. We introduce here useful notation. We will use $\mathbb{N}$ to denote the set of natural numbers excluding 0. Let $m, n \in \mathbb{N}$ such that $m \leq n$. We use $[\![m, n]\!]$ to denote the set $\{m, m+1, \ldots, n\}$. We use the letter b to denote the width of a permutation and κ for the length of the keys. Let $d \in \mathbb{N}$. The set $\{0,1\}^d$ denotes the strings of length d bits, and $\{0,1\}^*$ denotes $\bigcup_{a \in \mathbb{N}} \{0,1\}^a$. 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 $s_1 \oplus s_2$ to denote the bitwise addition of s_1 and s_2.

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 by $\mathbf{Perms}(b)$. 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'. Given $E, E' \in \{0,1\}^*$ such that $E \prec E'$, $E' \setminus E$ denotes the string obtained by removing from E' its first $|E|$ bits.

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

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

The minimum collision entropy is defined by

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

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 $\mathcal{D}$ placed in world W_a, for $a \xleftarrow{\$} \{0,1\}$, that we denote by $\mathcal{D}^{W_a}$. Without loss of generality, we assume that $\mathcal{D}$ is a deterministic algorithm that never makes queries for which it already knows the answer. The advantage of $\mathcal{D}$ is defined as follows:

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

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 $\mathcal{D}$ 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 $T = T_{\mathrm{GOOD}} \cup T_{\mathrm{BAD}}$. If there exists $\epsilon_1, \epsilon_2 > 0$ such that

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

then

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

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. In order to avoid length-extension-like attacks, the set of strings queried to Kirby must form a prefix-free set.

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: $\mathsf{F}(S) = P(S) + S$. It takes as input a sequence of b-bit blocks E of length at least 1. First, it initializes the state S with the concatenation of a κ-bit key K and a $(b - \kappa)$-bit key identifier id. It sequentially absorbs the blocks of E by adding each block to the state one by one 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 Fig. 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 $E = (E_1, \ldots, E_n)$
  Output: a $b$-bit string $Z$
  $S \leftarrow \mathrm{K}||\mathrm{id}$
  $S \leftarrow S \oplus P(S)$
  for $i = 1$ to $n$ do
      $S \leftarrow S \oplus E_i$
      $S \leftarrow S \oplus P(S)$
  end for
  return $Z \leftarrow S$
```

3.2 Design Rationale

The Kirby construction builds a VIL-PRF from a permutation. Its main objectives are to be simple, versatile, robust, have good multi-user provable security in the random permutation model and allow for compact implementation.

The overall design of Kirby relies solely on bitwise additions and the permutation itself. We opted for bitwise addition due to its typically low cost in hardware implementations.

It is worth noting that this operation could alternatively be replaced by modular additions or by word-by-word modular additions, as done for example in Salsa20 with 32-bit words. The memory required by a round-based implementation of Kirby instantiated with an iterated permutation is only twice the width of the permutation. Altogether, Kirby offers compactness in its design.

The critical path of the construction for single-block inputs E, which is the minimal time between availability of the last block (i.e. latency) of E and that of the output, is the critical path of the permutation, the input block addition, 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.

In the latter sections, we show how using identifiers, the feedforward, and the restriction on prefix-freeness of the input are essential to obtain a good security bound.

3.3 Comparison with Salsa

Kirby can be seen as a generalization of the construction underlying the stream cipher Salsa20 [Ber08]. Salsa20 operates on a state S of size 512 bits, which is itself split into 16 different 32-bit words. S is initialized with the 64-bit counter, the 64-bit nonce, the 256-bit key, and a 128-bit constant. Then, the permutation is applied to S, and each 32-bit word of the output is fed-forward with the corresponding word in S by adding them modulo 2^{32}. Moreover, Salsa20 has been generalized to XSalsa20 [Ber11] in the fact that it supports longer nonces. In XSalsa20, the 128 first bits of the nonce, along with the 256-bit key and the 128-bit constant are fed into the permutation. Then, 256 bits of the obtained output are truncated, forming a key to Salsa20, with the remaining 64 bits of the nonce and 64-bit counter used as such.

Therefore, Kirby could be seen as a Salsa-like construction that supports arbitrary-length input and with full-state absorption. However, Kirby differs from Salsa20 in several aspects. Firstly, Kirby systematically applies a feed-forward with bitwise addition after each permutation call. Additionally, XSalsa20 overwrites a part of the state with the data to absorb, while with Kirby the diversifier is added block by block to the state, allowing thus full-state absorption. Lastly, Kirby is designed with multi-user security in mind, featuring identifiers with a dedicated security proof.

4 Security Model

We will prove the security of Kirby under two different security models, specified in respectively Sections 4.1 and 4.2, and describe in this paragraph the common setup. Our security proofs will be 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 $\boldsymbol{\mu} := (\mu_1, \ldots, \mu_s)$. In the two security models, the adversary has access to a primitive oracle $\mathcal{O}_P$, which takes as input a string in $\{0,1\}^b$, and a bit that denotes the direction of the query. The adversary has additionally access to μ construction oracles, denoted by $\mathcal{O}_{C_1}, \ldots, \mathcal{O}_{C_\mu}$. The set of strings queried via the construction oracles is denoted by $\mathbb{E}$, and contains elements of form (m, E), where the block sequence $E \in \{0,1\}^{b+}$ has been queried to the oracle O_{C_m}. For $m \in [\![1, \mu]\!]$, $\mathbb{E}_m$ is defined as follows:

$$\mathbb{E}_m = \left\{ E \in \{0,1\}^{b+} \mid (m, E) \in \mathbb{E} \right\} .$$

The key and identifier of user m are denoted respectively by K_m and id_m. We will use $\mathtt{IK}[m]$ as an abbreviation for $\mathrm{K}_m || \mathrm{id}_m$. We will assume that the adversary always make at least one query to each oracle $\mathcal{O}_{C_m}$, as otherwise, we can rather use the security bound for a number of users equal to the number of queried construction oracles.

4.1 Indistinguishability

Our goal is to prove an upper bound on the advantage of any adversary to distinguish μ instances of Kirby based on a random permutation from μ independent random oracles, under the assumption that for each construction oracle $\mathcal{O}_{C_m}$, the set of queried strings $\mathbb{E}_m$ is prefix-free.

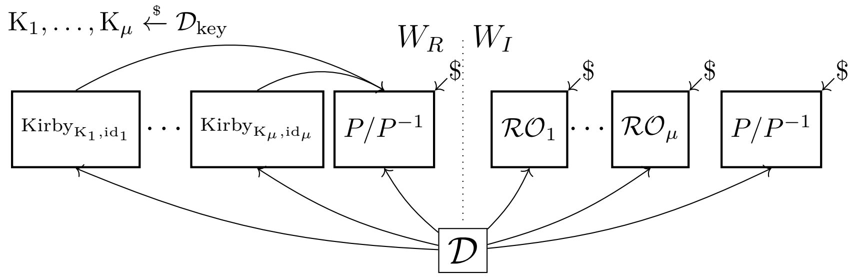
$\mathrm{K}_1, \ldots, \mathrm{K}_\mu \xleftarrow{\$} \mathcal{D}_{\mathrm{key}}$
W_R
W_I
\$
\$
\$
\$
$\mathrm{Kirby}_{\mathrm{K}_1,\mathrm{id}_1}$
$\cdots$
$\mathrm{Kirby}_{\mathrm{K}_\mu,\mathrm{id}_\mu}$
P/P^{-1}
$\mathcal{RO}_1$
$\cdots$
$\mathcal{RO}_\mu$
P/P^{-1}
$\mathcal{D}$

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

Specification of the Worlds. The security model is a special type of distinguishing game, akin to multi-user PRF. The adversary has to distinguish between the so-called real and ideal worlds, which are denoted respectively by W_R and W_I. In both worlds, the adversary $\mathcal{D}$ has access to a primitive oracle $\mathcal{O}_P$ and μ construction oracles denoted by $\mathcal{O}_{C_1}, \ldots, \mathcal{O}_{C_\mu}$, as described in detail at the beginning of Section 4. Let s be the number of coordinates in $\boldsymbol{\mu}$, 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 element of $\boldsymbol{\mu}$ at position t. Let $\mathrm{id}_m = x_{\texttt{Expand}\boldsymbol{\mu}[m]}$. Each of the worlds is specified as follows:

- In W_R, $P \xleftarrow{\$} \mathbf{Perms}(b)$, $\mathrm{K}_1, \ldots, \mathrm{K}_\mu \xleftarrow{\$} \mathcal{D}_{\mathrm{key}}$, $\mathcal{O}_{C_m}$ 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 or P^{-1};
- In 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 or P^{-1}. We stress that $\mathcal{RO}_1, \ldots, \mathcal{RO}_\mu$ are independent.

Fig. 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 permutation 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 queries that are required by construction queries in W_R. In the example above, $M = \ell + \ell' - x$. M will be also referred to as the *online complexity.* Moreover, we define by N the number of permutation queries made by the adversary. N will be also referred to as the *offline complexity.*

We now have all ingredients to define the security model. Let $\mathcal{D}_{\mathrm{key}}$ 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 $\mathrm{Adv}^{\mathrm{PRF}}_{\mathrm{Kirby}}(N, M, \boldsymbol{\mu}, \mathcal{D}_{\mathrm{key}})$ as the maximum advantage $\mathrm{Adv}(W_R, W_I)(\mathcal{D})$ over all distinguishers $\mathcal{D}$ such that:

- $\mathcal{D}$ has access to μ construction oracles in both W_R and W_I;
- In W_R, keys are sampled according to $\mathcal{D}_{\mathrm{key}}$, identifiers are partitioned according to $\boldsymbol{\mu}$;
- For each $m \in [\![1, \mu]\!]$, the set $\mathbb{E}_m$ is prefix-free;
- The construction queries made by $\mathcal{D}$ require in W_R a minimum number of permutation evaluation equal to M without considering duplicate evaluations from repeated prefixes (see paragraph "Metrics for queries" of Section 4.1 for more detailed explanations);
- $\mathcal{D}$ makes N permutation queries.

The quantity $\mathrm{Adv}^{\mathrm{PRF}}_{\mathrm{Kirby}}(N, M, \boldsymbol{\mu}, \mathcal{D}_{\mathrm{key}})$ is bounded in Lemma 1.

4.2 Leakage Resilience

Leakage resilience captures the property that revealing a state S_E after absorption of a block sequence $E \in \{0,1\}^{b+}$ does not compromise the secrecy of any state $S_{E'}$ obtained after absorbing a prefix block sequence $E' \prec E$ if E' has never been queried. These latter states, stemming from unqueried prefixes, will be referred to as "earlier states". Note that the earlier states include the state containing the key.

At first sight, such a property seems already captured by the PRF distinguishing game, as the latter indicates among others that having access to one state does not compromise the secrecy of the earlier intermediate states. However, in the leakage resilience setting, the adversary has slightly more freedom notably in the fact that its queries to each construction oracle do not necessarily form a prefix-free set. To illustrate this point, consider the following scenario: i) a construction query is made with input (E_1, E_2), the adversary obtains the associated state ii) later on, a construction query to the same oracle is made with input (E_1, E_2') with $E_2 \neq E_2'$, but this time, the state after having absorbed E_1 leaks. The queries do not form a prefix-free set, thus such a scenario would not be covered by the PRF distinguishing game. Moreover, as the adversary is adaptive, we cannot swap the queries at steps (i) and (ii) without loss of generality on the class of adversaries considered. We suspect that adaptivity in that case does not help the adversary to increase its success probability, but this needs to be proven formally, as explained in [Mau02, JÖS12]. Instead, we opted for slightly extending the class of adversaries considered by the distinguishing game. Looking ahead, in Section 5 we extend the prefix-freeness condition to the two conditions 1 and 2. Theorem 1 bounds the distinguishing advantage of such adversaries, and Lemma 2 leverages the theorem to bound the leakage resilience of Kirby.

The security game underlying leakage resilience is not a distinguishing game, but rather a guessing game, and is illustrated in Fig. 3. The adversary has access to the construction oracles and primitive oracles from the real world W_R from Fig. 2, and has additional access to a forgery oracle `Forge`. The latter takes as input a tuple $((m, E), S)$ with $m \in [\![1, \mu]\!]$, $E \in \{0,1\}^{b+}$, and $S \in \{0,1\}^b$, and returns $\top$ whenever the following three conditions are satisfied:

- The input (m, E) is a prefix of an element in $\mathbb{E}$;
- The input (m, E) does not contain a prefix in $\mathbb{E}$;
- S equals to $\mathcal{O}_{C_m}(E)$.

If at least one of these conditions is not satisfied, `Forge` returns $\perp$. We assume that the adversary submits all of its forgery attempts at the end of the game, as the adversary can act as if its forgery attempts failed, and postpone its queries to `Forge` at the end of the game.

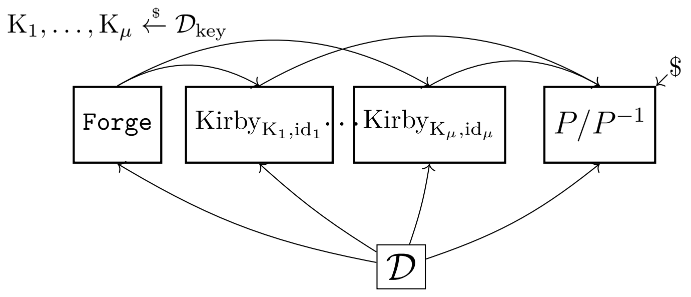
$\mathrm{K}_1, \ldots, \mathrm{K}_\mu \xleftarrow{\$} \mathcal{D}_{\mathrm{key}}$
Forge
$\mathrm{Kirby}_{\mathrm{K}_1,\mathrm{id}_1}$
$\mathrm{Kirby}_{\mathrm{K}_\mu,\mathrm{id}_\mu}$
P/P^{-1}
$
$\mathcal{D}$

Figure 3: Illustration of the leakage resilience security game. The symbol $ on top of an oracle means that the underlying primitive is sampled uniformly at random, and $\mathrm{Kirby}_{\mathrm{K,id}}$ denotes the construction Kirby initialized with the key K and the identifier id.

Metrics for Queries. Compared to the indistinguishability security game, the adversary has additional access to the oracle `Forge`, and we denote by t the total number of queries to this oracle. The probability that an adversary $\mathcal{A}$ has one of its `Forge`-queries returning $\top$ is denoted by $\mathrm{Adv}^{\mathrm{LR}}_{\mathrm{Kirby}}(\mathcal{A})$.

Let $\mathcal{D}_{\mathrm{key}}$ 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 $\mathrm{Adv}^{\mathrm{LR}}_{\mathrm{Kirby}}(N, M, t, \boldsymbol{\mu}, \mathcal{D}_{\mathrm{key}})$ as the maximum advantage $\mathrm{Adv}^{\mathrm{LR}}_{\mathrm{Kirby}}(\mathcal{A})$ over all adversaries $\mathcal{A}$ such that:

- $\mathcal{A}$ has access to μ construction oracles $(\mathcal{O}_{C_m})_{m \in [\![1,\mu]\!]}$;
- The keys are sampled according to $\mathcal{D}_{\mathrm{key}}$, identifiers are partitioned according to $\boldsymbol{\mu}$;
- $\mathcal{A}$ makes N permutation queries;
- The construction queries made by $\mathcal{A}$ require in W_R a minimum number of permutation evaluations equal to M without considering duplicate evaluations from repeated prefixes (see paragraph "Metrics for queries" of Section 4.1 for more detailed explanations);
- $\mathcal{A}$ makes t queries to the oracle `Forge` at the end of the interaction.

5 General Indistinguishability Theorem

In this section, we provide a security proof that bounds the advantage of an adversary in the distinguishing game from Section 4.1, but that covers a class of adversaries slightly larger than the one we target for the PRF advantage. This result will be useful for the PRF indistinguishability advantage, as well as the leakage resilience. The theorem and proof can be found in Section 5.1, while Section 5.2 discusses the tightness of the bound. The adversaries in this distinguishing game are constrained by specific relationships between construction and permutation queries. These restrictions are captured by conditions 1 and 2.

Condition 1. *For any construction query with input (m, E'), the adversary did not make construction queries before it with input (m, E) with $E \prec E'$.*

Condition 2. *None of the following scenarios occur:*

- *There exists a forward permutation query with input X such that at the moment of the query, there exists two construction queries $(m, E), (m, E')$, each with outputs Z and Z' respectively, and such that $E \prec E'$ and X equals the XOR of Z with the first block of $E' \setminus E$.*

- *There exists an inverse permutation query with input Y such that at the moment of the query, there exists two construction queries $(m, E), (m, E||e)$, each with outputs Z and Z' respectively, and such that $Y = Z \oplus Z' \oplus e$.*

These two conditions prevent that the adversary trivially manages to distinguish the real world W_R from the ideal world W_I. At a high-level view, the adversary could distinguish between W_R and W_I in two ways: either in W_R it is able to easily find two different construction queries that have the same output (the associated bad event is named **COL** in the proof), or it detects in W_I the disconnection between construction and permutation queries outputs (the associated bad event is named **GUESS** in the proof). Condition 1 prevents trivial ways to set **COL**, as it forbids the adversary to freely choose in W_R the permutation inputs inside construction evaluations. On the other side, condition 2 targets **GUESS**. It forbids the adversary to make permutation queries for which it already knows the answer in the W_R, or extend via permutation queries its knowledge of intermediate states that are located between two known states S_1, S_2, where S_1 is associated to a path that is a prefix to the one associated to S_2. The latter event is undesirable as the adversary can reach S_2 from S_1 by making solely permutation queries, and detect in W_I the inconsistency.

5.1 Main Theorem and Proof

Theorem 1. *Let $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_s) \in \mathbb{N}^s$, and $N, M \in \mathbb{N}$ such that $\sum_{i=1}^{s} \mu_i \leq M$. Let W_R and W_I be the worlds described in Section 4.1. Consider a distinguisher $\mathcal{D}$ such that:*

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

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

- *$\mathcal{D}$ respects conditions 1 and 2;*

- *The construction queries made by $\mathcal{D}$ require in W_R a minimum number of permutation evaluations equal to M without considering duplicate evaluations from repeated prefixes (see paragraph "Metrics for queries" of Section 4.1 for more detailed explanations);*

- *$\mathcal{D}$ makes N permutation queries.*

Then we have

$$\mathrm{Adv}(W_R, W_I)(\mathcal{D}) \leq \frac{3M(M-1)}{2^b} + \frac{NM}{2^b} + \frac{\sum_{i=1}^{s} \mu_i(\mu_i - 1)}{2 \times 2^{\mathcal{H}_{col}(\mathcal{D}_{key})}} + \frac{N \max_i \mu_i}{2^{\mathcal{H}_{min}(\mathcal{D}_{key})}}\,.$$

The remainder of this subsection is dedicated to the security proof.

Transcript Notation. In the following, we define the transcript induced by the interaction between the distinguisher and the oracles. Denote by q the total number of construction queries. The transcript is an ordered list with $N + q$ elements. Each permutation query results in the addition of a tuple (X, Y, d) to the transcript, where $d \in \{\mathit{fwd}, \mathit{inv}\}$ 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

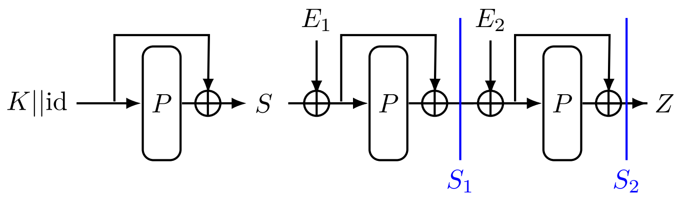

Figure 4: Illustration of the intermediate states in the real world. For this particular example, the tuples (m, S), $(m, (E_1), S_1)$, $(m, (E_1, E_2), S_2)$ belong to the extended transcript T.

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 after the interactive phase, right before the distinguisher outputs its decision bit. More precisely, in the extended transcript that we call T, the elements associated to permutation queries are kept unchanged. Moreover, in the real world, the states $\mathtt{IK}[m]$ are appended to the transcript. In the ideal world, the states $\mathtt{IK}[m]$ computed from dummy keys $K_1, \ldots, K_\mu \xleftarrow{\$} \mathcal{D}_{\mathrm{key}}$ are appended to the transcript. Additionally, one construction query $(m, \mathrm{id}_m, \mathrm{path} = (E_1, \ldots, E_n), Z)$ is split into following $n + 1$ transcript elements:

- A construction initialization element: $(m, \mathrm{path} = \epsilon, S)$
- n different construction absorb elements. For instance if $n = 4$ 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, and there exists no $E' \in \{0, 1\}^{b+}$ such that $E \prec E'$ and E' has been queried to $\mathcal{O}_{C_m}$. 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.
- In the ideal world, S equals $\mathcal{RO}_m(E)$.

To simplify the notation, when the path E equals ϵ, we omit it. There may be duplicates among the construction absorb elements and these are removed from the transcript. There are 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 permutation 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.

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

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

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

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

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, this bad event, coupled with condition 2, 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 permutation query. A transcript T is called *bad* if it sets **COL** $\vee$ **GUESS**. Some of the bad events are illustrated in Fig. 5.

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

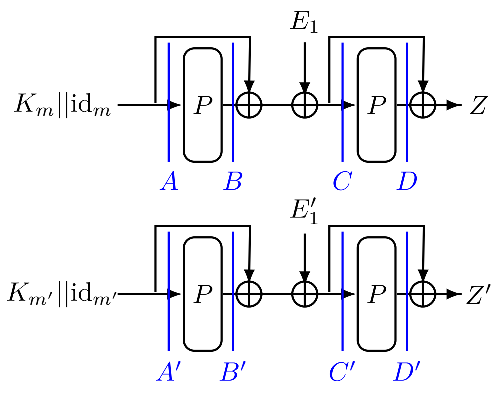

Figure 5: Illustration of the bad events in the real world. **COLkey** corresponds to $A = A'$, **COLfwd** to $C = C'$, $C = A'$, or $C = A$ (not all cases listed), **COLinv** to $D = D'$, $D = B$, $D = B'$, or $B = B'$ (not all cases listed). **GUESSkey** corresponds to (A, B, fwd) or $(A', B', \mathit{fwd}) \in T$. **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 $(C', D', \mathit{inv}) \in T$.

a good transcript. In particular the transcript is permutation-consistent, thus reachable in the real world. Moreover, in the real world, with a good transcript, every permutation call which does not correspond to a repeated subpath is fresh, so that τ induces $N+M$ different permutation calls. On the other side, in the ideal world, one transcript corresponds to N permutation outputs, and M random states. Therefore,

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

Now, it remains to upper bound the probability to obtain a bad transcript in the ideal world. We start with the probability of **COL**. First, **COLkey** can be set only at the end of the interaction. A collision can only occur between keys that have the same identifier. Therefore,

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

To address the event $\bigvee_{\mathbf{x}=1}^{2} \mathbf{COLfwdx} \vee \bigvee_{\mathbf{x}=1}^{3} \mathbf{COLinvx}$, that we denote by **IntCOL** for brievty, we will approach this by reasoning query-wise. For the purpose of this discussion, we keep track of the order at which the intermediate states $S[m, E]$ are generated. For the states generated at the end of the interaction, we impose that a state $S[m, E]$ with $E \neq \epsilon$ is always sampled before its parent $S[m, \mathrm{par}(E)]$. It is important to note that this rule applies not only to the non-interactive phase, but also to the interactive phase thanks to condition 1. Moreover, for this case we can assume that the dummy states $\mathtt{IK}[m]$ are generated at the beginning of the interaction without impacting this probability computation. Let then $i \in [\![1, M]\!]$ represent a specific intermediate state number. We denote by $\mathbf{IntCOL}[i]$ the event that the sampling of the i^{th} intermediate state triggers **IntCOL**. By basic probability theory, we have

$$\mathbf{Pr}\left(\mathbf{IntCOL}\right) \leq \sum_{i=1}^{M} \mathbf{Pr}\left(\mathbf{IntCOL}[i] \mid \neg\mathbf{IntCOL}[i-1]\right) \,,$$

where $\mathbf{IntCOL}[0]$ denotes an event always set. We first treat forward collisions (i.e., we assess $(\mathbf{COLfwd1} \vee \mathbf{COLfwd2})\,[i]$). Thanks to the restriction on the order of sampling, this event can be set only if the i^{th} state has form $S[m, \mathrm{par}(E)]$ and the collision responsible of the bad event has form $S[m, \mathrm{par}(E)] \oplus E_{\text{last}}$, where $S[m, E]$ has already been sampled. Now, for every node $[m, E]$ child of the i^{th} state, and for every previously sampled node $[m, E']$ (with E' possibly the empty string), $S[m, \mathrm{par}(E)] \oplus E_{\text{last}}$ collides with $S[m, \mathrm{par}(E')] \oplus E'_{\text{last}}$ (or $\mathtt{IK}[m']$ if $E' = \epsilon$) with probability $\frac{1}{2^b}$. We denote $\mathtt{NChild}(i)$ as the number of child nodes of node i. We have

$$\mathbf{Pr}\left((\mathbf{COLfwd1} \vee \mathbf{COLfwd2})\,[i] \mid \neg\mathbf{IntCOL}[i-1]\right) \leq \frac{\mathtt{NChild}(i) \times (M-1)}{2^b}\,.$$

Regarding inverse collisions (i.e., $(\mathbf{COLinv1} \vee \mathbf{COLinv2} \vee \mathbf{COLinv3})\,[i]$), this event can be bounded similarly, with a slight subtelty in the fact that the states $S[m]$ are also parent nodes, thus they are doubly counted.

Therefore, using that the sum of all child nodes equals to the number of edges (hence at most M), we obtain

$$\begin{aligned}\mathbf{Pr}\left(\mathbf{IntCOL}\right) &\leq \sum_{i=1}^{M} \frac{\mathtt{NChild}(i) \times 3(M-1)}{2^b} \\ &\leq \frac{3M(M-1)}{2^b}\,. \end{aligned} \tag{2}$$

Combining (1) and (2) together, we obtain

$$\mathbf{Pr}\left(\mathcal{D}^{W_I} \text{ sets } \mathbf{COL}\right) \leq \frac{3M(M-1)}{2^b} + \frac{\sum_{i=1}^{s} \mu_i(\mu_i - 1)}{2 \times 2^{\mathcal{H}_{\text{col}}(\mathcal{D}_{\text{key}})}}\,. \tag{3}$$

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

$$\mathbf{Pr}\left(\mathcal{D}^{W_I} \text{ sets } \mathbf{GUESSkey}\right) \leq \frac{N_{fwd} \max_i \mu_i}{2^{\mathcal{H}_{\min}(\mathcal{D}_{\text{key}})}}\,. \tag{4}$$

For **GUESSinv**2, we similarly have

$$\mathbf{Pr}\left(\mathcal{D}^{W_I} \text{ sets } \mathbf{GUESSinv}2\right) \leq \frac{N_{inv} \max_i \mu_i}{2^{\mathcal{H}_{\min}(\mathcal{D}_{\text{key}})}}\,. \tag{5}$$

Regarding **GUESSfwd** $\vee$ **GUESSinv**1, there are three possibilities to set this event: i) this event is set only at the end of the interaction ii) this event is set during the interactive phase, right after a permutation query, and iii) this event is set during the interactive phase, right after a construction query. Let us first show that case ii) is an impossible case. In the context of **GUESSfwd**, this sub-case would presuppose that, at the time of the permutation query, the adversary already knows two states $S[m, \mathrm{par}(E)]$ and $S[m, E']$ where $E \prec E'$ via construction queries. Now, in the context of **GUESSinv**1, this sub-case would imply that at the moment of the query, the adversary already made the construction queries $(m, \mathrm{par}(E))$ and (m, E). However, given such construction query histories, the act of making the permutation queries specified in the bad event is explicitly prohibited by condition 2. Consequently, we can conclude that case ii) is an impossible occurrence.

The remaining cases i) and iii) can be upper bounded similarly to $\bigvee_{\mathbf{x}=1}^{2} \mathbf{COLfwdx} \vee \bigvee_{\mathbf{x}=1}^{3} \mathbf{COLinvx}$. Indeed, for $i \in [\![1, M]\!]$, the i^{th} generated state can trigger the event $(\mathbf{GUESSfwd} \vee \mathbf{GUESSinv}1)\,[i]$ only if it has form $S[m, \mathrm{par}(E)]$, where $S[m, E]$ has necessarily already been sampled. Therefore, we obtain

$$\begin{aligned}\mathbf{Pr}\left(\mathcal{D}^{W_I} \text{ sets } \mathbf{GUESSfwd} \vee \mathbf{GUESSinv1}\right) &\leq \frac{N_{fwd}M}{2^b} + \frac{N_{inv}M}{2^b} \\ &\leq \frac{NM}{2^b}\,. \end{aligned} \tag{6}$$

Combining (4), (5), and (6) together, we obtain

$$\mathbf{Pr}\left(\mathcal{D}^{W_I} \text{ sets } \mathbf{GUESS}\right) \leq \frac{NM}{2^b} + \frac{N \max_i \mu_i}{2^{\mathcal{H}_{\min}(\mathcal{D}_{\text{key}})}}\,. \tag{7}$$

We can therefore conclude by plugging (3) and (7) together, which gives the probability that a bad transcript occurs in the ideal world. □

5.2 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. The distinguishers described below make construction queries that form a prefix-free set for each oracle. Setting **COL**, **GUESSfwd** $\vee$ **GUESSinv**, or **GUESSkey** allows in a straightforward way to mount distinguishing attacks that succeeds with probability close to the proven bound, 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 permutation queries and obtain $(X_i, Y_i, \mathit{inv}) \in T$, 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 permutation 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 permutation 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**.

6 PRF Advantage of Kirby

Lemma 1. *Let $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_s) \in \mathbb{N}^s$, and $N, M \in \mathbb{N}$ such that $\sum_{i=1}^s \mu_i \leq M$. We have*

$$\mathrm{Adv}^{PRF}_{\mathrm{Kirby}}(N, M, \boldsymbol{\mu}, \mathcal{D}_{key}) \leq \frac{3M(M-1)}{2^b} + \frac{NM}{2^b} + \frac{\sum_{i=1}^s \mu_i(\mu_i - 1)}{2 \times 2^{\mathcal{H}_{col}(\mathcal{D}_{key})}} + \frac{N \max_i \mu_i}{2^{\mathcal{H}_{min}(\mathcal{D}_{key})}}.$$

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

$$\mathrm{Adv}^{\mathrm{PRF}}_{\mathrm{Kirby}}(N, M, \boldsymbol{\mu}, \mathcal{D}_{\mathrm{key}}) \leq \frac{3M(M-1)}{2^b} + \frac{NM}{2^b} + \frac{N}{2^{\mathcal{H}_{\mathrm{min}}(\mathcal{D}_{\mathrm{key}})}}. \tag{8}$$

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

$$\mathrm{Adv}^{\mathrm{PRF}}_{\mathrm{Kirby}}(\mathcal{D}) \leq \frac{\mu^2}{2^{b-130}} + \frac{\mu N}{2^{b-64}} + \frac{N}{2^\kappa}.$$

One can reasonably assume that an adversary $\mathcal{D}$ has a computational power limited to $N \ll 2^{128}$. 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

$$\mathrm{Adv}^{\mathrm{PRF}}_{\mathrm{Kirby}}(N, M, \boldsymbol{\mu}, \mathcal{D}_{\mathrm{key}}) \leq \frac{3M(M-1)}{2^b} + \frac{NM}{2^b} + \frac{\mu(\mu - 1)}{2 \times 2^{\mathcal{H}_{\mathrm{col}}(\mathcal{D}_{\mathrm{key}})}} + \frac{N\mu}{2^{\mathcal{H}_{\mathrm{min}}(\mathcal{D}_{\mathrm{key}})}}.$$

Compared to (8), 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 (i.e., 128 bits), the key length should be increased to at least 192.

Proof of Lemma 1. The statement from Lemma 1 differs from Theorem 1 only in that conditions 1 and 2 form the theorem are replaced by a prefix-freeness condition in the former. An adversary that respects the prefix-freeness condition respects necessarily conditions 1 and 2. Therefore, we can directly bound the quantity $\mathrm{Adv}^{\mathrm{PRF}}_{\mathrm{Kirby}}(N, M, \boldsymbol{\mu}, \mathcal{D}_{\mathrm{key}})$ using Theorem 1. □

7 Leakage Resilience of Kirby

Lemma 2. *Let $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_s) \in \mathbb{N}^s$, and $N, M, t \in \mathbb{N}$ such that $\sum_{i=1}^{s} \mu_i \leq M$, and $t \leq 2^{b-1}$. We have*

$$\mathrm{Adv}^{LR}_{\mathrm{Kirby}}\left(N, M, t, \boldsymbol{\mu}, \mathcal{D}_{key}\right) \leq \frac{2t}{2^b} + \frac{3M(M-1)}{2^b} + \frac{(N+M)M}{2^b} + \frac{\sum_{i=1}^{s} \mu_i(\mu_i - 1)}{2 \times 2^{\mathcal{H}_{col}(\mathcal{D}_{key})}} + \frac{(N+M)\max_i \mu_i}{2^{\mathcal{H}_{min}(\mathcal{D}_{key})}} .$$

As it is reasonable to assume that the online complexity M is smaller than the offline complexity N, the bound for the leakage resilience is comparable to the one obtained for the PRF distinguishing advantage modulo constant factors, and the extra $2t/2^b$ term.

Proof of Lemma 2. The proof consists of multiple small security reductions, and eventually leverages Theorem 1. Consider an adversary $\mathcal{A}$ with resources specified by the theorem statement. We implicitly assume that the distinguishers keep track of all of their past primitive and construction queries, and never make a query for which they already know the answer according to their query history. We will re-use some notation from the proof of Theorem 1. Notably, we will use the graph representation of the adversarial construction queries, where the nodes of the graph are denoted by $[m, E]$, and $S[m, E]$ denotes the state of a node reached after absorbing E in the construction oracle $\mathcal{O}_{C_m}$.

Step 1: Transform the Adversary into a Distinguisher. From $\mathcal{A}$, we build a distinguisher $\mathcal{D}_1$ which replaces forgery queries by construction queries at the end of the interaction. If one of the attempts by $\mathcal{A}$ results in a successful forgery, $\mathcal{D}_1$ returns 0, otherwise 1. It is worth noting that we have constructed a distinguisher compatible with the distinguishing game from Fig. 2. However, $\mathcal{D}_1$ does not satisfy constraints 1 and 2. We have

$$\mathbf{Pr}\left(\mathcal{D}_1^{W_R} = 0\right) = \mathrm{Adv}^{\mathrm{LR}}_{\mathrm{Kirby}}\left(\mathcal{A}\right) .$$

The extra construction queries made by $\mathcal{D}_1$ to check whether $\mathcal{A}$ made a forgery do not increment the block cost of the construction queries, as these forgery attempts must correspond to intermediate states that were computed beforehand. Therefore, $\mathcal{D}_1$ makes at most N permutation queries and construction queries with at most M blocks.

Step 2: Make the Distinguisher Satisfy Condition 1. Now, from $\mathcal{D}_1$, we build another distinguisher $\mathcal{D}_2$ that aims to satisfy condition 1. $\mathcal{D}_2$ acts as follow:

- $\mathcal{D}_2$ relays faithfully every permutation query to $\mathcal{D}_1$;
- For every construction query from $\mathcal{D}_1$ with input (m, E'), if there exists already a construction query (m, E) such that $E \prec E'$, then $\mathcal{D}_2$ replaces the construction query by the appropriate permutation queries that allows to compute $S[m, E']$ from $S[m, E]$. Otherwise, $\mathcal{D}_2$ relays faithfully the construction query.

Remember that $\mathcal{A}$, thus $\mathcal{D}_2$, interacts with the real world. Therefore, this change only consists of replacing some construction queries by permutation queries. Therefore,

$$\mathbf{Pr}\left(\mathcal{D}_1^{W_R} = 0\right) = \mathbf{Pr}\left(\mathcal{D}_2^{W_R} = 0\right) .$$

$\mathcal{D}_2$ makes at most $N + M$ permutation queries, and construction queries with at most M blocks.

Step 3: Be Generous. Now, from $\mathcal{D}_2$, we build another distinguisher $\mathcal{D}_3$ that queries any intermediate state that is sandwiched in-between two construction queries. More precisely,

- $\mathcal{D}_3$ relays faithfully every primitive and construction query to $\mathcal{D}_2$;
- Additionally, before every construction query with input (m, E), if there exists $(m, E') \in \mathbb{E}$ such that $E \prec E'$[2], then $\mathcal{D}_3$ makes all of the intermediate construction queries between node $[m, E]$ and $[m, E']$, starting from the lowest-level nodes in the graph, finishing thus with node $[m, E]$.

This distinguisher does not violate condition 1, as the extra step never makes the distinguisher query a state of a parent node before the one of its child node. Note that the resources of $\mathcal{D}_3$ do not change over the ones of $\mathcal{D}_2$, as the extra construction queries made have already been captured in the construction query cost. Doing these extra queries give more information to the distinguisher, but this does not change the forgery success probability. Importantly, these supplementary intermediate states queried are not considered as challengeable by the oracle `Forge` at the moment when the queries are made. We have

$$\mathbf{Pr}\left(\mathcal{D}_2^{W_R} = 0\right) = \mathbf{Pr}\left(\mathcal{D}_3^{W_R} = 0\right) .$$

Step 4: Remove Trivial Permutation Queries. From $\mathcal{D}_3$, we build a distinguisher $\mathcal{D}_4$ that replaces the trivial permutation queries with the associated construction queries. In more detail,

- On a forward permutation query with input X, $\mathcal{D}_4$ checks whether there exist $(m, \mathrm{par}(E)), (m, E) \in \mathbb{E}$ such that $X = S[m, \mathrm{par}(E)] \oplus E_{\mathrm{last}}$. If this condition is satisfied, $\mathcal{D}_4$ outputs to $\mathcal{D}_3$ $S[m, E] \oplus S[m, \mathrm{par}(E)] \oplus E_{\mathrm{last}}$. If the condition is not satisfied, $\mathcal{D}_4$ faithfully relays the query.
- On an inverse permutation query with input X, $\mathcal{D}_4$ checks whether there exist $(m, \mathrm{par}(E)), (m, E) \in \mathbb{E}$ such that $Y = S[m, E] \oplus S[m, \mathrm{par}(E)] \oplus E_{\mathrm{last}}$. If this condition is satisfied, $\mathcal{D}_4$ outputs to $\mathcal{D}_3$ $S[m, \mathrm{par}(E)] \oplus E_{\mathrm{last}}$. Otherwise, $\mathcal{D}_4$ faithfully relays the query.

This transformation only replaces permutation queries that could have been determined from the construction query history. Therefore, the resources of $\mathcal{D}_4$ are not more important than the ones of $\mathcal{D}_3$, and we have

$$\mathbf{Pr}\left(\mathcal{D}_4^{W_R} = 0\right) = \mathbf{Pr}\left(\mathcal{D}_3^{W_R} = 0\right) .$$

Step 3 forces to fill the gaps between two construction queries where one is prefix of the other, and step 4 forbids permutation queries for which the answer is already known. As a consequence, this distinguisher will never make permutation queries that aims to complete a path between two nodes $[m, E]$ and $[m, E']$, where $E \prec E'$. Therefore, $\mathcal{D}_4$ satisfies both conditions 1 and 2.

[2]The other direction is forbidden thanks to the previous reduction.

Final Step: Leverage Theorem 1. By using the triangular inequality, and Theorem 1 with $\mathcal{D}_4$, we have

$$\begin{aligned}
\mathbf{Pr}\left(\mathcal{D}_4^{W_R}=0\right) &\le \mathbf{Pr}\left(\mathcal{D}_4^{W_I}=0\right)+\left|\mathbf{Pr}\left(\mathcal{D}_4^{W_R}=0\right)-\mathbf{Pr}\left(\mathcal{D}_4^{W_I}=0\right)\right| \\
&\le \mathbf{Pr}\left(\mathcal{D}_4^{W_I}=0\right)+\mathrm{Adv}(W_R,W_I)(\mathcal{D}_4) \\
&\le \mathbf{Pr}\left(\mathcal{D}_4^{W_I}=0\right)+\frac{3M(M-1)}{2^b}+\frac{(N+M)M}{2^b}+ \\
&\quad \frac{\sum_{i=1}^{s}\mu_i(\mu_i-1)}{2\times 2^{\mathcal{H}_{\mathrm{col}}(\mathcal{D}_{\mathrm{key}})}}+\frac{(N+M)\max_i \mu_i}{2^{\mathcal{H}_{\min}(\mathcal{D}_{\mathrm{key}})}}\,.
\end{aligned}$$

The forgery queries made by $\mathcal{A}$ at the end of the interaction correspond to intermediate states that do not have any prefix in the set of queried strings by $\mathcal{D}_4$. As a consequence, the reduction from $\mathcal{D}$ to $\mathcal{D}_4$ never turns these verification construction queries into permutation queries. Therefore, $\mathbf{Pr}\left(\mathcal{D}_4^{W_I}=0\right)$ is the probability that the adversary succeeds a forgery for a state that is sampled uniformly at random. Finally, noting that one forgery attempt only targets one state at a time, we obtain

$$\mathbf{Pr}\left(\mathcal{D}_4^{W_I}=0\right)\le\frac{t}{2^b-t}\le\frac{2t}{2^b}\,,$$

where we used that $t\le 2^{b-1}$. Wrapping up, we have

$$\begin{aligned}
\mathrm{Adv}_{\mathrm{Kirby}}^{\mathrm{LR}}(N,M,t,\boldsymbol{\mu},\mathcal{D}_{\mathrm{key}}) \le \frac{2t}{2^b}+\frac{3M(M-1)}{2^b}+\frac{(N+M)M}{2^b}+ \\
\frac{\sum_{i=1}^{s}\mu_i(\mu_i-1)}{2\times 2^{\mathcal{H}_{\mathrm{col}}(\mathcal{D}_{\mathrm{key}})}}+\frac{(N+M)\max_i \mu_i}{2^{\mathcal{H}_{\min}(\mathcal{D}_{\mathrm{key}})}}\,,
\end{aligned}$$

and this concludes the proof. □ □

8 Building a Deck Function from Kirby

In this section we define a mode 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 uses two mappings that we specify in the following sections. In Section 8.1 we specify an injective mapping that encodes a sequence of arbitrary-length strings to single string. In Section 8.2 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 8.3 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 $\mathrm{encByte}(x)$ and the encoding of an integer x in the range $[\![0,2^{8B}-1]\!]$ in a B-byte block $\mathrm{encBlock}_B(x)$.

8.1 Injective Mapping from String Sequences to a Single String

The encoding converts a non-empty sequence of an arbitrary number of strings, each of arbitrary length, into a single string. It does this by concatenating strings, with each string 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 $\dots b_{-2}b_{-1}b_0$. 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 applied recursively.

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 $D$
```

8.2 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.

8.3 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, i.e., the counter block. The Kirby state after absorbing the string blocks of E can be cached and then the output sequence can be computed in parallel 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 $\mathsf{Prefix}(D, b, \mathrm{cnt})$

```
**Input**: string $D$, the permutation width $b$ and counter value $cnt \in \mathbb{N} \cup \{0\}$
**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
$E_{i+1} \leftarrow \mathrm{encBlock}_B(cnt)||10^{\ell-1}$
```

Algorithm 5 Kirby-DECK(M, b, n)

```
**Input**: sequence of string $M = M_0, M_1, \dots, M_{m-1}$, the permutation width $b$ 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
    $E \leftarrow \mathsf{Prefix}(D, b, \mathrm{cnt})$
    $Z = Z||\mathrm{Kirby}(E)$
    $\mathrm{cnt} \leftarrow \mathrm{cnt} + 1$
end while
Truncate $Z$ to its first $n$ bytes
return $Z$
```

References

[ABJ+22] Najwa Aaraj, Emanuele Bellini, Ravindra Jejurikar, Marc Manzano, Raghvendra Rohit, and Eugenio Salazar. Farasha: A provable permutation-based parallelizable PRF. *IACR Cryptol. ePrint Arch.*, page 1150, 2022. URL: https://eprint.iacr.org/2022/1150.

[B+08] Daniel J Bernstein et al. Chacha, a variant of salsa20. In *Workshop record of SASC*, volume 8, pages 3–5. Citeseer, 2008.

[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.

[Ber11] Daniel J Bernstein. Extending the salsa20 nonce. In *Workshop record of Symmetric Key Encryption Workshop*, volume 2011, 2011.

[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.

[DM19] Christoph Dobraunig and Bart Mennink. Security of the suffix keyed sponge. *IACR Trans. Symmetric Cryptol.*, 2019(4):223–248, 2019. URL: https://doi.org/10.13154/tosc.v2019.i4.223-248, doi:10.13154/TOSC.V2019.I4.223-248.

[DM24] Christoph Dobraunig and Bart Mennink. Generalized initialization of the duplex construction. *Applied Cryptography and Network Security (ACNS)*, 2024. To appear.

[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.

[JÖS12] Dimitar Jetchev, Onur Özen, and Martijn Stam. Understanding adaptivity: Random systems revisited. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 313–330. Springer, 2012. doi:10.1007/978-3-642-34961-4_20.

[Mau02] Ueli M. Maurer. Indistinguishability of random systems. In Lars R. Knudsen, editor, *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, volume 2332 of *Lecture Notes in Computer Science*, pages 110–132. Springer, 2002. doi:10.1007/3-540-46035-7_8.

[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.