IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2025, No. 1, pp. 97–137. DOI:10.46586/tosc.v2025.i1.97-137

To Pad or Not to Pad? Padding-Free Arithmetization-Oriented Sponges

Charlotte Lefevre, Mario Marhuenda Beltrán and Bart Mennink

Radboud University, Nijmegen, The Netherlands
charlotte.lefevre@ru.nl, m.marhuenda@cs.ru.nl, b.mennink@cs.ru.nl

Abstract. The sponge is a popular construction for hashing and keyed hashing, and the duplex for authenticated encryption. They are proven to achieve approximately $2^{c/2}$ security, where c is the so-called capacity. This approach generalizes to arithmetization-oriented constructions, that operate on elements from a finite field of size p: in this case, security is guaranteed up to $p^{c/2}$. However, to hash securely, the sponge needs to injectively pad the message, and likewise, authenticated encryption schemes often flip bits in the inner part to ensure domain separation. While these bit manipulations have little (but non-zero) influence on the efficiency and security in case of a field of size 2, they become more profound for larger fields. For example, Reinforced Concrete operates on a field with $p \approx 2^{256}$, absorbs 2 elements per permutation evaluation, and has a capacity $c = 1$. Consequently, injective padding results in superfluous permutation evaluations half of the time, and domain separation in the inner part would reduce the capacity to 0 and thus void security. In this work, we investigate an alternative approach to padding and domain separation for the sponge through the use of non-cryptographic permutations (NCPs) to transform the inner state. The idea dates back to the Merkle-Damgård with permutation construction (ASIACRYPT 2007) but we use it in a much more generalized form in the sponge and in the duplex. We demonstrate that this approach allows for NCP-based padding and NCP-based domain separation at a constant loss, regardless of the size of the field. We apply our findings to arithmetization-oriented element-wise sponging (akin to the recently introduced SAFE) and authenticated encryption.

Keywords: sponge · field elements · padding · SAFE · indifferentiability

1 Introduction

The sponge hash function construction of Bertoni et al. [BDPV07] has been the inspiration of dozens of symmetric cryptographic modes for hashing and authenticated encryption in the last decades. It operates on a b-bit state, which is split into an outer part of size r (the *rate*) and inner part of size c (the *capacity*). Internally, it uses a cryptographic b-bit permutation $\mathcal{P}$. To hash a message M, it is first *injectively padded* to a string of size a multiple of r bits, subsequently cut into r-bit blocks, and these blocks are then added to the outer part of the state, interleaved by an evaluation of $\mathcal{P}$ on the state. After the last message block is absorbed, the digest is squeezed from the outer part of the state in r-bit blocks, again interleaved by evaluations of $\mathcal{P}$. If we assume that $\mathcal{P}$ is a random permutation, this construction is proven to behave like a random oracle in the indifferentiability framework [MRH04, CDMP05] as long as the total number of permutation evaluations is at most $2^{c/2}$ [BDPV08]. Naito and Ohta proved that the same bound holds, even if the initial absorption is $r + c/2$ bits, and squeezing is performed at $r + c/2 - \log_2(c)$ bits at a time [NO14]. A depiction of the sponge and this optimized version is given in Figure 1.

Received: 2024-11-22 Accepted: 2025-01-23 Published: 2025-03-07

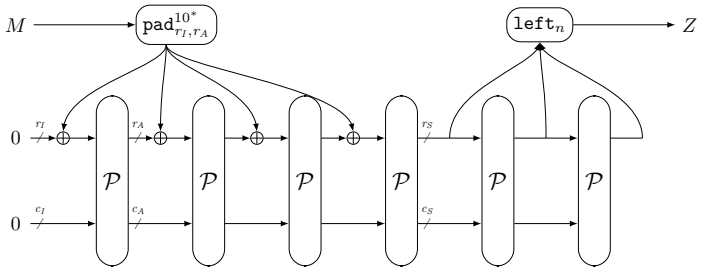

Figure 1: Sponge construction, with a requested output of n bits. Here, the function $\mathtt{pad}^{10^*}_{r_I,r_A}$ appends to M a 1, followed by enough 0s so that the final message block is of full length. The default sponge has $c_I = c_A = c_S$, whereas the optimized version has $c_I = c_A/2$ and $c_S = c_A/2 + \log_2(c_A)$.

The sponge construction is the core behind the Keccak hash function [BDPV11b], later adopted to the SHA-3 hash function standard [Nat15]. This standard operates on a state of $b = 1600$ bits, and has a capacity ranging from $c = 448$ (with rate $r = 1152$) to $c = 1024$ (with rate $r = 576$). Another famous hash function based on the sponge is Ascon-Hash, from the Ascon suite [DEMS21], recently selected for standardization as lightweight cryptographic hashing and authenticated encryption scheme [NIS19]. This upcoming standard operates on a state of $b = 320$ bits, with a capacity ranging from $c = 192$ to $c = 256$.

1.1 The Cost of Padding

In these cases, the message expansion incurred by the padding is not so much of an efficiency penalty. Indeed, the minimum injective padding, or 10^*-padding, adds a single 1 and a sufficient number of 0s to that the message M is of size a multiple of r bits. For a sponge evaluation, this incurs an extra permutation evaluation *only if* the unpadded M turns out to have a size divisible by r. Even if a slightly more involved padding is employed (e.g., the SHA-3 hash function standard [Nat15] first pads M with 01 to separate plain hashing from XOFing, and then adopts the 10^*1-padding), the expected amount of permutation calls incurred purely due to the padding is negligible. In other words, padding is not the bottleneck in bit-oriented cryptographic schemes.

We see a comparable phenomenon in keyed applications of the sponge. For example, in sponge-based authenticated encryption schemes such as SpongeWrap [BDPV11a] or Ascon [DEMS21], a bit is XORed to the inner part of the state for separating the absorption of associated data from the encryption of plaintext. Likewise, in Ascon-PRF [DEMS24], a bit is XORed to the inner part of the state right before squeezing, to separate absorption from tag generation and therewith thwart a certain type of length extension attack (refer to Mennink [Men23, Section 8.3]). These bit manipulations have limited negative affect on the generic security bound.

However, the situation drastically changes when we consider schemes that are evaluated on prime field elements rather than bits, so-called arithmetization-oriented or finite-field friendly schemes. The rise of these applications in the last years, notably in zero-knowledge proof systems [FS86], fully homomorphic encryption [Gen09], and multiparty computation in the head (MPCiTH) [KKW18], has lead to a large amount of cryptographic hash functions, such as Poseidon [GKR+21], Poseidon2 [GKS23], Anemoi and Jive [BBC+23], MiMC [AGR+16], Reinforced Concrete [GKL+22], XHash [ABK+23], Tip5 [SLS+23], and Monolith [GKL+24]. Some of these designs propose permutations that operate over a large field, $\mathbb{F}_p$ for a large p, and are instantiations of the sponge.

As the security proof of the sponge construction is field-agnostic, the bound carries

over mutatis mutandis. Notably: if the permutation does not operate on $\mathbb{F}_2$ but rather $\mathbb{F}_p$, the construction is secure as long as the number of permutation evaluations is at most $p^{c/2}$. Naturally, in field-oriented designs, b, c, and r are smaller. For example, Poseidon [GKR+21] takes $p \geq 2^{32}$ (requiring $c = 8$ to achieve approximately 128-bit security), XHash [ABK+23] takes $p \approx 2^{64}$, and Reinforced Concrete [GKL+22] even operates with $p \approx 2^{256}$ and operates on a state of $b = 3$ elements, split into a capacity of $c = 1$ and a rate of $r = 2$. If such a function is used for general-purpose hashing, a superfluous permutation evaluation just to process the padding has to be made for half of the messages on average. This can be quite costly [HBHW23].

1.2 The Rise of SAFE

To solve the issue of padding, and various other issues in arithmetization-oriented hashing, Aumasson et al. [AKMQ23] proposed SAFE: Sponge API for Field Elements. SAFE is a generic API for sponge functions specifically tailored towards its use on field elements. In a nutshell, SAFE requires the user to first *hash* the input-output pattern IO of the sponge evaluation into a digest, that can be placed in the inner part of the initial state, and subsequently it allows for element-wise absorption and squeezing as long as these operations obey to the initial input-output pattern IO. The hash function evaluation can then additionally hash certain other bit-based data or a domain separator D. This construction was recently proven secure up to $p^{c/2}$ queries by Khovratovich et al. [KBM23], and SAFE has been implemented in the Neptune hash framework and some zero-knowledge proof projects [Set22, MK22]. A comparable construction has recently been introduced by Ashur and Singh Bhati [AB24].

The core idea of the input-output pattern is that, in many applications, such as commitment schemes, Fiat-Shamir protocols, and verifiable encryption of cryptocurrency transactions, the exact input-output pattern is known and fixed in advance, and even stays constant for many evaluations. By thus hashing this pattern into the inner state, one de facto considers sponge hashing with a prefix-free padding, and no 10^*-padding needs to be performed. Of course, for general-purpose applications, using the input-output pattern, or prefix-free padding in general, is not an ideal solution.

1.3 Our Contributions

In this work, we will consider an alternative approach: we will investigate the use of non-cryptographic permutations (NCPs) to indicate end-of-message and to perform domain separation. The approach takes inspiration of Hirose [Hir18], who is itself inspired from the Merkle-Damgård with permutation construction [HPY07]. In a nutshell, Hirose et al. proposed to transform the state of a Merkle-Damgård with an NCP to indicate the end-of-message, and to use two different NCPs depending on whether the unpadded message was full (i.e., of size a multiple of the message block length) or partial (i.e., not full).[1]

We will adopt this idea to the sponge and discuss its potential in full generality, including a discussion on the requirements on the NCPs in various sponge-based schemes. In detail, we will consider three main types of construction in this work:

Hashing (Section 3). We consider the sponge hash function, or more specifically the optimized version with increased initial absorption and increased squeezing, but with an NCP to transform the inner part of the state (different NCPs depending on whether the

[1] One may observe similar appearance of NCPs together with secret primitives. For example, the SUNDAE authenticated encryption scheme [BBLT18] multiplies the state by 2 or 4 depending on whether data is full or partial. Likewise, CMAC [IK03, Dwo05] blinds the final state differently depending on whether data is full or partial.

message is full or partial). This contribution comes closest to the idea of Hirose, but differs in that we apply it to the optimized sponge design. In addition, we consider two variants: `sponge-pi` that allows the initial state to be selected from a restricted set of IVs and `sponge-pi$` that uses an external random oracle to hash certain bit-based input into the initial state (akin to SAFE [AKMQ23] and the work of Grassi and Mennink [GM22]). We demonstrate that security is achieved up to a constant factor 7 loss in the bound, *regardless of the size of the field*. In a bit more detail, we prove that the two constructions are indifferentiable from a random oracle up to approximately $p^{c/2}/7$ queries.

Keyed Sponge (Section 4). We consider the full keyed sponge [BDPV12, MRV15] with an NCP right before squeezing, dubbed `fks-pi`. For this construction we prove that security degrades by a factor 7, *regardless of the size of the field*. In addition, this NCP nicely separates between absorption and tag generation, exactly as Ascon-PRF did.

Duplex (Section 5). We consider the use of NCPs in the duplex construction [BDPV11a]. This generalization of the duplex to a padding-free setting is the by far most complex one, the main reason being that the duplex is general on purpose and individual duplexing calls can have different roles in a bigger construction. Because of this reason, the duplex may potentially require an NCP in every duplexing call, and these NCPs also need to be different. This brings us to two variants `duplex-pi` and `duplex-pi$`, again differing in whether a hash function is used to hash certain bit-based input into the initial state, and we prove that these two achieve indifferentiability security (the same model as the original analysis [BDPV11a] and that of Degabriele et al. [DFG23]) up to approximately $\left(p^c/(\eta^2-\eta+1)\right)^{1/2}$ queries, where η denotes the number of NCPs.

1.4 Comparison

Overall, the use of NCPs does not come for free. In particular, for `sponge-pi` we get a constant factor loss, and for `duplex-pi` with η different NCPs we lose a factor $(\eta^2-\eta+1)$. However, different from simply carrying over the bit-oriented result to finite fields, this loss remains constant *regardless of the size of the field*. This actual loss furthermore highly depends on the actual number of NCPs in use. As we demonstrate in the applications, the `duplex-pi` construction can be used to implement the `sponge-pi` construction with $\eta = 3$ NCPs, as we basically consider three NCPs (the identity, and two different ones to separate between full and partial plaintext). This also matches above-mentioned constant loss 7.

One can consider our results for $p = 2$ to be a generalization of padding into the inner part, and which technically is equivalent to reducing the capacity by 1, which also means a constant factor loss. In this bit-wise setting, it is worth noting that there have been schemes that already adopted this technique to a certain extent. In particular, as we discuss in our applications in Section 7, the ESCH hash function of the NIST lightweight cryptography competition SPARKLE [BBC+19] is a special case of `sponge-pi`, taking bit rotations with a different offset as NCPs. Our result on `sponge-pi`, as a corollary, implies security of the ESCH mode. Likewise, one can implement `SpongeWrap-pi` as an authenticated encryption scheme on top of `duplex-pi` with 3 NCPs (as we also discuss in Section 7).

However, yet, the true power of our schemes only becomes apparent when operating on bigger fields. Indeed, taking for example $p \approx 2^{64}$, one could typically take a sponge construction with $c = 4$ and $r = 1$ to achieve approximately 128-bit security and absorb with one element of approximately 64 bits at a time. With normal padding, each invocation then takes an extra permutation call, and padding into the inner part would reduce the inner part from $c = 4$ to $c = 3$, for which the conventional proofs only guarantee approximately

96-bit security. If we go for the extreme case of Reinforced Concrete (with $b = 3$, $c = 1$, and $r = 2$), padding would result in a superfluous permutation call for half of the messages (as $r = 2$), an element flip in the capacity would turn the existing security bound to $p^{c-1}/2 = \mathcal{O}(1)$ (as $c - 1 = 0$), but our solution still gives a very decent security bound of $p^{c/2}/7$, yielding approximately 128-bit security.

1.5 Outline

We present preliminary material in Section 2. Our finite-field friendly sponges `sponge-pi` and `sponge-pi$` are presented in Section 3, the keyed sponge `fks-pi` in Section 4, and the duplexes `duplex-pi` and `duplex-pi$` in Section 5. The security proofs are gathered in Section 6. Section 7 includes example applications of our constructions, and we conclude in Section 8.

2 Preliminaries

We present basic notation in Section 2.1, the notion of distinguishers in Section 2.2, basic security models in Section 2.3, and we discuss a multicollision result in Section 2.4.

2.1 Notation

We denote $\boldsymbol{A} \sqcup \boldsymbol{B}$ as the disjoint union of sets $\boldsymbol{A}$ and $\boldsymbol{B}$. The empty set is denoted by ϵ. If $\boldsymbol{S}$ is a finite set, we write $x \xleftarrow{\$} \boldsymbol{S}$ to denote that x is sampled uniformly from $\boldsymbol{S}$.

Let $n \in \mathbb{N}$ such that $n > 1$. We will abbreviate the set $\{1, \ldots, n\}$ as $[n]$. Given $k < n$, we denote by $(n)_k$ the falling factorial of n of depth k, i.e. $\prod_{i=0}^{k-1}(n-i)$.

We denote $\mathbb{F}_p^{\leq a} = \bigcup_{i=0}^{a} \mathbb{F}_p^i$, $\mathbb{F}_p^* = \bigcup_{i=0}^{\infty} \mathbb{F}_p^i$, and $\mathbb{F}_p^\infty = \prod_{i=1}^{\infty} \mathbb{F}_p^i$. I.e., $\mathbb{F}_p^*$ is the set of all *finite* tuples of elements of $\mathbb{F}_p$ of any length, and $\mathbb{F}_p^\infty$ the set of all *infinite* tuples of elements of $\mathbb{F}_p$. Note that $\mathbb{F}_p^*$ contains the empty tuple, which we also denote ϵ.

Let $s = (s_1, s_2, \ldots, s_n)$ and $s' = (s'_1, s'_2, \ldots, s'_m) \in \mathbb{F}_p^*$, and $\ell, \ell' \in \mathbb{N}$ with $1 \leq \ell \leq \ell'$. We denote by $s \| s'$ the tuple $(s_1, \ldots, s_n, s'_1, \ldots, s'_m)$. In particular, if $s = (1)$ and s' is of the form 0^α, we abbreviate the concatenation as 10^α. Moreover, if $n = m$, we denote $s \oplus s'$ as the element-wise addition of s and s'. Moreover, $s[\ell : \ell']$ refers to the empty tuple if $\ell > n$, and $(s_\ell, \ldots, s_{\max\{\ell', n\}})$ otherwise. If $\ell \leq n$, we define $\texttt{left}_\ell(s)$ to be $s[1 : \ell]$ and $\texttt{right}_\ell(s)$ to be $s[n - \ell : n]$.

We denote by $\texttt{pad}_{r_I, r_A}$ the function that takes as input a tuple $M \in \mathbb{F}_p^*$, and returns a tuple of field blocks $\overline{M} \in \mathbb{F}_p^{\leq r_I} \times (\mathbb{F}_p^{\leq r_A})^*$ such that only the last block of the tuple may be incomplete. $\texttt{pad}_{r_I, r_A}(\cdot)$ is formally defined in Algorithm 1. In our constructions, this padding function will be used via a more technical function called $\texttt{padBlock}_{r_I, r_A}(\cdot)$. It takes as input a tuple $M \in \mathbb{F}_p^*$, and returns a tuple of field blocks to be absorbed into the state, additionally with a label $d \in \{p, f\}$ helping to choose which NCP to apply. $\texttt{padBlock}_{r_I, r_A}(\cdot)$ is also defined in Algorithm 1. Additionally, we will overload the notation by denoting $\texttt{pad}_r(\cdot) = \texttt{pad}_{r,r}(\cdot)$ and $\texttt{padBlock}_r(\cdot) = \texttt{padBlock}_{r,r}(\cdot)$.

Let $\texttt{RO}^\infty : \mathbb{F}_p^* \to \mathbb{F}_p^\infty$ denote a random oracle in the sense of Bellare and Rogaway [BR93]. From $\texttt{RO}^\infty$ we build the function $\texttt{RO} : \mathbb{F}_p^* \times \mathbb{N} \to \mathbb{F}_p^*$, which, on input (M, n), returns $\texttt{RO}^\infty(M)[1 : n]$. We will abuse notation and refer to $\texttt{RO}$ as a random oracle.

2.2 Distinguishing Advantage

We will consider security of our modes in a distinguishing setup. Here, we consider a distinguisher $\texttt{D}$ that is given access to some oracle $\texttt{O} \in \{\texttt{W}_\texttt{R}, \texttt{W}_\texttt{I}\}$, denoted $\texttt{D}^\texttt{O}$, and that

Algorithm 1 Definition of pad and padBlock

Function $\mathtt{pad}_{r_I,r_A}$
Input: $M \in \mathbb{F}_p^*$
Output: $\overline{M} \in \mathbb{F}_p^{\leq r_I} \times (\mathbb{F}_p^{\leq r_A})^*$
1: $\ell \leftarrow 1 + \lceil (|M| - r_I)/r_A \rceil$
2: $M_1 \leftarrow M[1 : r_I]$
3: **for** $1 \leq i \leq \ell - 1$ **do**
4: $\quad M_{i+1} \leftarrow M[r_I + (i-1)r_A + 1 : r_I + ir_A]$
5: **end for**
6: **return** $(M_1, \ldots, M_\ell)$

Function $\mathtt{padBlock}_{r_I,r_A}$
Input: $M \in \mathbb{F}_p^*$
Output: $(\overline{M}, d) \in (\mathbb{F}_p^{r_I} \times (\mathbb{F}_p^{r_A})^*) \times \{p, f\}$
1: **if** $|M| < r_I$ **then**
2: $\quad M \leftarrow M \| 10^{-|M|-1 \bmod r_I}$
3: $\quad d \leftarrow p$
4: **else if** $|M| - r_I \bmod r_A > 0$ **then**
5: $\quad M \leftarrow M \| 10^{-(|M|-r_I)-1 \bmod r_A}$
6: $\quad d \leftarrow p$
7: **else**
8: $\quad d \leftarrow f$
9: **end if**
10: **return** $(\mathtt{pad}_{r_I,r_A}(M), d)$

outputs a decision bit $b \in \{0, 1\}$. We define

$$\Delta_{\mathtt{D}}(\mathtt{W_R}\,;\,\mathtt{W_I}) = \left|\Pr\left[\mathtt{D}^{\mathtt{W_R}} \Rightarrow 1\right] - \Pr\left[\mathtt{D}^{\mathtt{W_I}} \Rightarrow 1\right]\right| .$$

In our work, the oracle ($\mathtt{W_R}$ or $\mathtt{W_I}$) will be randomized, D has unbounded computational power and its complexity is solely measured by the number of queries it makes to the oracle (to be defined shortly). The interaction that D has with its oracle is summarized in a transcript τ. We denote by $D_{\mathtt{W_R}}$ (resp., $D_{\mathtt{W_I}}$) the probability distribution of transcripts when interacting with $\mathtt{W_R}$ (resp., $\mathtt{W_I}$). We say that a transcript τ is attainable if $\Pr[D_{\mathtt{W_I}} = \tau] > 0$, and denote the set of all attainable transcripts by $\mathcal{T}$.

In the proof, we will use Patarin's H-Coefficient technique [Pat08, CS14], that allows us to bound $\Delta_{\mathtt{D}}(\mathtt{W_R}\,;\,\mathtt{W_I})$ by making a clever separation of transcripts into good and bad ones, and analyzing the distance between the two worlds for good transcripts and the probability that a bad transcript occurs for $\mathtt{W_I}$.

Lemma 1 (H-Coefficient Technique)**.** *Consider a fixed deterministic adversary* D*. Let* $\mathcal{T} = \mathcal{T}_{good} \sqcup \mathcal{T}_{bad}$ *be a partition of* $\mathcal{T}$ *into good and bad transcripts. Suppose that*

$$\frac{\Pr[D_{\mathtt{W_R}} = \tau]}{\Pr[D_{\mathtt{W_I}} = \tau]} \geq 1 - \varepsilon, \quad \forall \tau \in \mathcal{T}_{good}\,,$$
$$\Pr[D_{\mathtt{W_I}} \in \mathcal{T}_{bad}] \leq \delta\,.$$

Then, we have:

$$\Delta_{\mathtt{D}}(X\,;\,Y) \leq \varepsilon + \delta\,.$$

2.3 Security Models

We will consider two main security models, namely PRF security and indifferentiability.

Indifferentiability. For the keyless constructions in this paper, we use the indifferentiability framework, first introduced by Maurer et al. [MRH04] and refined to the context of hash functions by Coron et al. [CDMP05].

Consider a construction C, relying on an ideal primitive P: $\mathtt{C}^{\mathtt{P}} : \mathbf{S}^* \to \mathbf{S}^*$. Consider a simulator S with the same interface as P. Indifferentiability considers the advantage of a distinguisher D in distinguishing the real world $\mathtt{W_R} = (\mathtt{C}^{\mathtt{P}}, \mathtt{P})$ from the ideal world

$\mathtt{W_I} = (\mathtt{RO}, \mathtt{S}^{\mathtt{RO}})$. In detail the advantage of a distinguisher $\mathtt{D}$ in differentiating $\mathtt{C}$ from a random oracle $\mathtt{RO}$ with respect to a simulator $\mathtt{S}$ is defined as

$$\mathbf{Adv}^{\mathrm{iff}}_{\mathtt{C},\mathtt{S}}(\mathtt{D}) = \Delta_{\mathtt{D}}\left(\mathtt{C}^{\mathtt{P}}, \mathtt{P}\,;\, \mathtt{RO}, \mathtt{S}^{\mathtt{RO}}\right) . \tag{1}$$

Without loss of generality, we assume that $\mathtt{D}$ does not make redundant queries, i.e., queries it already knows the answer to.

PRF Security. For the keyed constructions, we consider multi-user PRF security. Consider a construction $\mathtt{C}$, relying on an ideal primitive $\mathtt{P}$ and keyed from a keyspace $\mathbf{K}$: $\mathtt{C}^{\mathtt{P}} : \mathbf{K} \times \mathbf{S}^* \to \mathbf{S}^*$. Assume we consider μ users, all with independent random keys, and denote by $\mathtt{C}^{\mathtt{P}}_{K_{id}}$ the instance of user $id \in [\mu]$. Multi-user PRF security considers the advantage of a distinguisher $\mathtt{D}$ in distinguishing the real world $\mathtt{W_R} = \left((\mathtt{C}^{\mathtt{P}}_{K_{id}})_{id\in[\mu]}, \mathtt{P}\right)$ from the ideal world $\mathtt{W_I} = \left((\mathtt{RO}_{id})_{id\in[\mu]}, \mathtt{P}\right)$:

$$\mathbf{Adv}^{\mu\text{-prf}}_{\mathtt{C}}(\mathtt{D}) = \Delta_{\mathtt{D}}\left((\mathtt{C}^{\mathtt{P}}_{K_{id}})_{id\in[\mu]}, \mathtt{P}\,;\, (\mathtt{RO}_{id})_{id\in[\mu]}, \mathtt{P}\right) . \tag{2}$$

Without loss of generality, we assume that $\mathtt{D}$ does not make redundant queries, i.e., queries it already knows the answer to.

2.4 Multicollision Result

We will use the following multicollision result of Choi et al. [CLL19, page 187], and later slightly improved by Lefevre and Mennink [LM24].

Lemma 2. *Let $q, R \in \mathbb{N}$. Consider the experiment of throwing uniformly at random q balls in R bins. For $u \in [R]$, denote by S_u the size of the bin number u. Then,*

$$\mathbb{E}\left[\max_{u\in[R]} S_u\right] \le \frac{2q}{R} + 3\ln(R) + 4\,.$$

The proof of this lemma can be found in [LM24].

3 Arithmetization-Oriented Sponges

We describe our two functions `sponge-pi` and `sponge-pi$` in Section 3.1, and discuss their security in Section 3.2.

3.1 Specification of `sponge-pi` and `sponge-pi$`

In a nutshell, `sponge-pi` and `sponge-pi$` are both based on the sponge construction [BDPV07], but are (i) optimized in the absorption rate at initialization and in the squeezing rate as in Naito and Ohta [NO14], and (ii) adopting Hirose's trick [Hir18] by transforming the inner state before squeezing using an NCP. Both constructions separate between full and partial message, by the use of the NCP. The difference between the two constructions is in the initialization: in `sponge-pi`, the initial value is selected in advance from a family of μ pre-determined initial values, whereas in `sponge-pi$`, they are outputs of a random oracle, comparable to how it was done in [AKMQ23, GM22].

Setup. Let $b, r_I, c_I, r_A, c_A, r_S, c_S \in \mathbb{N}$ such that $b = r_I + c_I = r_A + c_A = r_S + c_S$ and $c_I \le c_A$. Both constructions operate on top of a cryptographic permutation $\mathcal{P} : \mathbb{F}_p^b \to \mathbb{F}_p^b$. The `sponge-pi` construction is defined for a predetermined set of initial values $(IV_{id})_{id\in[\mu]}$, whereas `sponge-pi$` operates on a hash function $\mathcal{H} : \mathcal{D} \to \mathbb{F}_p^{c_I}$ such that $[\mu] \subseteq \mathcal{D}$.

Algorithm 2 `sponge-pi` and `sponge-pi$`

Function `CoreSponge`
Input: $(IV, M, n) \in \mathbb{F}_p^b \times \mathbb{F}_p^* \times \mathbb{N}_+$
Output: $Z \in \mathbb{F}_p^*$
1: $S \leftarrow IV$
2: $Z \leftarrow \epsilon$
3: $(M_1, \ldots, M_\ell)$, $d \leftarrow \texttt{padBlock}_{r_I, r_A}(M)$
4: **for** $1 \le i \le \ell - 1$ **do**
5: $\quad S \leftarrow \mathcal{P}\left(S \oplus (M_i \| 0^*)\right)$
6: **end for**
7: $S \leftarrow \mathcal{P}\left(\pi^d\left(S \oplus (M_\ell \| 0^*)\right)\right)$
8: **for** $1 \le i \le \lceil \frac{n}{r} \rceil$ **do**
9: $\quad Z \leftarrow Z \| \texttt{left}_r(S)$
10: $\quad S \leftarrow \mathcal{P}(S)$
11: **end for**
12: **return** $Z[1:n]$

Function `sponge-pi`
Input: $(id, M, n) \in [\mu] \times \mathbb{F}_p^* \times \mathbb{N}_+$
Output: $Z \in \mathbb{F}_p^*$
1: $IV \leftarrow 0^{r_I} \| IV_{id}$
2: **return** `CoreSponge`(IV, M, n)

Function `sponge-pi$`
Input: $(id, M, n) \in [\mu] \times \mathbb{F}_p^* \times \mathbb{N}_+$
Output: $Z \in \mathbb{F}_p^*$
1: $IV \leftarrow 0^{r_I} \| \mathcal{H}(id)$
2: **return** `CoreSponge`(IV, M, n)

Consider two NCPs:

$$\pi^p : \mathbb{F}_p^{c_I} \to \mathbb{F}_p^{c_I}, \qquad \pi^f : \mathbb{F}_p^{c_I} \to \mathbb{F}_p^{c_I}.$$

By abuse of notation, given $A \in \mathbb{F}_p^*$ and $B \in \mathbb{F}_p^{c_I}$, for $d \in \{p, f\}$ we use $\pi^d(A \| B)$ to refer to $A \| \pi^d(B)$.

The two NCPs are required to satisfy two constraints. The first constraint ensures no collision between the NCPs and no fixed-point for π^f (note that we do not need to avoid a fixed-point for π^p, because partial messages always get padded to at least one full block):

Constraint 1. *For any $x \in \mathbb{F}_p^{c_I}$, we must have*

$$\pi^p(x) \neq \pi^f(x) \text{ and } x \neq \pi^f(x).$$

The second constraint is only relevant in case the user can choose the IVs (i.e., for `sponge-pi`), and requires Constraint 1 to apply even over different IVs:

Constraint 2. *For any distinct $id_1, id_2 \in [\mu]$, we must have*

$$\left\{IV_{id_1}, \pi^f(IV_{id_1}), \pi^p(IV_{id_1})\right\} \cap \left\{IV_{id_2}, \pi^f(IV_{id_2}), \pi^p(IV_{id_2})\right\} = \epsilon.$$

A simple example that satisfies Constraint 1 is to take the identity permutation for π^p and addition by a non-zero constant C on the rightmost c_I field elements for π^f. In the context of `sponge-pi`, if $c_I > 1$, then in order to satisfy Constraint 2, the initial values could be encoded on the leftmost $c_I - 1$ field elements, while the constant C is added only to the rightmost field element. If instead $c_I = 1$ and the IVs are generated dynamically (e.g., via a counter), the procedure for generating IVs can be adapted to ensure that any new initial value IV must differ from a prior IV', $IV' + C$, and $IV' - C$. On the other hand, if a predefined, arbitrary, list of unique IVs is provided independently of the construction, then selecting NCPs to meet Constraint 2 is challenging, and in that case, using the `sponge-pi$` variant may be more practical.

Constructions. The two constructions are described in Algorithm 2 and illustrated in Figure 2.

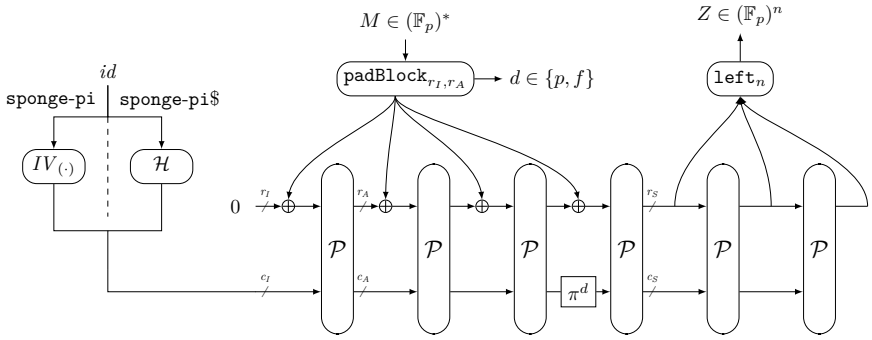

Figure 2: `sponge-pi` and `sponge-pi$` constructions, with a requested output of n field elements. The two functions differ in the selection of the IV. For `sponge-pi`, on input of $id \in [\mu]$, the function "$IV_{(\cdot)}$" selects the initial value IV_{id}, whereas for `sponge-pi$` the initial value is generated by an evaluation of $\mathcal{H}$.

3.2 Security of `sponge-pi` and `sponge-pi$`

We will prove the security of both constructions. Here, we will quantify the complexity of the distinguisher by the number of permutation evaluations required to perform all computations in the real world. This quantity is denoted by $Q_{\mathcal{P}}$. Moreover, with the construction `sponge-pi$`, we denote by $Q_{\mathcal{H}}$ the number of queries made to the hash function $\mathcal{H}$ on the input domain $[\mu]$. We have the following results:

Theorem 1. *Let $Q_{\mathcal{P}} \in \mathbb{N}$ and $(IV_{id})_{id\in[\mu]} \subset \mathbb{F}_p^{c_I}$. Let π^p and π^f be two NCPs satisfying Constraints 1 and 2. Let* `C` *denote the* `sponge-pi` *construction based on a random permutation $\mathcal{P}$. There exists a simulator* `S` *such that, for any distinguisher* `D` *making at most $Q_{\mathcal{P}}$ permutation evaluations,*

$$\boldsymbol{Adv}^{\text{iff}}_{\mathtt{C},\mathtt{S}}(\mathtt{D}) \leq \frac{7Q_{\mathcal{P}}(Q_{\mathcal{P}}-1)}{p^{c_A}} + \frac{7\mu Q_{\mathcal{P}}}{2p^{c_I}} + \frac{3{Q_{\mathcal{P}}}^2}{p^b} + \frac{(3r_S\ln(p)+4)\,Q_{\mathcal{P}}}{p^{c_S}}\,.$$

Theorem 2. *Let $Q_{\mathcal{P}}, Q_{\mathcal{H}} \in \mathbb{N}$. Let π^p and π^f be two NCPs satisfying Constraint 1. Let* `C$` *denote the* `sponge-pi$` *construction based on a random oracle $\mathcal{H}$ and a random permutation $\mathcal{P}$. There exists a simulator* `S` *such that, for any distinguisher* `D` *making at most $Q_{\mathcal{P}}$ permutation evaluations and $Q_{\mathcal{H}}$ random oracle evaluations on the input domain $[\mu]$,*

$$\boldsymbol{Adv}^{\text{iff}}_{\mathtt{C\$},\mathtt{S}}(\mathtt{D}) \leq \frac{7Q_{\mathcal{P}}(Q_{\mathcal{P}}-1)}{p^{c_A}} + \frac{22Q_{\mathcal{P}}Q_{\mathcal{H}}}{p^{c_I}} + \frac{7Q_{\mathcal{H}}(Q_{\mathcal{H}}-1)}{2p^{c_I}} + \frac{3{Q_{\mathcal{P}}}^2}{p^b} + \frac{(3r_S\ln(p)+4)\,Q_{\mathcal{P}}}{p^{c_S}}\,.$$

Even though we stated security of the two constructions in separate theorems (to make the conditions explicit), the proofs are in fact very similar, and we prove Theorems 1 and 2 simultaneously in Section 6.1.

Interpretation of the Bounds. Ignoring logarithmic factors, the bounds are of the form

$$\mathbf{Adv}^{\text{iff}}_{\text{sponge-pi},\mathtt{S}}(\mathtt{D}) = \mathcal{O}\left(\frac{{Q_{\mathcal{P}}}^2}{p^{c_A}} + \frac{\mu Q_{\mathcal{P}}}{p^{c_I}} + \frac{Q_{\mathcal{P}}}{p^{c_S}}\right),$$
$$\mathbf{Adv}^{\text{iff}}_{\text{sponge-pi\$},\mathtt{S}}(\mathtt{D}) = \mathcal{O}\left(\frac{{Q_{\mathcal{P}}}^2}{p^{c_A}} + \frac{Q_{\mathcal{H}}Q_{\mathcal{P}}}{p^{c_I}} + \frac{\binom{Q_{\mathcal{H}}}{2}}{p^{c_I}} + \frac{Q_{\mathcal{P}}}{p^{c_S}}\right).$$

By putting $\mu = 1$, i.e., allowing a single initial value, the former bound is identical to that of Naito and Ohta [NO14]. The second bound is identical, noting that $Q_{\mathcal{H}} \leq \mu$. The multi-user setting, where the distinguisher may choose from μ initial values, linearly scales the second term for `sponge-pi`, de facto reducing the initial capacity c_I by a logarithmic factor (in $\log_p$). For `sponge-pi$`, this multi-user setting leads to an extra term covering collisions between hash function calls, similar to [AKMQ23, GM22].

In general, to get a balanced bound, one needs $p^{c_I}/\mu \approx p^{c_A}/2$ in the case of `sponge-pi`. Assume we have a field size around 2^{64}, security goal of 2^{128}, and around 2^{64} users. Then, we need $b \geq 5$ and $c_A = 4$, and we can lower c_I to 3 and c_S to around 2. For the case of `sponge-pi$`, if the application accepts any kind of input to $\mathcal{H}$, it is better to take $r_A = r_I$. However, still the squeezing rate can be larger (i.e., $r_S \approx r_A + c_A/2$) without any significant loss in the security.

4 Arithmetization-Oriented Keyed Sponge

We describe our function `fks-pi` in Section 5.1, and discuss its security in Section 5.2.

4.1 Specification of `fks-pi`

The `fks-pi` construction basically turns the full keyed sponge [BDPV12, MRV15] into one that performs padding using NCPs.

Setup. Let $b, r, c \in \mathbb{N}$ such that $b = r + c$ and $k \leq b$. The construction operates on top of a cryptographic permutation $\mathcal{P} : \mathbb{F}_p^b \to \mathbb{F}_p^b$. It is furthermore defined for a predetermined set of initial values $(IV_{id})_{id\in[\mu]}$ from $\mathbb{F}_p^{b-k}$ and an array of keys $(K_{id})_{id\in[\mu]}$ from $\mathbb{F}_p^k$.

As in Section 3, consider two NCPs (cf. Algorithm 3):

$$\pi^p : \mathbb{F}_p^b \to \mathbb{F}_p^b\,, \qquad \pi^f : \mathbb{F}_p^b \to \mathbb{F}_p^b\,.$$

The NCPs are required to satisfy Constraint 3.

Constraint 3. *For any $M, M' \in \mathbb{F}_p^b$, and any distinct*

$$F_\cdot(\cdot),\, G_\cdot(\cdot) \in \{(x,y) \mapsto x + y,\ (x,y) \mapsto \pi^f(x+y),\ (x,y) \mapsto \pi^p(x+y)\}\,,$$

we must have:

$$\Pr\left[F_x(M) = G_{x'}(M')\,; \quad x, x' \xleftarrow{\$} \mathbb{F}_p^b\right] \leq \frac{\xi}{p^b}\,, \tag{3}$$

$$\Pr\left[F_x(M) = G_x(M')\,; \quad x \xleftarrow{\$} \mathbb{F}_p^b\right] \leq \frac{\xi}{p^b}\,, \tag{4}$$

for some small $\xi \in \mathbb{Z}_{\geq 0}$.

To clarify Constraint 3, we give the following example.

Example 1. Fix distinct $C, \widehat{C} \in \mathbb{F}_p^b \setminus \{0, 1\}$, and consider:

$$\pi^p(x) = C \cdot x\,,$$
$$\pi^f(x) = \widehat{C} \cdot x\,.$$

Then, it can be easily verified that π^p and π^f satisfy (3) and (4) for $\xi = 1$.

Algorithm 3 `fks-pi`

Function `fks-pi`
Input: $(id, M, n) \in [\mu] \times \mathbb{F}_p^* \times \mathbb{N}_+$
Output: $Z \in \mathbb{F}_p^*$
1: $S \leftarrow \mathcal{P}(K_{id} \,\|\, IV_{id})$
2: $Z \leftarrow \epsilon$
3: $(M_1, \ldots, M_\ell)$, $d \leftarrow \texttt{padBlock}_b(M)$
4: **for** $1 \le i \le \ell - 1$ **do**
5: $\quad S \leftarrow \mathcal{P}\left(S \oplus M_i\right)$
6: **end for**
7: $S \leftarrow \mathcal{P}\left(\pi^d\left(S \oplus M_\ell\right)\right)$
8: **for** $1 \le i \le \lceil \frac{n}{r} \rceil$ **do**
9: $\quad Z \leftarrow Z \,\|\, \mathbf{left}_r\left(S\right)$
10: $\quad S \leftarrow \mathcal{P}(S)$
11: **end for**
12: **return** $Z[1:n]$

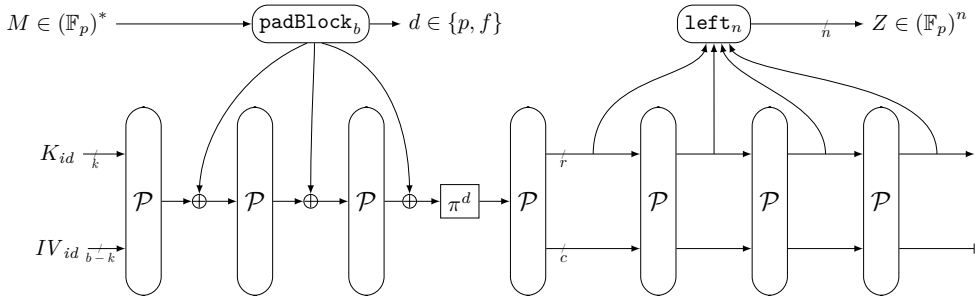

Figure 3: `fks-pi` construction, with a requested output of n field elements.

Construction. The construction is described in Algorithm 3 and illustrated in Figure 3.

4.2 Security of `fks-pi`

In this section, we describe the security of `fks-pi`. We consider a distinguisher `D` as described in Section 2.3. The number of permutation queries made by `D` is denoted by $Q_{\mathcal{P}}$ and the number of construction queries made by `D` is measured by the number of permutation evaluations that would have been required in the real world to compute the construction calls made by the distinguisher. This quantity is denoted by $Q_{\mathcal{C}}$.

Furthermore, To facilitate the description, let us represent $(IV_{id})_{id\in[\mu]}$ with two lists, L_1 and L_2, defined as follows: L_1 contains all elements appearing in $(IV_{id})_{id\in[\mu]}$ without any repetition. L_2, having the same length as L_1, records the *multiplicity* of each element. In other words, the s^{th} element in L_1 appears $L_2[s]$ times in $(IV_{id})_{id\in[\mu]}$. From this, we define:

$$u_s = L_2[s]\,, \qquad u_{\max} = \max_s u_s\,.$$

We have the following theorem:

Theorem 3. *Let $Q_{\mathcal{C}}, Q_{\mathcal{P}} \in \mathbb{N}$, $(IV_{id})_{id\in[\mu]} \in \left(\mathbb{F}_p^{b-k}\right)^\mu$ and let $u_{\max}$ and u_s be as above. Let π^p and π^f be two NCPs satisfying Constraint 3 with parameter ξ. Let* `C` *denote the* `fks-pi` *construction based on a random permutation $\mathcal{P}$. For any distinguisher* `D` *making at*

most $Q_{\mathcal{P}}$ permutation evaluations and $Q_{\mathcal{C}}$ construction queries, we have:

$$\boldsymbol{Adv}_{\mathtt{C}}^{\mu\text{-prf}}(\mathtt{D}) \leq \sum_s \frac{u_s(u_s-1)}{p^{k+1}} + \frac{Q_{\mathcal{P}} u_{\max}}{p^k} + \frac{9\xi {Q_{\mathcal{C}}}^2}{p^b} + \frac{28 Q_{\mathcal{C}} Q_{\mathcal{P}}}{p^b} + \frac{14(6r+8)Q_{\mathcal{P}}}{p^c}\,. \quad (5)$$

The proof of Theorem 3 is given in Section 6.2.

Interpretation of the Bound. In practice $Q_{\mathcal{C}} \ll Q_{\mathcal{P}}$ and ξ is a small constant, this means that at a high level, the bound is of the form:

$$\mathcal{O}\left(\sum_s \frac{u_s(u_s-1)}{p^{k+1}} + \frac{Q_{\mathcal{P}} u_{\max}}{p^k} + \frac{Q_{\mathcal{C}} Q_{\mathcal{P}}}{p^b} + \frac{Q_{\mathcal{P}}}{p^c}\right).$$

Without loss of generality, we ignore constant and logarithmic factors. We first consider the most favorable case for the multi-user setting, i.e., when all users have distinct IVs. Then, `fks-pi` is generically secure as long as $Q_{\mathcal{P}} \ll \min\left\{p^k, p^b/Q_{\mathcal{C}}, p^c\right\}$and $Q_{\mathcal{C}} \ll \min\left\{p^{b/2}, p^c, p^k\right\}$. For instance, if $p \approx 2^{32}$, and assuming that $\mu \ll 2^{32}$ and $Q_{\mathcal{C}} \ll 2^{64}$, then taking b, c, and k as small as respectively 6, 4, and 4 guarantees 128 bits of security, and a throughput of 2 field elements per permutation call. On the other hand, if all users share the same IV, the key size needs to be increased by a factor of $\log_p(\mu)$ in order to maintain 128 bits of security.

5 Arithmetization-Oriented Duplex

We describe our two functions `duplex-pi` and `duplex-pi$` in Section 5.1, and discuss their security in Section 5.2.

5.1 Specification of `duplex-pi` and `duplex-pi$`

The `duplex-pi` and `duplex-pi$` constructions are based on the simple duplex construction [BDPV11a], but employ NCPs per duplexing call in lieu of padding. The case is significantly more subtle than the case of `sponge-pi`, `sponge-pi$`, and `fks-pi`, because duplexing calls can have various different roles depending on the use case, and the NCPs need to account for that. The difference between the two constructions is, just like in Section 3, in the initialization: in `duplex-pi`, the initial value is selected in advance from a family of μ pre-determined initial values, whereas in `sponge-pi$`, they are outputs of a random oracle.

Setup. Let $b, r_I, c_I, r_A, c_A, r_S, c_S \in \mathbb{N}$ such that $b = r_I + c_I = r_A + c_A = r_S + c_S$ and $c_I \leq c_A$. Both constructions operate on top of a cryptographic permutation $\mathcal{P} : \mathbb{F}_p^b \to \mathbb{F}_p^b$. The `duplex-pi` construction is defined for a predetermined set of initial values $(IV_{id})_{id\in[\mu]}$, whereas `duplex-pi$` operates on a hash function $\mathcal{H} : \mathcal{D} \to \mathbb{F}_p^{c_I}$ such that $[\mu] \subseteq \mathcal{D}$.

The duplex constructions differ from the sponge constructions in that every permutation evaluation is a duplex call that absorbs data and squeezes data. Depending on the application, individual duplexing calls may have different roles and we need to accommodate for this (e.g., domain separation) by using more NCPs. In detail, we consider the following family of NCPs:

$$\left(\pi_i^d : \mathbb{F}_p^{c_I} \to \mathbb{F}_p^{c_I}\right)_{i\in[\zeta],\, d\in\{p,f\}}.$$

This family of NCPs is required to satisfy two constraints. In detail, below Constraint 4 applies to both constructions and Constraint 5 applies only to the case where the user can choose the IVs (i.e., for `duplex-pi`).

Constraint 4. *For any $x \in \mathbb{F}_p^{c_I}$ and any distinct $(i_1, d_1), (i_2, d_2) \in [\zeta] \times \{p, f\}$, we must have*

$$\pi_{i_1}^{d_1}(x) \neq \pi_{i_2}^{d_2}(x)\,.$$

A simple example that satisfies Constraint 4 is to encode each (i, d) with a unique constant $c_{i,d}$ and then take $\pi_i^d(x) = x \oplus c_{i,d}$.

Constraint 5. *For any distinct $id_1, id_2 \in [\mu]$ and any (not necessarily distinct) (i_1, d_1), $(i_2, d_2) \in [\zeta] \times \{p, f\}$, we must have*

$$\pi_{i_1}^{d_1}(IV_{id_1}) \neq \pi_{i_2}^{d_2}(IV_{id_2})\,.$$

Constructions. A duplex is a stateful object, and has two interfaces, "`init`" and "`next`". There have been various suggestions on what exactly would be captured by an `init` (e.g., should the first `next` call be part of an `init` or not?), and how a `next` call should be defined (e.g., should a `next` call start at absorbing plaintext, or at permuting the state?). Mennink gave a treatment on how the phasing of the keyed duplex evolved [Men23, Section 3.4]. However, as we focus on the unkeyed duplex (and we will in fact prove its indifferentiability), we will adopt the convention that a `next` call consists of absorbing-permuting-squeezing (as in the original duplex proposal [BDPV11a] but also in the work of Degabriele et al. [DFG23]) and that an `init` call initializes the state *and* does the initial `next` call (which happens at a different rate):

- "`init`" initializes an instance of the duplex object. It takes as input the value *id* to initialize the state (here, the two constructions `duplex-pi` and `duplex-pi$` differ) with the correct IV at its inner part. It additionally takes as input a message block $M \in \mathbb{F}_p^*$ with $|M| \leq r_I$ and a domain separator i. It pads the message block *if needed*, using $\texttt{padBlock}_{r_I}$, absorbs it into the state, applies the appropriate NCP, permutes and returns the leftmost r_S field elements of the state;
- "`next`" duplexes the state. It takes as input a message block $M \in \mathbb{F}_p^*$ with $|M| \leq r_A$ and a domain separator i. It pads the message block *if needed*, using $\texttt{padBlock}_{r_A}$, absorbs it into the state, applies the appropriate NCP, permutes and returns the leftmost r_S field elements of the state.

The scheme can be parallelized over different instances by incorporating a unique identifier u alongside a state that gets initialized. We admit that for every duplexing call a NCP is applied on the inner part of the state. This might seem as a lot of overhead at first sight, but these NCPs can be implemented with constant additions, and the NCP that is expected to be used most often can be set to the identity (in fact, the original duplex construction [BDPV11a] did 10^*-padding for every `next` call). Finally, we remark that, if $M_1, M_2 \in \mathbb{F}_p$, then absorbing first M_1 and then M_2 does not necessarily give the same output as absorbing $M_1 \| M_2$ together. However, this constraint is not so confusing in practice, since typically, before a domain separator is applied, the message blocks are first of full length, and only the last block is potentially not full.

The two constructions are described in Algorithm 4 and illustrated in Figure 4. Here, in the illustration, the `next`-wise padding is integrated in the figure for visibility.

5.2 Security of `duplex-pi` and `duplex-pi$`

We prove security of both constructions in this section. Before doing so, we remark that security of the original duplex was derived from that of the sponge [BDPV11a], and one might wonder whether we can likewise reduce the security of `duplex-pi` to `sponge-pi` (or `duplex-pi$` to `sponge-pi$`). However, our description of `duplex-pi`, and in particular

Algorithm 4 `duplex-pi` and `duplex-pi$`

Function `duplex-pi.init`
Input: $(id, M, i) \in [\mu] \times \mathbb{F}_p^{\leq r_I} \times [\zeta]$
Output: $Z \in \mathbb{F}_p^{r_A}$
1: $IV \leftarrow 0^{r_I} \| IV_{id}$
2: **return** `CoreDuplex.init`(IV, M, i)

Function `duplex-pi.next`
Input: $(M, i) \in \mathbb{F}_p^{\leq r_A} \times [\zeta]$
Output: $Z \in \mathbb{F}_p^{r_A}$
1: **return** `CoreDuplex.next`(M, i)

Function `CoreDuplex.init`
Input: $(IV, M, i) \in \mathbb{F}_p^{b} \times \mathbb{F}_p^{\leq r_I} \times [\zeta]$
Output: $Z \in \mathbb{F}_p^{r_S}$
1: $S \leftarrow IV$
2: $\overline{M}, d \leftarrow \texttt{padBlock}_{r_I}(M)$
3: $S \leftarrow \mathcal{P}\left(\pi_i^d\left(S \oplus (\overline{M} \| 0^*)\right)\right)$
4: $Z \leftarrow \texttt{left}_{r_S}(S)$
5: **return** Z

Function `duplex-pi$.init`
Input: $(id, M, i) \in [\mu] \times \mathbb{F}_p^{\leq r_I} \times [\zeta]$
Output: $Z \in \mathbb{F}_p^{r_S}$
1: $IV \leftarrow 0^{r_I} \| \mathcal{H}(id)$
2: **return** `CoreDuplex.init`(IV, M, i)

Function `duplex-pi$.next`
Input: $(M, i) \in \mathbb{F}_p^{\leq r_A} \times [\zeta]$
Output: $Z \in \mathbb{F}_p^{r_S}$
1: **return** `CoreDuplex.next`(M, i)

Function `CoreDuplex.next`
Input: $(M, i) \in \mathbb{F}_p^{\leq r_A} \times [\zeta]$
Output: $Z \in \mathbb{F}_p^{r_S}$
1: $\overline{M}, d \leftarrow \texttt{padBlock}_{r_A}(M)$
2: $S \leftarrow \mathcal{P}\left(\pi_i^d\left(S \oplus (\overline{M} \| 0^*)\right)\right)$
3: $Z \leftarrow \texttt{left}_{r_S}(S)$
4: **return** Z

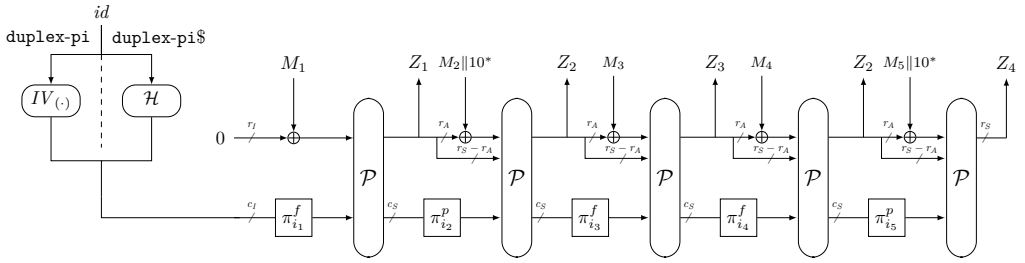

Figure 4: `duplex-pi` and `duplex-pi$` constructions. Note that the padding function `padBlock` is integrated inside the figure for visibility. The two functions differ in the selection of the IV. For `duplex-pi`, on input of $id \in [\mu]$, the function "$IV_{(\cdot)}$" selects the initial value IV_{id}, whereas for `duplex-pi$` the initial value is generated by an evaluation of $\mathcal{H}$.

the use of various NCPs to capture domain separators, is much more general. Stated differently, `sponge-pi` and `sponge-pi$` have a fixed pattern of absorb-then-squeeze, which means that one can get away with three NCPs (counting the identity as one), but in the duplex more NCPs may be required and the conditions are also more restrictive.

To formally argue indifferentiability of the duplex constructions, we have to compare its construction with a special random object, an online random oracle, or `ORO`. It extends the notion of a `RO` from Section 2 in that it is stateful and repeats outputs when the sequence of duplexing calls is repeated. We take the formalism of Degabriele et al. [DFG23], which is a keyless version of the ideal extendable input function (IXIF) of Daemen et al. [DMV17]. The `ORO` is defined in Algorithm 5. Here, the function `append` takes as input a list and an object and appends the object to the list.

Before going to the final theorems, we wish to note that the counting of the distinguisher's resources specified in Section 3.2 still applies, though it becomes slightly more delicate. In a nutshell, for each duplexing call, we can define a path that lead to that duplexing call (this is exactly the path *path* in the `ORO` functionality of Algorithm 5). $Q_{\mathcal{P}}$

Algorithm 5 `ORO`

Function `ORO.init`	**Function** `ORO.next`
Input: $(id, M, i) \in [\mu] \times \mathbb{F}_p^{\leq r_I} \times [\zeta]$	**Input:** $(M, i) \in \mathbb{F}_p^{\leq r_A} \times [\zeta]$
Output: $Z \in \mathbb{F}_p^{r_S}$	**Output:** $Z \in \mathbb{F}_p^{r_S}$
1: $path \leftarrow (id, M, i)$	1: $path \leftarrow path.\texttt{append}(M, i)$
2: $Z \leftarrow \texttt{RO}(path, r_S)$	2: $Z \leftarrow \texttt{RO}(path, r_S)$
3: **return** Z	3: **return** Z

then count the number of permutation evaluations as well as the number of unique paths in the evaluation of the duplex. In other words, every `init` or `next` call counts as 1 in the number of queries, except if

- in the real world, a permutation call repeats because the associated path had already occurred before, or
- in the ideal world, the associated path had already been queried to `RO` before.

This method of counting might appear counter intuitive, and has led to misinterpretations (see, e.g., Lefevre [Lef24]). However, doubly counting repeated paths, given the current exiting proof techniques, seems to unavoidably lead to a loss in the tightness of the bound. The underlying reason is that current security proofs consider bad events that are triggered by fresh randomness only (i.e., only `RO` or permutation calls with new inputs), and they do not keep track of repeated paths.

We are now ready to prove security of `duplex-pi` and `duplex-pi$`.

Theorem 4. *Let $Q_{\mathcal{P}} \in \mathbb{N}$, $(IV_{id})_{id \in [\mu]} \subset \mathbb{F}_p^{c_I}$. Let $\left(\pi_i^d : \mathbb{F}_p^{c_I} \to \mathbb{F}_p^{c_I}\right)_{i \in [\zeta],\, d \in \{p,f\}}$ be a family of NCPs satisfying Constraints 4 and 5. Let* `C` *denote the* `duplex-pi` *construction based on a random permutation $\mathcal{P}$. There exists a simulator* `S` *such that, for any distinguisher* `D` *making at most $Q_{\mathcal{P}}$ permutation evaluations,*

$$\boldsymbol{Adv}^{\text{iff}}_{\texttt{C},\texttt{S}}(\texttt{D}) \leq \frac{(4\zeta^2 - 2\zeta + 1)Q_{\mathcal{P}}(Q_{\mathcal{P}} - 1)}{p^{c_A}} + \frac{(4\zeta^2 - 2\zeta + 1)\mu Q_{\mathcal{P}}}{2p^{c_I}} + \frac{3{Q_{\mathcal{P}}}^2}{p^b} + \frac{(3r_S \ln(p) + 4)\, Q_{\mathcal{P}}}{p^{c_S}}.$$

Theorem 5. *Let $Q_{\mathcal{P}}, Q_{\mathcal{H}} \in \mathbb{N}$. Let $\left(\pi_i^d : \mathbb{F}_p^{c_I} \to \mathbb{F}_p^{c_I}\right)_{i \in [\zeta],\, d \in \{p,f\}}$ be a family of NCPs satisfying Constraint 4. Let* `C$` *denote the* `duplex-pi$` *construction based on a random oracle $\mathcal{H}$ and a random permutation $\mathcal{P}$. There exists a simulator* `S` *such that, for any distinguisher* `D` *making at most $Q_{\mathcal{P}}$ permutation evaluations and $Q_{\mathcal{H}}$ random oracle evaluations,*

$$\boldsymbol{Adv}^{\text{iff}}_{\texttt{C\$},\texttt{S}}(\texttt{D}) \leq \frac{(4\zeta^2 - 2\zeta + 1)Q_{\mathcal{P}}(Q_{\mathcal{P}} - 1)}{p^{c_A}} + \frac{(4\zeta^2 - 2\zeta + 1)Q_{\mathcal{H}}(Q_{\mathcal{H}} - 1)}{2p^{c_I}} + \frac{3(4\zeta^2 - 2\zeta + 1)Q_{\mathcal{P}}Q_{\mathcal{H}}}{p^{c_I}} + \frac{3{Q_{\mathcal{P}}}^2}{p^b} + \frac{(3r_S \ln(p) + 4)\, Q_{\mathcal{P}}}{p^{c_S}}.$$

The proofs of Theorems 4 and 5 are very similar to each other, and are jointly given in Section 6.3.

Interpretation of the Bounds. Ignoring logarithmic factors, the bounds are of the form

$$\mathbf{Adv}^{\text{iff}}_{\texttt{duplex-pi},\texttt{S}}(\texttt{D}) = \mathcal{O}\left(\frac{\zeta^2 {Q_{\mathcal{P}}}^2}{p^{c_A}} + \frac{\zeta^2 \mu Q_{\mathcal{P}}}{p^{c_I}} + \frac{Q_{\mathcal{P}}}{p^{c_S}}\right),$$

$$\mathbf{Adv}^{\text{iff}}_{\texttt{duplex-pi\$},\texttt{S}}(\texttt{D}) = \mathcal{O}\left(\frac{\zeta^2 {Q_{\mathcal{P}}}^2}{p^{c_A}} + \frac{\zeta^2 Q_{\mathcal{H}}(Q_{\mathcal{H}} - 1)}{p^{c_I}} + \frac{\zeta^2 Q_{\mathcal{H}} Q_{\mathcal{P}}}{p^{c_I}} + \frac{Q_{\mathcal{P}}}{p^{c_S}}\right).$$

Algorithm 6 Setup of the simulators S and S$ for sponge-pi and sponge-pi$

Function S.Initialize()
1: $\mathtt{S}.\mathbf{C}_\mathtt{P} \leftarrow \epsilon$
2: $\mathtt{S}.\mathrm{IV} \leftarrow \{(IV_{id}, id)\}_{id \in [\mu]}$

Function S.ValidIV()
1: $S \leftarrow \epsilon$
2: **for all** $(IV_{id}, id) \in \mathtt{S}.\mathrm{IV}$ **do**
3: $\quad S \leftarrow S \cup \{(0^{r_I} \| IV_{id}, id)\}$
4: **end for**
5: **return** S

Function S$.Initialize()
1: $\mathtt{S\$}.\mathbf{C}_\mathtt{P} \leftarrow \epsilon$
2: $\mathtt{S\$}.\mathbf{C}_\mathtt{H} \leftarrow \epsilon$

Function S$.ValidIV()
1: $S \leftarrow \epsilon$
2: **for all** $(id, h) \in \mathtt{S\$}.\mathbf{C}_\mathtt{H}$ **do**
3: $\quad S \leftarrow S \cup \{(0^{r_I} \| h, id)\}$
4: **end for**
5: **return** S

Compared to the bounds of sponge-pi and sponge-pi$, we obtain a loss quadratic in the number of NCPs. However, the actual number of required NCPs is fixed by the applications and is typically small. For example, in Section 7, we show that $\zeta = 2$ suffices for authenticated encryption. In this case, with appropriately chosen parameters c_I and c_S (as discussed in Section 3.2), duplex-pi and duplex-pi$ achieve indifferentiability security up to approximately $\approx p^{c_A/2}$ queries.

6 Security Proofs

6.1 Proofs of Theorems 1 and 2

In this section we will provide a joint proof for Theorems 1 and 2. We begin by introducing some useful notation and defining the simulator. Then, we introduce an intermediate world, decompose the distance, and bound each component.

Setup. The simulator, named S for sponge-pi and S$ for sponge-pi$, stores its primitive queries in a table $\mathbf{C}_\mathtt{P}$, which comprises tuples of the form (X, Y, dir), where the image of X by $\mathtt{S}_\mathtt{P}$ is defined to be Y, and the query was made in the direction $\mathrm{dir} \in \{\mathrm{fwd}, \mathrm{inv}\}$. Moreover, for sponge-pi$, the simulator has an additional oracle $\mathtt{S\$}_\mathtt{H}$, that produces uniformly random answers, just like a random oracle $\mathcal{H}$. The simulator further stores these hash queries in a table $\mathbf{C}_\mathtt{H}$ that contains tuples of the form (id, h), where the image of id by the oracle is h. Moreover, let

$$\mathbf{C} = \begin{cases} \mathbf{C}_\mathtt{P} & \text{for sponge-pi}, \\ (\mathbf{C}_\mathtt{P}, \mathbf{C}_\mathtt{H}) & \text{for sponge-pi\$}. \end{cases}$$

Let us further define a function ValidIV(), as defined in Algorithm 6 for both simulators. This function, when called, returns a set of tuples of the form (IV, id) such that $IV \in \mathbb{F}_p^b$ corresponds to a valid initial sponge state. In the case of the construction sponge-pi, IV can be of the form $0^{r_I} \| IV_{id}$ for all $id \in [\mu]$ while for sponge-pi$, the (IV, id)s are of the form $(0^{r_I} \| h, id)$, for all $(id, h) \in \mathbf{C}_\mathtt{H}$ at the moment of the query. To keep consistency, id will be used as a label for the RO call that the simulator makes.

Graph Notation. From its table $\mathbf{C}$, the simulator derives a tree construction. The roots are of the form $[id]$, where there exists $(IV, id) \in \mathtt{ValidIV}()$. The other nodes are elements in $\mathbb{F}_p^b$. Given $X, Y \in \mathbb{F}_p^b$, $m_p \in (\mathbb{F}_p)^{\leq r_A - 1} \setminus \{\epsilon\}$, and $m_f \in (\mathbb{F}_p)^{r_A}$ we define four kinds of edges:

- $X \xrightarrow{m_f/a} Y$ denotes that $(X \oplus (m_f \| 0^{c_A}), Y, \mathrm{dir}) \in \mathbf{C}_\mathtt{P}$;

- $X \longrightarrow Y$ is a special case of the above, and it denotes that $(X, Y, \text{dir}) \in \mathbf{C}_{\mathtt{P}}$;
- $X \xrightarrow{m_p/p} Y$ denotes that $(\pi^p(X \oplus (m_p \| 10^*)), Y, \text{dir}) \in \mathbf{C}_{\mathtt{P}}$;
- $X \xrightarrow{m_f/f} Y$ denotes that $(\pi^f(X \oplus (m_f \| 0^{c_A})), Y, \text{dir}) \in \mathbf{C}_{\mathtt{P}}$.

The symbol a stands for absorb, while p and f indicate a last message with respectively a partial and a full block. Similarly, given $Y \in \mathbb{F}_p^b$, $(IV, id) \in \mathtt{ValidIV}()$, $m_p \in (\mathbb{F}_p)^{\leq r_I - 1}$, and $m_f \in (\mathbb{F}_p)^{r_I}$, we define three kinds of edges between a root node $[id]$ and Y as follows:

- $[id] \xrightarrow{m_f/a} Y$ denotes that $(IV \oplus (m_f \| 0^{c_I}), Y, \text{dir}) \in \mathbf{C}_{\mathtt{P}}$;
- $[id] \xrightarrow{m_p/p} Y$ denotes that $(\pi^p(IV \oplus (m_p \| 10^*)), Y, \text{dir}) \in \mathbf{C}_{\mathtt{P}}$;
- $[id] \xrightarrow{m_f/f} Y$ denotes that $(\pi^f(IV \oplus (m_f \| 0^{c_I})), Y, \text{dir}) \in \mathbf{C}_{\mathtt{P}}$.

The set of *absorbing paths* includes paths that correspond to intermediate states whose nodes would be in the real world intermediate states *during the absorbing phase*. These paths are of the form

$$[id] \xrightarrow{m_1/a} S_2 \xrightarrow{m_2/a} \cdots \xrightarrow{m_n/a} S_{n+1}\,.$$

We will abbreviate those as $[id] \xrightarrow{m_1 \| \cdots \| m_n/a} S_{n+1}$. The set of *squeezing paths* include paths whose nodes would be in the real world intermediate states *during the squeezing phase*. They are of the form

$$[id] \xrightarrow{m_1/a} S_2 \xrightarrow{m_2/a} \cdots \xrightarrow{m_{n-1}/a} S_m \xrightarrow{m_n/d} Z_1 \longrightarrow Z_2 \longrightarrow \cdots \longrightarrow Z_\ell\,,$$

for $d \in \{p, f\}$. We will abbreviate this as $[id] \xrightarrow{m_1 \| \cdots \| m_n, \ell} Z_\ell$. Note that the $/d$ symbol in the path is not included. This is because the message blocks are not padded in this graph representation, so that the length of $m_1 \| \cdots \| m_n$ provides sufficient information to deduce whether the last call uses a full or partial block. The set of paths that are *valid* corresponds to the union of absorbing paths with squeezing paths. We make a clear distinction between them, since the simulator only needs to guarantee consistency for squeezing paths. Moreover, we define $\mathtt{Rooted}(\mathbf{C})$ as the set of rooted nodes $Z \in \mathbb{F}_p^b$ from which one can build a valid path $[id] \xrightarrow{M/a} Z$ or $[id] \xrightarrow{M, \ell} Z$.

Even when there are no collisions, there might be some overlap between squeezing paths and absorbing paths when π^p is the identity permutation. For instance, assume that $(IV \oplus m \| 10^{c_I}, S_1, \text{fwd}), (S_1, S_2, \text{fwd}) \in \mathbf{C}_{\mathtt{P}}$ for some $(IV, id) \in \mathtt{ValidIV}()$, then this leads to two paths:

$$[id] \xrightarrow{m/p, 2} S_2 \text{ and } [id] \xrightarrow{m \| 1 \| 0^{r_A}/a} S_2\,.$$

Note that this overlap also happens with the plain sponge construction.

The Simulator. The simulator keeps track of the graph construction and uses it to output consistent answers. It is essentially the same sort of simulator as in the sponge construction [BDPV08], but it manages a more complex set of valid paths. Since `sponge-pi` and `sponge-pi$` differ only in the initial state, $\mathtt{S}_{\mathtt{P}}$ and $\mathtt{S\$}_{\mathtt{P}}$ (resp., $\mathtt{S}_{\mathtt{P}}^{-1}$ and $\mathtt{S\$}_{\mathtt{P}}^{-1}$) are the same. Algorithm 7 describes the simulators.

Algorithm 7 Simulator interfaces for `sponge-pi` and `sponge-pi$`

Function $\mathtt{S}_{\mathtt{P}}^{-1}(Y)$ (same for $\mathtt{S\$}_{\mathtt{P}}^{-1}$)

if $\exists (X, Y) \in \mathtt{S}.\mathbf{C}_{\mathtt{P}}$ **then**
  return Y
end if
$X \xleftarrow{\$} \mathbb{F}_p^b$
$\mathtt{S}.\mathbf{C}_{\mathtt{P}} \leftarrow \mathtt{S}.\mathbf{C}_{\mathtt{P}} \cup \{(X, Y)\}$
return X

Function $\mathtt{S\$}_{\mathtt{H}}(id)$

if $\exists (id, h) \in \mathtt{S\$}.\mathbf{C}_{\mathtt{H}}$ **then**
  return h
end if
$h \xleftarrow{\$} \mathbb{F}_p^{c_I}$
$\mathtt{S\$}.\mathbf{C}_{\mathtt{H}} \leftarrow \mathtt{S\$}.\mathbf{C}_{\mathtt{H}} \cup \{(id, h)\}$
return h

Function $\mathtt{S}_{\mathtt{P}}(X)$ (same for $\mathtt{S\$}_{\mathtt{P}}$)

if $\exists (X, Y) \in \mathtt{S}.\mathbf{C}_{\mathtt{P}}$ **then**
  return X
end if
$Y \xleftarrow{\$} \mathbb{F}_p^b$
// Start of squeezing phase with partial block
if $\exists\, (IV, id) \in \mathtt{S.ValidIV}()$, $S \in \mathbb{F}_p^b$, $M \in \mathbb{F}_p^* \setminus \{\epsilon\}$, and $m \in \mathbb{F}_p^{\leq r_A - 1} \setminus \{\epsilon\}$ such that $[id] \xrightarrow{M/a} S$ and $X = \pi^p\left(S \oplus m \| 10^*\right)$ **then**
  $Y_o \leftarrow \mathtt{RO}((id, M \| m), r_S)$
  $Y_i \xleftarrow{\$} \mathbb{F}_p^{c_S}$
  $Y \leftarrow Y_o \parallel Y_i$
end if
// Start of squeezing phase with full block
if $\exists\, (IV, id) \in \mathtt{S.ValidIV}()$, $S \in \mathbb{F}_p^b, M \in \mathbb{F}_p^* \setminus \{\epsilon\}$, and $m \in \mathbb{F}_p^{r_A}$ such that $[id] \xrightarrow{M/a} S$ and $X = \pi^f\left(S \oplus m \| 0^{c_A}\right)$ **then**
  $Y_o \leftarrow \mathtt{RO}((id, M \| m), r_S)$
  $Y_i \xleftarrow{\$} \mathbb{F}_p^{c_S}$
  $Y \leftarrow Y_o \parallel Y_i$
end if
// In the middle of a squeezing phase
if $\exists\, (IV, id) \in \mathtt{S.ValidIV}()$, $M \in \mathbb{F}_p^*$, and $\ell \geq 1$ such that $[id] \xrightarrow{M,\ell} X$ **then**
  $Y_o \leftarrow \mathtt{RO}((id, M), (\ell + 1) r_S)[\ell r_S : (\ell + 1) r_S - 1]$
  $Y_i \xleftarrow{\$} \mathbb{F}_p^{c_S}$
  $Y \leftarrow Y_o \parallel Y_i$
end if
// Start of squeezing phase from IV with one partial block
if $\exists\, (IV, id) \in \mathtt{S.ValidIV}()$ and $m \in \mathbb{F}_p^{\leq r_I - 1}$ such that $X = \pi^p\left(IV \oplus m \| 10^*\right)$ **then**
  $Y_o := \mathtt{RO}((id, m), r_S)$
  $Y_i \xleftarrow{\$} \mathbb{F}_p^{c_S}$
  $Y \leftarrow Y_o \parallel Y_i$
end if
// Start of squeezing phase from IV with one full block
if $\exists\, (IV, id) \in \mathtt{S.ValidIV}()$ and $m \in \mathbb{F}_p^{r_I}$ such that $X = \pi^f\left(IV \oplus m \| 0^{c_I}\right)$ **then**
  $Y_o := \mathtt{RO}((id, m), r_S)$
  $Y_i \xleftarrow{\$} \mathbb{F}_p^{c_S}$
  $Y \leftarrow Y_o \parallel Y_i$
end if
$\mathtt{S}.\mathbf{C}_{\mathtt{P}} \leftarrow \mathtt{S}.\mathbf{C}_{\mathtt{P}} \cup \{(X, Y)\}$
return Y

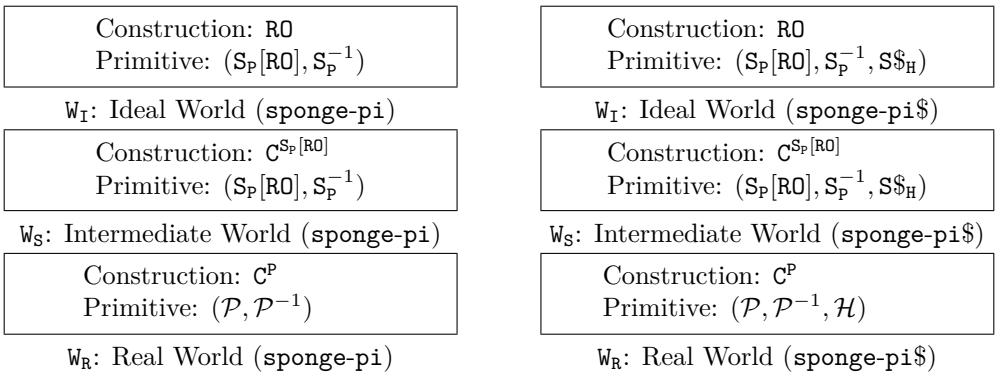

Figure 5: Worlds involved in the security proof.

World Splitting. Similarly to [NO14], we introduce an intermediate world $\mathtt{W_S}$, as illustrated in Figure 5. In this world, the construction oracle gives access to sponge-pi or sponge-pi$, based on the simulator, which is itself based on a random oracle hidden from the adversary. When dealing with sponge-pi$, $\mathtt{W_S}$ additionally includes the simulator $\mathtt{S\$_H}$. For the sake of the proof, we introduce a table $\widehat{\mathbf{C}_\mathtt{P}}$, that comprises tuples of the form (X, Y, dir, O), where $O = D$ means that the query was from the distinguisher, else $O = C$ means that the query comes from the construction. Similarly with sponge-pi$, define $\widehat{\mathbf{C}_\mathtt{H}}$ as an extension of $\mathbf{C}_\mathtt{H}$, that comprises tuples of the form (I, h, O), where the image of I by $\mathcal{H}$ is h, and the query comes from either the distinguisher (i.e., $O = D$), or the construction (i.e., $O = C$). With these extended tables, we can differentiate the set of rooted nodes: $\mathtt{Rooted}_D(\widehat{\mathbf{C}})$ represents the rooted nodes reached solely through distinguisher queries.

We have

$$\mathbf{Adv}^{\mathrm{iff}}_{\mathtt{C},\mathtt{S}}(\mathtt{D}) \leq \Delta_\mathtt{D}\left(\mathtt{W_R}\,;\,\mathtt{W_S}\right) \tag{6}$$

$$+\ \Delta_\mathtt{D}\left(\mathtt{W_S}\,;\,\mathtt{W_I}\right). \tag{7}$$

We first bound (6), which boils down to evaluating the distance between the simulator and a random permutation. Remark that $\mathtt{S\$_H}$ is a random oracle that is independent from $\mathtt{S_P}$ and $\mathtt{S_P}^{-1}$. Then, since $\mathtt{S_P}$ makes random oracle calls with disjoint inputs for any fresh query, it behaves like a random function. This distance can thus be upper bounded by the PRP/PRF switching lemma, and we obtain

$$\Delta_\mathtt{D}\left(\mathtt{W_R}\,;\,\mathtt{W_S}\right) \leq \frac{\binom{Q_\mathcal{P}}{2}}{p^b} \leq \frac{{Q_\mathcal{P}}^2}{p^b}. \tag{8}$$

The remainder of the proof is dedicated to the analysis of (7).

Bad Events. Let Q denote the total number of queries made by the distinguisher, so that $Q = Q_\mathcal{P}$ for sponge-pi, and $Q = Q_\mathcal{P} + Q_\mathcal{H}$ for sponge-pi$. For $i \in [Q]$, let $\mathbf{C}_\mathtt{P}[i]$ and $\mathbf{C}_\mathtt{H}[i]$ denote the state of respectively $\mathbf{C}_\mathtt{P}$ and $\mathbf{C}_\mathtt{H}$ after the first i queries. We define a family of bad events, indexed by a query index $i \in [Q]$, and defined over $\mathbf{C}$. sponge-pi and sponge-pi$ only diverge in the definition of bad events related to the initial values.

- **CollP**$[i]$: $\exists (X, Y, \mathrm{dir}) \in \mathbf{C}_\mathtt{P}[i] \setminus \mathbf{C}_\mathtt{P}[i-1]$, $(X', Y', \mathrm{dir}') \in \mathbf{C}_\mathtt{P}[i-1]$ such that

$$\left\{\mathtt{right}_{c_A}(Z), \pi^p\left(\mathtt{right}_{c_A}(Z)\right), \pi^f\left(\mathtt{right}_{c_A}(Z)\right)\right\} \cap \left\{\mathtt{right}_{c_A}(Z'), \pi^p\left(\mathtt{right}_{c_A}(Z')\right), \pi^f\left(\mathtt{right}_{c_A}(Z')\right)\right\} \neq \epsilon,$$

where

$$(Z, Z') = \begin{cases} (Y, Y') & \text{if dir} = \text{fwd}\,, \\ (X, X') & \text{otherwise}\,; \end{cases}$$

- **ConnectP**$[i]$: $\exists (X, Y, \text{dir}) \in \mathbf{C}_{\mathtt{P}}[i] \setminus \mathbf{C}_{\mathtt{P}}[i-1]$, $(X', Y', \text{dir}') \in \mathbf{C}_{\mathtt{P}}[i-1]$ such that

$$\left\{\mathtt{right}_{c_A}(Z), \pi^p\left(\mathtt{right}_{c_A}(Z)\right), \pi^f\left(\mathtt{right}_{c_A}(Z)\right)\right\} \cap \left\{\mathtt{right}_{c_A}\left(\bar{Z}'\right), \pi^p\left(\mathtt{right}_{c_A}\left(\bar{Z}'\right)\right), \pi^f\left(\mathtt{right}_{c_A}\left(\bar{Z}'\right)\right)\right\} \neq \epsilon\,,$$

  where

$$(Z, \bar{Z}') = \begin{cases} (Y, X') & \text{if dir} = \text{fwd}\,, \\ (X, Y') & \text{otherwise}\,; \end{cases}$$

- **Guess**$[i]$: $\exists (X, Y, \text{dir}) \in \mathbf{C}_{\mathtt{P}}[i] \setminus \mathbf{C}_{\mathtt{P}}[i-1]$; such that $Y \notin \mathtt{Rooted}_D(\widehat{\mathbf{C}}[i])$, but $Y \in \mathtt{Rooted}(\widehat{\mathbf{C}}[i])$;
- **ConnectIV**$[i]$ (for `sponge-pi` only): $\exists (X, Y, \text{dir}) \in \mathbf{C}_{\mathtt{P}}[i]$, $id \in [\mu]$ such that

$$\left\{\mathtt{right}_{c_I}(Z), \pi^p\left(\mathtt{right}_{c_I}(Z)\right), \pi^f\left(\mathtt{right}_{c_I}(Z)\right)\right\} \cap \left\{IV_{id}, \pi^p\left(IV_{id}\right), \pi^f\left(IV_{id}\right)\right\} \neq \epsilon\,,$$

  where

$$Z = \begin{cases} Y & \text{if dir} = \text{fwd}\,, \\ X & \text{otherwise}\,; \end{cases}$$

- **CollH**$[i]$ (for `sponge-pi$` only): $\exists (id, h) \in \mathbf{C}_{\mathtt{H}}[i] \setminus \mathbf{C}_{\mathtt{H}}[i-1]$, $(id', h') \in \mathbf{C}_{\mathtt{H}}[i-1]$ such that

$$\left\{h, \pi^p(h), \pi^f(h)\right\} \cap \left\{h', \pi^p(h'), \pi^f(h')\right\} \neq \epsilon\,;$$

- **ConnectPH**$[i]$ (for `sponge-pi$` only):
  - either $\exists (X, Y, \text{dir}) \in \mathbf{C}_{\mathtt{P}}[i] \setminus \mathbf{C}_{\mathtt{P}}[i-1]$, $(id, h) \in \mathbf{C}_{\mathtt{H}}[i-1]$ such that

$$\left\{h, \pi^p(h), \pi^f(h)\right\} \cap \left\{\mathtt{right}_{c_I}(Z), \pi^p\left(\mathtt{right}_{c_I}(Z)\right), \pi^f\left(\mathtt{right}_{c_I}(Z)\right)\right\} \neq \epsilon\,,$$

    where

$$Z = \begin{cases} Y & \text{if dir} = \text{fwd}\,, \\ X & \text{otherwise}\,, \end{cases}$$

  - or $\exists (id, h) \in \mathbf{C}_{\mathtt{H}}[i] \setminus \mathbf{C}_{\mathtt{H}}[i-1]$, $(X, Y, \text{dir}) \in \mathbf{C}_{\mathtt{P}}[i-1]$ such that there exists $Z \in \{X, Y\}$ with

$$\left\{h, \pi^p(h), \pi^f(h)\right\} \cap \left\{\mathtt{right}_{c_I}(Z), \pi^p\left(\mathtt{right}_{c_I}(Z)\right), \pi^f\left(\mathtt{right}_{c_I}(Z)\right)\right\} \neq \epsilon\,.$$

We define **BadIV**$[i]$ as follows:

$$\mathbf{BadIV}[i] = \begin{cases} \mathbf{ConnectIV}[i] & \text{for } \mathtt{sponge\text{-}pi}\,, \\ \mathbf{CollH}[i] \vee \mathbf{ConnectPH}[i] & \text{for } \mathtt{sponge\text{-}pi\$}\,, \end{cases}$$

and $\mathbf{Bad}[i] := \mathbf{CollP}[i] \vee \mathbf{ConnectP}[i] \vee \mathbf{BadIV}[i] \vee \mathbf{Guess}[i]$. Moreover, for any of those bad events $\mathbf{Event}[i]$, let $\mathbf{Event}$ be

$$\bigvee_{i=1}^{Q} \mathbf{Event}[i] \,.$$

The bad events $\mathbf{CollP} \vee \mathbf{ConnectP} \vee \mathbf{BadIV}$ ensure that the simulator responds consistently with respect to the random oracle. Intuitively, $\mathbf{CollP}$ addresses collisions in the inner part of the construction, modulo the application of the NCPs, ensuring that each simulator query is mapped to at most one squeezing path. The absence of $\mathbf{ConnectP}$ guarantees that an unrooted path cannot later become rooted, which is crucial for maintaining consistency in the simulator's responses. Finally, $\mathbf{BadIV}$ ensures that the rightmost c_I field elements of the initial values appear only at the roots of the tree construction and that the simulator does not connect to the IVs via inverse calls. Additionally, for sponge-pi$, this bad event ensures that a hash evaluation does not inadvertently create a new valid path and that the hash outputs adhere to Constraint 2.

$\mathtt{W_S}$ Versus $\mathtt{W_I}$ as Long as No Bad. We will show that, conditioned on $\neg\mathbf{Bad}$, the worlds $\mathtt{W_I}$ and $\mathtt{W_S}$ have the same distribution. This requires two steps. The first step, detailed in Lemma 3, shows that as long as no bad event occurs, the simulator in $\mathtt{W_S}$ provides consistent answers. However, this alone is not sufficient because the simulator in $\mathtt{W_S}$ has more information than the one in $\mathtt{W_I}$. To address this, we need to ensure that the additional knowledge available to the simulator in $\mathtt{W_S}$ does not provide it with extra information. Intuitively, this is ensured by the absence of $\mathbf{Guess}$. Lemma 4 makes explicit that $\mathtt{W_S}$ and $\mathtt{W_I}$ behave identically until $\mathbf{Bad}$ occur.

Lemma 3. *Let $\mathcal{C}$ be either the* sponge-pi *or* sponge-pi$ *construction, based on the simulator, which is itself based on a random oracle* RO*. For any $M \in p^*$ such that there exists a valid path $[id] \xrightarrow{M,\ell} Z$ in $\mathbf{C}$, where $(IV, id) \in \mathtt{ValidIV}()$, we have*

$$\mathcal{C}(id, M, \ell r_S) = \mathtt{RO}((id, M), \ell r_S) \,.$$

Proof. By $\neg\mathbf{ConnectP} \wedge \neg\mathbf{BadIV}$, the valid paths are expanded from left to right, and only with forward queries. For instance, if we have a valid path $[id] \xrightarrow{m_1/a} S_1 \xrightarrow{m_2/a} S_2$ with $m_1 \in (\mathbb{F}_p)^{r_I}$, $m_2 \in (\mathbb{F}_p)^{r_A}$, then the following queries were made in order: (i) (for the case of sponge-pi$) query to $\mathtt{S\$_H}$ with input id, and output IV_{id}; (ii) query to $\mathtt{S_P}$ with input $0^{r_I} \| IV_{id} \oplus m_1 \| 0^{c_I}$, and output S_1; (iii) query to $\mathtt{S_P}$ with input $S_1 \oplus m_2 \| 0^{c_A}$, and output S_2.

Keeping this in mind, we argue next that $\neg\mathbf{Bad}$ prevents from having two valid paths of the form $[id_1] \xrightarrow{P_1} S_1$ and $[id_2] \xrightarrow{P_2} S_2$ with $S_1 \neq S_2$, but such that a subsequent forward query with input X and output Y extends these two paths into two valid, squeezing paths that look like $[id_1] \xrightarrow{P_1'} Y$ and $[id_2] \xrightarrow{P_2'} Y$. The list of possible valid paths $[id_e] \xrightarrow{P_e} S_e$ for $e \in \{1, 2\}$ and possibilities for X are the following:

a. $\exists M_e \in \mathbb{F}_p^* \backslash \{\epsilon\}$, $m_e \in \mathbb{F}_p^{\leq r_A - 1} \backslash \{\epsilon\}$, and $[id_e] \xrightarrow{M_e/a} S_e$ such that $X = \pi^p(S_e \oplus m \| 10^*)$;

b. $\exists M_e \in \mathbb{F}_p^* \setminus \{\epsilon\}$, $m \in \mathbb{F}_p^{r_A}$, and $[id_e] \xrightarrow{M_e/a} S_e$ such that $X = \pi^f(S_e \oplus m \| 0^{c_A})$;

c. $\exists M_e \in \mathbb{F}_p^*, \ell \geq 1$, and $[id_e] \xrightarrow{M_e,\ell} S_e$ such that $X = S_e$;

d. $\exists m \in \mathbb{F}_p^{\leq r_I - 1}$, $(IV_e, id_e) \in \mathtt{ValidIV}()$ such that $X = \pi^p(IV_e \oplus m \| 10^*)$ (i.e., $S_e = IV_e$);

e. $\exists m \in \mathbb{F}_p^{r_I}$, $(IV_e, id_e) \in \texttt{ValidIV}()$ such that $X = \pi^f\,(IV_e \oplus m \| 0^{c_I})$ (i.e., $S_e = IV_e$).

The initial values IV_e are never present in the middle of a path thanks to ¬**BadIV**. Moreover, thanks to Constraint 2 for `sponge-pi` and ¬**CollH** for `sponge-pi$`, it is not possible that two different initial values are bound to the same permutation call. Moreover, ¬**CollP** prevents that two forward permutation calls collide on their inner part modulo application of the permutations π^d. Considering all possible combinations, we conclude that having two valid, converging paths as described above is impossible.

Now, assume that we have two paths sharing the same node, i.e., there exists $S \in \mathbb{F}_p^b, (IV_1, id_1), (IV_2, id_2) \in \texttt{ValidIV}()$ and two path labels P_1, P_2 with $(id_1, P_1) \neq (id_2, P_2)$ such that $[id_1] \xrightarrow{P_1} S$ and $[id_2] \xrightarrow{P_2} S$. Thanks to the previous paragraph, and ¬**BadIV**, this means that the intermediate states of both paths are pairwise the same, but that the paths can be read differently. First of all, remark that with Constraint 2 for `sponge-pi` and ¬**CollH** for `sponge-pi$` we must have $id_1 = id_2$. Finally, thanks to Constraint 1 and the absence of **Bad**, this overlapping does not involve confusing $\cdot/f$ with $\cdot/p$ paths and $\cdot/f$ with $\cdot/a$ paths. In the event where $\pi^p(x) = x$, this leaves us with the only possibility where path P_1 is still in the absorption phase, while path P_2 is in the squeezing phase. For such a scenario, in order to extend P_1 into a squeezing path (which would require consistent answers), either the permutation π^f, or a non-zero message block must be added/applied to the state, which will therefore yield to an extended path P_1^* that diverges from any valid extension of P_2. This is therefore not problematic for the consistency.

To summarize, as long as no **Bad** occurs, we only have to take care of the consistency of the answers of the forward simulator. This simulator, in Algorithm 7 contains 5 "if" conditions, which cover all possible scenarios where the query extends any path to a squeezing path. These "if" conditions are analyzed in the second paragraph of the current proof, and we showed as long as no **Bad** occurs there is at most one path that satisfies them. Moreover, such a path can only satisfy one of the "if" conditions. Therefore, the simulator is consistent with respect to the random oracle. □

We finally argue that $\mathtt{W_I}$ and $\mathtt{W_S}$ are identically distributed as long as no bad occurs.

Lemma 4. *We have*

$$\mathbf{Pr}\left[\mathtt{D}^{\mathtt{W_I}} \Rightarrow 1 \mid \neg\mathbf{Bad}\right] = \mathbf{Pr}\left[\mathtt{D}^{\mathtt{W_S}} \Rightarrow 1 \mid \neg\mathbf{Bad}\right].$$

Proof. We already proved in Lemma 3 that the simulator in $\mathtt{W_S}$ replies consistently with respect to the random oracle, but this is insufficient. Indeed, another difference between $\mathtt{W_S}$ and $\mathtt{W_I}$ is that $\mathtt{W_S}$ is indirectly aware of the construction queries made by the adversary. If we were in the setting of public indifferentiability [YMO09, DRS09, MPS12], this would not be a problem, as in that setting the simulator in the ideal world is aware of the construction queries. However, public indifferentiability is weaker than indifferentiability, and we need to further argue that ¬**Bad** makes them not distinguishable. The bad event **Guess** guarantees that the simulator cannot guess a rooted path that comes from construction queries. For instance, if there exists $(IV, id) \in \texttt{ValidIV}()$ and a valid path $[id] \xrightarrow{m_2/a} S_1 \xrightarrow{m_2/f} S_2$ that was generated only from construction queries, then ¬**Guess** guarantees that the adversary will never make an inverse query with input S_1 or S_2, nor a forward query with input $\pi^f(S_1 \oplus m_2 \| 0^{c_A})$ *before* the forward query with input $IV \oplus m_2 \| 0^{c_I}$ was made. In that case, the fact that the simulator has earlier access to the construction queries in $\mathtt{W_S}$ does not change the distribution of the answers from he point of view of the adversary. Thus

$$\mathbf{Pr}\left[\mathtt{D}^{\mathtt{W_I}} \Rightarrow 1 \mid \neg\mathbf{Bad}\right] = \mathbf{Pr}\left[\mathtt{D}^{\mathtt{W_S}} \Rightarrow 1 \mid \neg\mathbf{Bad}\right].$$ □

Probability of Bad. We showed that $\mathtt{W_S}$ and $\mathtt{W_I}$ behave identically as long as no bad event occurs. What remains is to bound the probability of the bad events, which we do in Lemma 5.

Lemma 5. *For* $\mathtt{X} \in \{\mathtt{S}, \mathtt{I}\}$*, with the construction* `sponge-pi`*, we have*

$$\mathbf{Pr}\left[\mathtt{D}^{\mathtt{W_X}} \textit{ sets } \mathbf{Bad}\right] \leq \frac{7Q_{\mathcal{P}}(Q_{\mathcal{P}}-1)}{p^{c_A}} + \frac{7\mu Q_{\mathcal{P}}}{2p^{c_I}} + \frac{2{Q_{\mathcal{P}}}^2}{p^b} + \frac{(3r_S\ln(p)+4)\,Q_{\mathcal{P}}}{p^{c_S}}\,.$$

For $\mathtt{X} \in \{\mathtt{S}, \mathtt{I}\}$*, with the construction* `sponge-pi$`*, we have*

$$\mathbf{Pr}\left[\mathtt{D}^{\mathtt{W_X}} \textit{ sets } \mathbf{Bad}\right] \leq \frac{7Q_{\mathcal{P}}(Q_{\mathcal{P}}-1)}{p^{c_A}} + \frac{7Q_{\mathcal{H}}(Q_{\mathcal{H}}-1)}{2p^{c_I}} + \frac{22Q_{\mathcal{P}}Q_{\mathcal{H}}}{p^{c_I}} + \frac{2{Q_{\mathcal{P}}}^2}{p^b} + \frac{(3r_S\ln(p)+4)\,Q_{\mathcal{P}}}{p^{c_S}}\,.$$

Proof. The bad events are defined over the simulator table $\mathbf{C}$, and **Guess** can only be set in $\mathtt{W_S}$. Moreover, the simulator receives extra queries in $\mathtt{W_S}$, which only increases its success probability to set **Bad**. We will thus evaluate the probability of **Bad** in $\mathtt{W_S}$. A "query" here refers to a call to the simulator, regardless whether this comes from the adversary or the construction. We have

$$\begin{aligned}\mathbf{Pr}_{\mathtt{W_S}}[\mathbf{Bad}] &\leq \sum_{i=1}^{Q} \mathbf{Pr}\left[\mathbf{Bad}[i] \mid \neg\mathbf{Bad}[i-1]\right] \\ &\leq \sum_{i=1}^{Q} \mathbf{Pr}\left[\mathbf{CollP}[i] \mid \neg\mathbf{Bad}[i-1]\right] + \mathbf{Pr}\left[\mathbf{ConnectP}[i] \mid \neg\mathbf{Bad}[i-1]\right] \\ &\quad + \mathbf{Pr}\left[\mathbf{BadIV}[i] \mid \neg\mathbf{Bad}[i-1]\right] + \mathbf{Pr}\left[\mathbf{Guess}[i] \mid \neg\mathbf{Bad}[i-1]\right]\,, \end{aligned} \tag{9}$$

where $\mathbf{Bad}[0]$ denotes an event that never holds. We will evaluate each probability individually. First of all, it is important to note that as long as $\neg\mathbf{Bad}$ holds, the simulator samples its answers uniformly at random, either with direct sampling, or samples the outer part with `RO` calls, and the inner part by itself. In the latter case, the same `RO` entry is never accessed for two different inputs.

Given $i \in [Q]$, let i_P denote the number of queries to $\mathtt{S_P}$ or $\mathtt{S_P}^{-1}$ made before the i^{th} query, and i_H the number of queries to $\mathtt{S\$_H}$ made before the i^{th} query. Moreover, let $\mathbf{1}^i_P$ (resp., $\mathbf{1}^i_H$) denote the indicator function equal to 1 whenever the i^{th} query is a query to $\mathtt{S_P}$ or $\mathtt{S_P}^{-1}$ (resp., $\mathtt{S\$_H}$).

CollP[*i*]. This bad event concerns collisions over c_A field elements. For each old query (X', Y'), the bad event covers 7 different collision scenarios (noting that $\pi^d(x) = \pi^d(x')$ and $x = x'$ are the same case). Therefore,

$$\mathbf{Pr}_{\mathtt{W_S}}\left[\mathbf{CollP}[i] \mid \neg\mathbf{Bad}[i-1]\right] \leq \mathbf{1}^i_P \frac{7i_P}{p^{c_A}}\,. \tag{10}$$

ConnectP[*i*]. Similarly, we have

$$\mathbf{Pr}_{\mathtt{W_S}}\left[\mathbf{ConnectP}[i] \mid \neg\mathbf{Bad}[i-1]\right] \leq \mathbf{1}^i_P \frac{7i_P}{p^{c_A}}\,. \tag{11}$$

BadIV[*i*] with `sponge-pi`. This bad event concerns colliding with one of the μ initial values. For each of these values, there are at most 7 possible collision scenarios, over c_I field elements. Therefore,

$$\mathbf{Pr}_{\mathtt{W_S}}\left[\mathbf{BadIV}[i] \mid \neg\mathbf{Bad}[i-1]\right] = \mathbf{Pr}_{\mathtt{W_S}}\left[\mathbf{ConnectIV}[i] \mid \neg\mathbf{Bad}[i-1]\right] \leq \mathbf{1}^i_P \frac{7\mu}{p^{c_I}}\,. \tag{12}$$

BadIV[i] with sponge-pi$. We have

$$\mathbf{Pr}\left[\mathbf{CollH}[i] \mid \neg\mathbf{Bad}[i-1]\right] \leq \mathbf{1}_H^i \frac{7i_H}{p^{c_I}}\,.$$

ConnectPH[i] also concerns collisions on c_I field elements. Assuming $\neg\mathbf{Bad}[i-1]$, we have two possibilities:

- The i^{th} query is a hash query. There are at most $14i_P$ values from $\mathbf{C}_{\mathtt{P}}$ to hit. The hash answer is uniformly random, thus the query sets **ConnectPH** with a probability of at most $\mathbf{1}_H^i \frac{14i_P}{p^{c_I}}$;
- The i^{th} query is a query to $\mathtt{S}_{\mathtt{P}}$ or $\mathtt{S}_{\mathtt{P}}^{-1}$. Then the uniformly random answer collides with a prior hash output with probability of at most $\mathbf{1}_P^i \frac{7i_H}{p^{c_I}}$.

Therefore:

$$\mathbf{Pr}_{\mathtt{W_S}}\left[\mathbf{BadIV}[i] \mid \neg\mathbf{Bad}[i-1]\right] \leq \mathbf{1}_H^i \frac{7i_H}{p^{c_I}} + \mathbf{1}_H^i \frac{14i_P}{p^{c_I}} + \mathbf{1}_P^i \frac{7i_H}{p^{c_I}}\,. \tag{13}$$

Guess[i]. Setting this event is similar to a guessing game: in order to win, the adversary must be able to guess a node in a valid path without having constructed the path from left to right with its primitive queries. First of all, with sponge-pi$, this case can occur if the adversary makes a forward query with input X, where there exists $(I, h, C) \in \widehat{\mathbf{C}_{\mathtt{H}}}[i-1]$, but there exists no $(I, h, id) \in \widehat{\mathbf{C}_{\mathtt{H}}}[i-1]$ such that $\mathtt{right}_{c_I}(X) = h$. The simulator is a random oracle, thus the values h are generated uniformly at random, and hidden from the adversary as long as it does not make the queries. Therefore, we obtain a probability of at most

$$\mathbf{1}_P^i \frac{i_H}{p^{c_I}}\,.$$

The remaining cases consist of guessing nodes that were generated by queries to $\mathtt{S}_{\mathtt{P}}$ from the construction sponge-pi or sponge-pi$. Thanks to the construction oracle, D has access to the r_S upper field elements of at most $Q_{\mathcal{P}}$ different nodes. Given $u \in (\mathbb{F}_p)^{r_S}$, we define, similarly to Choi et al. [CLL19], the random variable F_u as follows:

$$F_u := \left|\left\{(x, y, \text{fwd}, C) \in \widehat{\mathbf{C}_{\mathtt{P}}} \mid \mathtt{left}_{r_S}(y) = u\right\}\right|\,,$$

i.e., F_u is the number of construction queries which outer part hit u. The distribution of the random variables $(F_u)_u$ is the same as the bin-and-balls experiment described in Lemma 2. Now, given a query $v\|w$ with $v \in (\mathbb{F}_p)^{r_S}$ and $w \in (\mathbb{F}_p)^{c_S}$, the probability that **Guess**[i] is set, conditioned on the query history of the $i-1$ previous queries $\mathbf{C}_{\mathtt{P}}$, is upper bounded by

$$\frac{\max_{u \in (\mathbb{F}_p)^{r_S}} F_u}{p^{c_S}}\,.$$

Therefore, by summing over all possible $\mathbf{C}_{\mathtt{P}}$ we obtain

$$\mathbf{Pr}_{\mathtt{W_S}}\left[\mathbf{Guess}[i] \mid \neg\mathbf{Bad}[i-1]\right] \leq \frac{\mathbb{E}\left[\max_{u \in (\mathbb{F}_p)^{r_S}} F_u\right]}{p^{c_S}}\,.$$

We can use Lemma 2, and obtain

$$\mathbb{E}\left[\max_{a \in (\mathbb{F}_p)^{r_S}} F_u\right] \leq \frac{2Q_{\mathcal{P}}}{p^{r_S}} + 3r_S \ln(p) + 4\,,$$

so that

$$\mathbf{Pr}_{\mathtt{W_S}}\left[\mathbf{Guess}[i] \mid \neg\mathbf{Bad}[i-1]\right] \leq \begin{cases} \mathbf{1}_P^i \dfrac{2Q_{\mathcal{P}}}{p^b} + \mathbf{1}_P^i \dfrac{3r_S \ln(p)+4}{p^{c_S}} \text{ for } \texttt{sponge-pi}\,, \\ \mathbf{1}_P^i \dfrac{2Q_{\mathcal{P}}}{p^b} + \mathbf{1}_P^i \dfrac{3r_S \ln(p)+4}{p^{c_S}} + \mathbf{1}_P^i \dfrac{i_H}{p^{c_I}} \text{ for } \texttt{sponge-pi\$}\,. \end{cases} \tag{14}$$

Conclusion. For sponge-pi, plugging (10) to (12) and (14) into (9) gives

$$\mathbf{Pr}_{\mathtt{W_S}}\left[\mathbf{Bad}\right] \leq \frac{7Q_{\mathcal{P}}(Q_{\mathcal{P}}-1)}{p^{c_A}} + \frac{7\mu Q_{\mathcal{P}}}{2p^{c_I}} + \frac{2{Q_{\mathcal{P}}}^2}{p^b} + \frac{(3r_S\ln(p)+4)\,Q_{\mathcal{P}}}{p^{c_S}}\,.$$

For sponge-pi\$, by plugging (10), (11), (13) and (14) into (9), we obtain

$$\mathbf{Pr}_{\mathtt{W_S}}\left[\mathbf{Bad}\right] \leq \frac{7Q_{\mathcal{P}}(Q_{\mathcal{P}}-1)}{p^{c_A}} + \frac{7Q_{\mathcal{H}}(Q_{\mathcal{H}}-1)}{2p^{c_I}} + \frac{22Q_{\mathcal{P}}Q_{\mathcal{H}}}{p^{c_I}} + \frac{2{Q_{\mathcal{P}}}^2}{p^b} + \frac{(3r_S\ln(p)+4)\,Q_{\mathcal{P}}}{p^{c_S}}\,,$$

hence the lemma. □

Conclusion. From Lemma 4, and using the fundamental lemma of game-playing [BR06],

$$\Delta_{\mathtt{D}}\left(\mathtt{W_S}\,;\,\mathtt{W_I}\right) \leq \mathbf{Pr}_{\mathtt{W_S}}\left[\mathbf{Bad}\right]\,. \tag{15}$$

We can use Lemma 5 to upper bound this term. We plug the obtained bound into (6), along with (8) into (7), and this concludes the theorem.

6.2 Proof of Theorem 3

Let D be any information-theoretic adversary that has access to either the real world, i.e., $(\texttt{fks-pi}^{\mathcal{P}}_{K_{id}})_{id\in[\mu]}$, or the ideal world, i.e., $(\texttt{RO}_{id})_{id\in[\mu]}$. For brevity, we denote the real world as $\mathtt{W_R}$ and the ideal world $\mathtt{W_I}$. D has access to resources as described in Section 2.3. Our proof of Theorem 3 is inspired by that of [DM19], but differs due to the appearance of the NCPs in our construction and the use of the H-Coefficient technique of Lemma 1.

Description of Transcripts. We focus first on the transcripts in $\mathtt{W_R}$. Note that queries are of the form (id_i, M_i, n_i), where id_i is the oracle index, M_i the message queried, and n_i the requested number of output elements. The response is a string Z_i of length n_i elements.

Note that fks-pi pads the message M_i to $(M_i^1, \ldots, M_i^\ell)$, $d_i \leftarrow \texttt{padBlock}_b(M_i)$. The evaluation of fks-pi, thus, in total, leads to $\ell_i + \lceil n_i/r \rceil$ evaluations of $\mathcal{P}$. For this i^{th} query, the state before the j^{th} evaluation of $\mathcal{P}$ is denoted by s_i^j and the state after the evaluation by t_i^j. We add those to the tuples, yielding a construction query transcript

$$\tau_{\mathcal{C}} = \left\{\left(id_i, M_i, n_i, Z_i, \left(s_j^i\right)_{j\geq 1}, \left(t_j^i\right)_{j\geq 1}\right)\right\}_{i=1}^{q}\,,$$

where q is the total number of construction queries. Note that different queries may share a common prefix, and this transcript may contain duplicate information. However, for good transcripts (as we will define them later), these q tuples contain exactly $Q_{\mathcal{C}}$ input-output tuples of $\mathcal{P}$. In the ideal world $\mathtt{W_I}$, the values s_i^j will be sampled uniformly at random with replacement while maintaining consistency with RO for the values Z_i, and the values t_i will be adapted based on the message input and the NCPs.

Finally, $\tau_{\mathcal{P}}$ contains the query response pairs made to $\mathcal{P}$:

$$\tau_{\mathcal{P}} = \{(x_i, y_i)\}_{i=1}^{Q_{\mathcal{P}}}\,.$$

The joint transcript is $\tau = (\tau_{\mathcal{C}}, \tau_{\mathcal{P}})$.

Definition of Good and Bad Transcripts. For any $s, t \in \mathbb{F}_p^b$, we say that $s \simeq t$, if:

$$\left\{s, \pi^p(s), \pi^f(s)\right\} \cap \left\{t, \pi^p(t), \pi^f(t)\right\} \neq \epsilon\,. \tag{16}$$

Note that, trivially, we have:

$$t = s \iff \pi^p(t) = \pi^p(s) \iff \pi^f(t) = \pi^f(s)\,,$$

which means that only 7 comparisons are needed to verify whether (16) holds.

Let $\mathcal{T}$ be the set of all attainable transcripts, i.e., all transcripts that occur with positive probability. We define the following bad events:

- **CollCC**: $\exists (id_i, M_i, n_i, Z_i, (s_i^j)_{j\geq 1}, (t_i^j)_{j\geq 1}), (id_{i'}, M_{i'}, n_{i'}, Z_{i'}, (s_{i'}^j)_{j\geq 1}, (t_{i'}^j)_{j\geq 1}) \in \tau_{\mathcal{C}}$, $j, j' \in \mathbb{N}$ such that $(id_i, M_i^1, \ldots, M_i^j) \neq (id_{i'}, M_{i'}^1, \ldots, M_{i'}^{j'})$ and
$$\left((j,j') \neq (1,1) \text{ and } s_i^j \simeq s_{i'}^{j'}\right) \text{ or } t_i^j \simeq t_{i'}^{j'}\,;$$
- **CollC**: $\exists (id_i, M_i, n_i, Z_i, (s_i^j)_{j\geq 1}, (t_i^j)_{j\geq 1}) \in \tau_{\mathcal{C}}$, $j \in \mathbb{N}$ such that
$$j > 1 \text{ and } \left(s_i^j = \pi^p\left(s_i^j\right) \text{ or } s_i^j = \pi^f\left(s_i^j\right)\right)\,;$$
- **CollCP**: $\exists (id_i, M_i, n_i, Z_i, (s_i^j)_{j\geq 1}, (t_i^j)_{j\geq 1}) \in \tau_{\mathcal{C}}$, $j \in \mathbb{N}$, $(x, y) \in \tau_{\mathcal{P}}$ such that
$$\left(j \neq 1 \text{ and } s_i^j \simeq x\right) \text{ or } t_i^j \simeq y\,;$$
- **CollCKey**: $\exists id \neq id' \in [\mu]$ such that
$$\boldsymbol{K}[id'] \| \boldsymbol{IV}[id'] = \boldsymbol{K}[id] \| \boldsymbol{IV}[id]\,;$$
- **CollPKey**: $\exists id \in [\mu]$, $(x, y) \in \tau_{\mathcal{P}}$ such that
$$\boldsymbol{K}[id] \| \boldsymbol{IV}[id] = x\,.$$

We define the event **Bad** as:

$$\mathbf{Bad} = \mathbf{CollCC} \vee \mathbf{CollC} \vee \mathbf{CollCP} \vee \mathbf{CollCKey} \vee \mathbf{CollPKey}\,,$$

and we say that a transcript τ is called *bad*, i.e., $\tau \in \mathcal{T}_{bad}$, if it satisfies **Bad**. We partition $\mathcal{T}$ into $\mathcal{T} = \mathcal{T}_{good} \sqcup \mathcal{T}_{bad}$.

Bounding the Ratio for Good Transcripts. Consider any $\tau \in \mathcal{T}_{good}$ and consider the case where $\mathtt{D}$ is in $\mathtt{W_I}$. By definition, the intermediate states s_i^j are generated uniformly at random with replacement, and there are exactly $Q_{\mathcal{C}} - \mu$ of them. Furthermore, there are $Q_{\mathcal{P}}$ permutation queries and μ keys (for the initial states). Hence, we have:

$$\mathbf{Pr}\left[D_{\mathtt{W_I}} = \tau\right] = \frac{1}{(p^b)^{Q_{\mathcal{C}}-\mu}\,(p^k)^{\mu}\,(p^b)_{Q_{\mathcal{P}}}}\,. \tag{17}$$

Now, consider the case where $\mathtt{D}$ is placed in $\mathtt{W_R}$. In this case the values s_i^j are generated uniformly at random without replacement using $\mathcal{P}$, and there are exactly $Q_{\mathcal{C}}$ of them, and they are all different from the $Q_{\mathcal{P}}$ tuples in $\tau_{\mathcal{P}}$. Thus, the whole transcript defines $Q_{\mathcal{C}} + Q_{\mathcal{P}}$ tuples for $\mathcal{P}$. Furthermore, there are μ keys (for the initial states). Hence, we have:

$$\mathbf{Pr}\left[D_{\mathtt{W_R}} = \tau\right] = \frac{1}{(p^k)^{\mu}\,(p^b)_{Q_{\mathcal{C}}+Q_{\mathcal{P}}}}\,. \tag{18}$$

Hence, from (17) and (18), we get $\mathbf{Pr}\left[D_{\mathtt{W_R}} = \tau\right] \geq \mathbf{Pr}\left[D_{\mathtt{W_I}} = \tau\right]$, or equivalently:

$$\frac{\mathbf{Pr}\left[D_{\mathtt{W_R}} = \tau\right]}{\mathbf{Pr}\left[D_{\mathtt{W_I}} = \tau\right]} \geq 1\,. \tag{19}$$

Bounding the Probability of Bad Transcripts in $\mathtt{W_I}$. Now, it remains to evaluate the probability that a bad event occurs in $\mathtt{W_I}$. To ease our analysis we introduce the following helper events:

- $\mathbf{CollCP}^+$: $\exists(id_i, M_i, n_i, Z_i, (s_i^j)_{j\geq 1}, (t_i^j)_{j\geq 1}) \in \tau_{\mathcal{C}}$, $(x, y) \in \tau_{\mathcal{P}}$, $j \in \mathbb{N}$ such that
$$j \neq 1 \text{ and } s_i^j \simeq x\,;$$
- $\mathbf{CollCP}^-$: $\exists(id_i, M_i, n_i, Z_i, (s_i^j)_{j\geq 1}, (t_i^j)_{j\geq 1}) \in \tau_{\mathcal{C}}$, $(x, y) \in \tau_{\mathcal{P}}$, $j \in \mathbb{N}$ such that
$$t_i^j \simeq y\,.$$

Notice that these two events are sub-cases of **CollCP**. We introduce the following random variables:

$$\mathtt{Col}_F = \max_{z\in\mathbb{F}_p^r} \#\left\{x \in \mathbb{F}_p^b \mid \exists(x, y) \in \mathbf{C_P} \text{ such that } \mathbf{left}_r(x) = z\right\},$$
$$\mathtt{Col}_B = \max_{z\in\mathbb{F}_p^r} \#\left\{y \in \mathbb{F}_p^b \mid \exists(x, y) \in \mathbf{C_P} \text{ such that } \mathbf{left}_r(y) = z\right\},$$

where $\mathbf{C_P}$ is the table containing all the input-output pairs of $\mathcal{P}$ done during a construction queries, i.e., the input output pairs that are unknown to $\mathtt{D}$. We evaluate each case below.

- **CollCC**: There are at most $2\binom{Q_{\mathcal{C}}}{2}$ pairs that can trigger this event, each of them triggers this event with probability at most $7\xi/p^b$ Hence, the probability of setting this event is at most:
$$\frac{14\xi\binom{Q_{\mathcal{C}}}{2}}{p^b}\,.$$
- **CollC**: There are at most $Q_{\mathcal{C}}$ values that can trigger this event, each of them triggers this event with probability at most $2\xi/p^b$ Hence, the probability of setting this event is at most:
$$\frac{2\xi Q_{\mathcal{C}}}{p^b}\,.$$
- **CollCP**: By the union bound we have:
$$\Pr[\mathbf{CollCP}] \leq \Pr\left[\mathbf{CollCP}^+\right] + \Pr\left[\mathbf{CollCP}^-\right]. \tag{20}$$
We bound each term separately.
  - $\mathbf{CollCP}^+$: Fix $\theta_F \in \mathbb{N}$, suppose that $\mathtt{Col}_F = \theta_F$, and let x be as in the definition of this event. Then, there can be at most θ_F construction queries whose outer part is $\mathbf{left}_r(x)$, and each of these triggers this bad event with probability at most $7/p^c$. Since there are $Q_{\mathcal{P}}$ possible values for x, we get:
$$\begin{aligned}\Pr\left[\mathbf{CollCP}^+\right] &= \sum_{\theta} \Pr\left[\mathbf{CollCP}^+ \mid \mathtt{Col}_F = \theta_F\right] \Pr[\mathtt{Col}_F = \theta_F] \\ &\leq \sum_{\theta} \frac{7Q_{\mathcal{P}}\theta_F \Pr[\mathtt{Col}_F = \theta_F]}{p^c} = \frac{7Q_{\mathcal{P}}\,\mathbb{E}[\mathtt{Col}_F]}{p^c}\,.\end{aligned} \tag{21}$$
  - $\mathbf{CollCP}^-$ Fix $\theta_B \in \mathbb{N}$ and suppose that $\mathtt{Col}_B = \theta_B$. Then we can repeat the analysis on $\mathbf{CollCP}^+$ to conclude that:
$$\Pr\left[\mathbf{CollCP}^+\right] = \frac{7Q_{\mathcal{P}}\,\mathbb{E}[\mathtt{Col}_B]}{p^c}\,. \tag{22}$$

Hence, from (20) to (22) we get that **CollCP** happens with probability at most:

$$\Pr[\textbf{CollCP}] = \frac{7Q_{\mathcal{P}}\,\mathbb{E}[\mathtt{Col}_F + \mathtt{Col}_B]}{p^c}\,;$$

- **CollCKey**: Clearly, the probability that two users with the same IV have their key colliding is upper bounded by:

$$\sum_s \frac{u_s(u_s-1)}{p^{k+1}}\,;$$

- **CollPKey**: There are $Q_{\mathcal{P}}$ distinct primitive intput-output pairs and each of them succeeds in triggering this event with probability at most $u_{\max}/p^k$. Hence, the probability of setting this event is at most:

$$\frac{Q_{\mathcal{P}}u_{\max}}{p^k}\,.$$

Grouping everything together, we get:

$$\textbf{Pr}\,[\textbf{Bad}] \le \sum_s \frac{u_s(u_s-1)}{p^{k+1}} + \frac{Q_{\mathcal{P}}u_{\max}}{p^k} + \frac{14\xi\binom{Q_c}{2}}{p^b} + \frac{2\xi Q_{\mathcal{C}}}{p^b} + \frac{7Q_{\mathcal{P}}\,\mathbb{E}[\mathtt{Col}_F + \mathtt{Col}_B]}{p^c}\,. \tag{23}$$

In order to bound the expectations, we use Lemma 2, yielding:

$$\mathbb{E}[\mathtt{Col}_F] \le \frac{2(Q_{\mathcal{C}}-\mu)}{p^r} + 3r + 4 \le \frac{2Q_{\mathcal{C}}}{p^r} + 3r + 4\,, \tag{24}$$

$$\mathbb{E}[\mathtt{Col}_B] \le \frac{2(Q_{\mathcal{C}}-\mu)}{p^r} + 3r + 4 \le \frac{2Q_{\mathcal{C}}}{p^r} + 3r + 4\,. \tag{25}$$

Combining (23) to (25) we get:

$$\begin{aligned}\textbf{Pr}\,[\textbf{Bad}] \le \sum_s \frac{u_s(u_s-1)}{p^{k+1}} + \frac{Q_{\mathcal{P}}u_{\max}}{p^k} + \frac{14\xi\binom{Q_c}{2}}{p^b} + \frac{2\xi Q_{\mathcal{C}}}{p^b} \\ + \frac{28Q_{\mathcal{C}}Q_{\mathcal{P}}}{p^b} + \frac{14(6r+8)Q_{\mathcal{P}}}{p^c}\,.\end{aligned} \tag{26}$$

We can simplify (26) by noticing that:

$$\frac{14\xi\binom{Q_c}{2}}{p^b} + \frac{2\xi Q_{\mathcal{C}}}{p^b} \le \frac{9\xi {Q_{\mathcal{C}}}^2}{p^b}\,. \tag{27}$$

From (26) and (27), we get the desired result:

$$\Delta_{\mathtt{D}}\,(\mathtt{W}_{\mathtt{R}}\,;\,\mathtt{W}_{\mathtt{S}}) \le \sum_s \frac{u_s(u_s-1)}{p^{k+1}} + \frac{Q_{\mathcal{P}}u_{\max}}{p^k} + \frac{9\xi {Q_{\mathcal{C}}}^2}{p^b} + \frac{28Q_{\mathcal{C}}Q_{\mathcal{P}}}{p^b} + \frac{14(6r+8)Q_{\mathcal{P}}}{p^c}\,.$$

This concludes the proof. □

Algorithm 8 Setup of the simulators S and S$ for duplex-pi and duplex-pi$

Function S.Initialize()
1: $\mathtt{S}.\mathbf{C}_{\mathtt{P}} \leftarrow \epsilon$
2: $\mathtt{S}.G \leftarrow \epsilon$
3: $\mathtt{S}.\mathrm{IV} \leftarrow \{(IV_{id}, id)\}_{id \in [\mu]}$

Function S.ValidIV()
1: $S \leftarrow \epsilon$
2: **for all** $(IV_{id}, id) \in \mathtt{S}.\mathrm{IV}$ **do**
3: $\quad S \leftarrow S \cup \{(0^{r_I} \| IV_{id}, id)\}$
4: **end for**
5: **return** S

Function S$.Initialize ()
1: $\mathtt{S\$}.\mathbf{C}_{\mathtt{P}} \leftarrow \epsilon$
2: $\mathtt{S}.G \leftarrow \epsilon$
3: $\mathtt{S\$}.\mathbf{C}_{\mathtt{H}} \leftarrow \epsilon$

Function S$.ValidIV()
1: $S \leftarrow \epsilon$
2: **for all** $(id, h) \in \mathtt{S\$}.\mathbf{C}_{\mathtt{H}}$ **do**
3: $\quad S \leftarrow S \cup \{(0^{r_I} \| h, id)\}$
4: **end for**
5: **return** S

6.3 Proofs of Theorems 4 and 5

Although it is not possible to reduce the security of duplex-pi and duplex-pi$ to the one of sponge-pi and sponge-pi$, the underlying proofs share many similarities. The graphical representation of the simulator's query history slightly differs, as each duplex call squeezes data. As a consequence, there are no absorbing paths, and no overlap between different paths is possible unless a bad event occurs. Moreover, due to the presence of multiple NCPs, the number of path types is multiplied by a factor of ζ. This will increase the number collision combinations in the bad events, hence the quadratical factor of the form ζ^2 in the bounds. Another distinction is that, unlike in the sponge-pi and sponge-pi$ settings, the idealized object here is *stateful*. However, due to the way the distinguisher resources are measured, we can resort to a step akin to Bertoni et al. [BDPV11a] to serialize the ORO calls, and obtain direct access to the underlying RO at no cost.

Setup. The setup is defined very similarly as the one of Section 6.1, except that the simulator maintains an additional structure named G that will be useful later. The setup algorithms are specified in Algorithm 8, where S and S$ denote respectively the simulator for duplex-pi and duplex-pi$.

Graph Notation. From its table $\mathbf{C}$, the simulator derives a tree construction, similar to the one in the proof of Theorems 1 and 2. However, this time every *valid* path corresponds to a *squeezing* path, and the paths are overloaded with the domain separator indexes. Given $X, Y \in \mathbb{F}_p^b$, $m_p \in (\mathbb{F}_p)^{\leq r_A - 1}$, $m_f \in (\mathbb{F}_p)^{r_A}$, and $i \in [\zeta]$, we define two kinds of edges:

- $X \xrightarrow{m_f/i} Y$ denotes that $\left(\pi_i^f\left(X \oplus (m \| 0^{c_A})\right), Y, \mathrm{dir}\right) \in \mathbf{C}_{\mathtt{P}}$;
- $X \xrightarrow{m_p/i} Y$ denotes that $\left(\pi_i^p\left(X \oplus (m \| 10^*)\right), Y, \mathrm{dir}\right) \in \mathbf{C}_{\mathtt{P}}$.

Similarly, given $Y \in \mathbb{F}_p^b$, $(IV, id) \in \mathtt{ValidIV}()$, $m_p \in (\mathbb{F}_p)^{\leq r_I - 1}$, and $m_f \in (\mathbb{F}_p)^{r_I}$, we define two kinds of edges between a root node $[id]$ and Y as follows:

- $[id] \xrightarrow{m_f/i} Y$ denotes that $\left(\pi_i^f\left(IV \oplus (m \| 0^{c_I})\right), Y, \mathrm{dir}\right) \in \mathbf{C}_{\mathtt{P}}$;
- $[id] \xrightarrow{m_p/i} Y$ denotes that $\left(\pi_i^p\left(IV \oplus (m \| 10^*)\right), Y, \mathrm{dir}\right) \in \mathbf{C}_{\mathtt{P}}$.

The set of *valid paths* is the set of paths that correspond to intermediate states that could be derived from a duplex-pi (or duplex-pi$) call. They are of the form

$$[id] \xrightarrow{m_1/i_1} S_2 \xrightarrow{m_2/i_2} \cdots \xrightarrow{m_n/i_n} S_{n+1}\,.$$

We will abbreviate those as $[id] \xrightarrow{(m_1,i_1),(m_2,i_2),\ldots,(m_n,i_n)} S_{n+1}$. Moreover, we define $\texttt{Rooted}(\mathbf{C})$ as the set of rooted nodes $Z \in \mathbb{F}_p^b$, from which one can build a valid path $[id] \xrightarrow{P} Z$.

Contrarily to the setting in Theorems 1 and 2, there cannot be an overlap between two distinct paths as long as no bad event occurs. In other words, one cannot have $[id_1] \xrightarrow{P_1} S$ and $[id_2] \xrightarrow{P_2} S$ where $(id_1, P_1) \neq (id_2, P_2)$.

Serializing the ORO and Simulator Definition. As we explained in Section 5.2, we count the number of distinguisher queries without doubly counting repeating paths. Thanks to this metric, the simulator can serialize the calls to the ORO with the function named Serialize() from Algorithm 9. This function takes as input a path $P := (M_1, i_1), \ldots, (M_\ell, i_\ell)$, and if $\ell > 1$, checks whether a RO call with input $(M_1, i_1), \ldots, (M_{\ell-1}, i_{\ell-1})$ has already been made (this is tracked explicitly thanks to the structure G), and if yes, it returns $\texttt{RO}(P, r_S)$. In other words, this function allows to directly query the stateless object RO, without any overhead. This makes the proof significantly easier, and close to the one of Theorems 1 and 2 at no cost. The simulators for duplex-pi and duplex-pi\$ are described in detail in Algorithm 9.

World Splitting. We again introduce an intermediate world $\texttt{W}_\texttt{S}$, where construction queries give access to the duplex algorithm from Figure 4, but with the permutation $\mathcal{P}$ replaced with the simulator S or S\$, itself based on an instance of an ORO hidden from the adversary. Moreover, primitive queries give also access to the same simulator. We have

$$\mathbf{Adv}^{\text{iff}}_{\texttt{C},\texttt{S}}(\texttt{D}) \leq \Delta_\texttt{D}\left(\texttt{W}_\texttt{R}\,;\,\texttt{W}_\texttt{S}\right) \tag{28}$$

$$+\ \Delta_\texttt{D}\left(\texttt{W}_\texttt{S}\,;\,\texttt{W}_\texttt{I}\right)\,. \tag{29}$$

As before, we have

$$\Delta_\texttt{D}\left(\texttt{W}_\texttt{R}\,;\,\texttt{W}_\texttt{S}\right) \leq \frac{\binom{Q_\mathcal{P}}{2}}{p^b} \leq \frac{{Q_\mathcal{P}}^2}{p^b}\,. \tag{30}$$

The remainder of the proof is dedicated to the analysis of (29).

Bad Events. The bad events are almost identical to the ones defined in the proof of Theorems 1 and 2, with the difference that there are more NCPs. Let Q denote the total number of queries made by the distinguisher, so that $Q = Q_\mathcal{P}$ for duplex-pi, and $Q = Q_\mathcal{P} + Q_\mathcal{H}$ for duplex-pi\$. For $i \in [Q]$, let $\mathbf{C}_\texttt{P}[i]$ and $\mathbf{C}_\texttt{H}[i]$ denote the state of respectively $\mathbf{C}_\texttt{P}$ and $\mathbf{C}_\texttt{H}$ after the first i queries. The family of bad events is indexed by a query index $i \in [Q]$, and over the tables of the simulator.

- $\mathbf{CollP}[i]$: $\exists (X, Y, \text{dir}) \in \mathbf{C}_\texttt{P}[i] \backslash \mathbf{C}_\texttt{P}[i-1]$, $(X', Y', \text{dir}') \in \mathbf{C}_\texttt{P}[i-1]$, and $(i_1, d_1), (i_2, d_2) \in [\zeta] \times \{p, f\}$ such that

$$\pi_{i_1}^{d_1}\left(\texttt{right}_{c_A}(Z)\right) = \pi_{i_2}^{d_2}\left(\texttt{right}_{c_A}(Z')\right)\,,$$

  where

$$(Z, Z') = \begin{cases} (Y, Y') & \text{if dir} = \text{fwd}\,, \\ (X, X') & \text{otherwise}\,; \end{cases}$$

- $\mathbf{ConnectP}[i]$: $\exists (X, Y, \text{dir}) \in \mathbf{C}_\texttt{P}[i] \setminus \mathbf{C}_\texttt{P}[i-1]$, $(X', Y', \text{dir}') \in \mathbf{C}_\texttt{P}[i-1]$, and $(i_1, d_1), (i_2, d_2) \in [\zeta] \times \{p, f\}$ such that

$$\pi_{i_1}^{d_1}\left(\texttt{right}_{c_A}(Z)\right) = \pi_{i_2}^{d_2}\left(\texttt{right}_{c_A}\left(\bar{Z}'\right)\right)\,,$$

Algorithm 9 Simulator interfaces for `duplex-pi` and `duplex-pi$`

Function $\mathtt{S\$_H}(id)$
1: **if** $\exists (id, h) \in \mathtt{S\$}.\mathbf{C}_\mathtt{H}$ **then**
2: **return** h
3: **end if**
4: $h \xleftarrow{\$} \mathbb{F}_p^{c_I}$
5: $\mathtt{S\$}.\mathbf{C}_\mathtt{H} \leftarrow \mathtt{S\$}.\mathbf{C}_\mathtt{H} \cup \{(id, h)\}$
6: **return** h

Function $\mathtt{S}_\mathtt{P}^{-1}(Y)$ (same for $\mathtt{S\$}_\mathtt{P}^{-1}$)
1: **if** $\exists (X, Y) \in \mathtt{S}.\mathbf{C}_\mathtt{P}$ **then**
2: **return** Y
3: **end if**
4: $X \xleftarrow{\$} \mathbb{F}_p^b$
5: $\mathtt{S}.\mathbf{C}_\mathtt{P} \leftarrow \mathtt{S}.\mathbf{C}_\mathtt{P} \cup \{(X, Y)\}$
6: **return** X

Function $\mathtt{S}_\mathtt{P}(X)$ (same for $\mathtt{S\$}_\mathtt{P}$)
1: **if** $\exists (X, Y) \in \mathtt{S}.\mathbf{C}_\mathtt{P}$ **then**
2: **return** X
3: **end if**
4: $Y \xleftarrow{\$} \mathbb{F}_p^b$
// In the middle of a valid path with full absorption
5: **if** $\exists\, (IV, id) \in \mathtt{S.ValidIV}()$, $S \in \mathbb{F}_p^b, (id, P) \in \mathtt{S}.G$, $i \in [\zeta]$, and $m \in \mathbb{F}_p^{r_A}$ such that $[id] \xrightarrow{P} S$ and $X = \pi_i^f\,(S \oplus m \| 0^{c_A})$ **then**
6: $P \leftarrow P.\mathtt{append}(m, i)$
7: $Y_o \leftarrow \mathtt{Serialize}\,(id, P)$
8: $Y_i \xleftarrow{\$} \mathbb{F}_p^{c_S}$
9: $Y \leftarrow Y_o \,\|\, Y_i$
// In the middle of a valid path with partial absorption
10: **else if** $\exists\, (IV, id) \in \mathtt{S.ValidIV}()$, $S \in \mathbb{F}_p^b, (id, P) \in \mathtt{S}.G$, $i \in [\zeta]$, and $m \in \mathbb{F}_p^{\leq r_A - 1}$ such that $[id] \xrightarrow{P} S$ and $X = \pi_i^p\,(S \oplus m \| 10^*)$ **then**
11: $P \leftarrow P.\mathtt{append}(m, i)$
12: $Y_o \leftarrow \mathtt{Serialize}\,(id, P)$
13: $Y_i \xleftarrow{\$} \mathbb{F}_p^{c_S}$
14: $Y \leftarrow Y_o \,\|\, Y_i$
// Start of squeezing phase from IV with one partial block
15: **else if** $\exists\, (IV, id) \in \mathtt{S.ValidIV}()$, $i \in [\zeta]$, and $m \in \mathbb{F}_p^{\leq r_I - 1}$ such that $X = \pi_i^p\,(IV \oplus m \| 10^*)$ **then**
16: $Y_o := \mathtt{Serialize}\,(id, (m, i))$
17: $Y_i \xleftarrow{\$} \mathbb{F}_p^{c_S}$
18: $Y \leftarrow Y_o \,\|\, Y_i$
// Start of squeezing phase from IV with one full block
19: **else if** $\exists\, (IV, id) \in \mathtt{S.ValidIV}()$, $i \in [\zeta]$, and $m \in \mathbb{F}_p^{r_I}$ such that $X = \pi_i^f\,(IV \oplus m \| 0^{c_I})$ **then**
20: $Y_o \leftarrow \mathtt{Serialize}\,(id, (m, i))$
21: $Y_i \xleftarrow{\$} \mathbb{F}_p^{c_S}$
22: $Y \leftarrow Y_o \,\|\, Y_i$
23: **end if**
24: $\mathtt{S}.\mathbf{C}_\mathtt{P} \leftarrow \mathtt{S}.\mathbf{C}_\mathtt{P} \cup \{(X, Y)\}$
25: **return** Y

Function $\mathtt{Serialize}(id, P)$
1: Parse P as $(M_1, i_1), \ldots, (M_\ell, i_\ell)$
2: **if** $\ell > 1$ and $(M_1, i_1), \ldots, (M_{\ell-1}, i_{\ell-1}) \notin \mathtt{S}.G$ **then**
3: **return** $\perp$
4: **end if**
5: $(Z_1, u) \leftarrow \mathtt{ORO.init}(id, M_1, i_1)$
6: **for** $l = 2, \ldots, \ell$ **do**
7: $Z_l \leftarrow \mathtt{ORO.next}(M_l, i_l)$
8: **end for**
9: $\mathtt{S}.G \leftarrow \mathtt{S}.G.\mathtt{append}((id, P))$
10: **return** Z_ℓ

where

$$(Z, \bar{Z}') = \begin{cases} (Y, X') & \text{if dir} = \text{fwd}\,, \\ (X, Y') & \text{otherwise}\,; \end{cases}$$

- **Guess**$[i]$: $\exists (X, Y, \text{dir}) \in \mathbf{C}_{\mathtt{P}}[i] \setminus \mathbf{C}_{\mathtt{P}}[i-1]$ such that $Y \notin \mathtt{Rooted}_D(\widehat{\mathbf{C}_{\mathtt{P}}}[i])$, but $Y \in \mathtt{Rooted}(\widehat{\mathbf{C}_{\mathtt{P}}}[i])$;
- **ConnectIV**$[i]$ (for `duplex-pi` only): $\exists (X, Y, \text{dir}) \in \mathbf{C}_{\mathtt{P}}[i]$, $id \in [\mu]$, and $(i_1, d_1), (i_2, d_2) \in [\zeta] \times \{p, f\}$ such that

$$\pi_{i_1}^{d_1}\left(\mathtt{right}_{c_I}(Z)\right) = \pi_{i_2}^{d_2}\left(IV_{id}\right)\,,$$

where

$$Z = \begin{cases} X & \text{if dir} = \text{inv}\,, \\ Y & \text{otherwise}\,; \end{cases}$$

- **CollH**$[i]$ (for `duplex-pi$` only): $\exists (id, h) \in \mathbf{C}_{\mathtt{H}}[i] \setminus \mathbf{C}_{\mathtt{H}}[i-1]$, $(id', h') \in \mathbf{C}_{\mathtt{H}}[i-1]$, and $(i_1, d_1), (i_2, d_2) \in [\zeta] \times \{p, f\}$ such that

$$\pi_{i_1}^{d_1}(h) = \pi_{i_2}^{d_2}(h')\,;$$

- **ConnectPH**$[i]$ (for `duplex-pi$` only):
  - either $\exists (X, Y, \text{dir}) \in \mathbf{C}_{\mathtt{P}}[i] \setminus \mathbf{C}_{\mathtt{P}}[i-1]$, $(id, h) \in \mathbf{C}_{\mathtt{H}}[i-1]$, and $(i_1, d_1), (i_2, d_2) \in [\zeta] \times \{p, f\}$ such that

$$\pi_{i_1}^{d_1}(h) = \pi_{i_2}^{d_2}\left(\mathtt{right}_{c_I}(Z)\right)\,,$$

    where

$$Z = \begin{cases} X & \text{if dir} = \text{inv}\,, \\ Y & \text{otherwise}\,, \end{cases}$$

  - or $\exists (id, h) \in \mathbf{C}_{\mathtt{H}}[i] \setminus \mathbf{C}_{\mathtt{H}}[i-1]$, $(X, Y, \text{dir}) \in \mathbf{C}_{\mathtt{P}}[i-1]$ such that there exists $Z \in \{X, Y\}$ with

$$\pi_{i_1}^{d_1}(h) = \pi_{i_2}^{d_2}\left(\mathtt{right}_{c_I}(Z)\right)\,.$$

We define **BadIV**$[i]$ as follows:

$$\mathbf{BadIV}[i] \begin{cases} \mathbf{ConnectIV}[i] & \text{for } \mathtt{duplex\text{-}pi}\,, \\ \mathbf{CollH}[i] \vee \mathbf{ConnectPH}[i] & \text{for } \mathtt{duplex\text{-}pi\$}\,, \end{cases}$$

and $\mathbf{Bad}[i] := \mathbf{CollP}[i] \vee \mathbf{ConnectP}[i] \vee \mathbf{BadIV}[i] \vee \mathbf{Guess}[i]$. Moreover, for any of those bad events **Event**$[i]$, let **Event** be

$$\bigvee_{i=1}^{Q} \mathbf{Event}[i]\,.$$

$\mathtt{W_S}$ versus $\mathtt{W_I}$, as Long as No Bad. The reasoning from Theorems 1 and 2 carries over as we show in Lemmas 6 and 8.

Lemma 6. *Let $\mathcal{C}$ be either the* duplex-pi *or* duplex-pi$ *construction, based on the simulator* S *or* S$*, which is itself based on an* ORO*. For any $(IV, id) \in \mathtt{ValidIV}()$ and valid path $[id] \xrightarrow{P} Z$ derived from the simulator table, it holds that the output of the sequence of calls $\mathcal{C}$.*init *and $\mathcal{C}$.*next *made with the path (id, P) matches the output of the corresponding sequence of calls* ORO.init *and* ORO.next *made with the path (id, P).*

Proof. Compared to sponge-pi/sponge-pi$ (Lemma 3), there is no distinction between absorb and squeeze paths, since every valid path leads to squeezing values. This is compensated by Constraint 4 (and Constraint 5 for duplex-pi), which enforces that no two NCPs can coincide. Therefore, as long as no bad event occurs, there cannot be an overlap between two different paths, and the reasoning from Lemma 3 carries over. □

We finally argue that $\mathtt{W_I}$ and $\mathtt{W_S}$ are identically distributed as long as **Bad** does not occur.

Lemma 7. *We have*

$$\mathbf{Pr}\left[\mathtt{D}^{\mathtt{W_I}} \Rightarrow 1 \mid \neg\mathbf{Bad}\right] = \mathbf{Pr}\left[\mathtt{D}^{\mathtt{W_S}} \Rightarrow 1 \mid \neg\mathbf{Bad}\right]$$

Proof. Recall that the proof of Lemma 4 shows that this guarantee follows from ¬**Guess** combined with the consistency of the answers established in Lemma 6. The same reasoning extends directly to the current case. □

Probability of Bad.

Lemma 8. *For $\mathtt{X} \in \{\mathtt{S}, \mathtt{I}\}$, with the construction* duplex-pi*, we have*

$$\mathbf{Pr}\left[\mathtt{D}^{\mathtt{W_X}} \textit{ sets } \mathbf{Bad}\right] \le \frac{(4\zeta^2 - 2\zeta + 1)Q_{\mathcal{P}}(Q_{\mathcal{P}} - 1)}{p^{c_A}} + \frac{(4\zeta^2 - 2\zeta + 1)\mu Q_{\mathcal{P}}}{2p^{c_I}} + \frac{2{Q_{\mathcal{P}}}^2}{p^b} + \frac{(3r_S \ln(p) + 4)\, Q_{\mathcal{P}}}{p^{c_S}}.$$

For $\mathtt{X} \in \{\mathtt{S}, \mathtt{I}\}$, with the construction duplex-pi$*, we have*

$$\mathbf{Pr}\left[\mathtt{D}^{\mathtt{W_X}} \textit{ sets } \mathbf{Bad}\right] \le \frac{(4\zeta^2 - 2\zeta + 1)Q_{\mathcal{P}}(Q_{\mathcal{P}} - 1)}{p^{c_A}} + \frac{(4\zeta^2 - 2\zeta + 1)Q_{\mathcal{H}}(Q_{\mathcal{H}} - 1)}{2p^{c_I}} + \frac{3(4\zeta^2 - 2\zeta + 1)Q_{\mathcal{P}}Q_{\mathcal{H}}}{p^{c_I}} + \frac{2{Q_{\mathcal{P}}}^2}{p^b} + \frac{(3r_S \ln(p) + 4)\, Q_{\mathcal{P}}}{p^{c_S}}.$$

Proof. The proof is identical as that of Lemma 5, except that the number of collision cases appearing in the bad events **CollP**,**ConnectP**, **ConnectIV**, **CollH**, **ConnectPH** grows quadratically in the number of domain separators. In more detail, there are at most $4\zeta^2$ different combinations of collisions of form $\pi_{i_1}^{d_1}(X) = \pi_{i_2}^{d_2}(X')$. However, $\pi_i^d(X) = \pi_i^d(X')$ is equivalent to $X = X'$, hence $2\zeta - 1$ duplicate cases can be removed from the count. □

Conclusion. From Lemma 4, and using the fundamental lemma of game-playing [BR06], we deduce that

$$\Delta_{\mathtt{D}}(\mathtt{W_S}\,;\,\mathtt{W_I}) \le \mathbf{Pr}_{\mathtt{W_S}}[\mathbf{Bad}]\,.$$

This term can be upper bounded using Lemma 8. Further plugging (30) into (29) completes the proofs of the theorems.

7 Applications

Applications of `sponge-pi` and `sponge-pi$`. The `sponge-pi` variant is basically the plain sponge construction, but with an application of a NCP on the inner part to account for domain separation between full or partial data. This idea has appeared before in a bit-oriented fashion. In particular, the ESCH hash function of the NIST lightweight cryptography competition SPARKLE [BBC+19] can be seen as a special variant of `sponge-pi`: it injectively pads the message only if its length is not a multiple of the rate. Then, right before squeezing, it adds a constant to the full state, which differs depending on whether the data was full or partial. As such, security of ESCH follows as an immediate corollary of Theorem 1, but for $p = 2$, the 0-string as initial value, and either $(b, c) = (384, 256)$ or $(b, c) = (512, 384)$ (ESCH operates with equal initial, absorbing, and squeezing rate).

Corollary 1. *Let $b, c \in \mathbb{N}$. Let $Q_{\mathcal{P}} \in \mathbb{N}$. Let `C`-$b$-$c$ denote the construction underlying the* ESCH *hash function based on a b-bit random permutation $\mathcal{P}$, with c-bit capacity. There exists a simulator `S` such that, for any distinguisher `D` making at most $Q_{\mathcal{P}}$ permutation evaluations,*

$$\boldsymbol{Adv}^{\text{iff}}_{\texttt{C}\text{-}b,\texttt{S}}(\texttt{D}) \leq \frac{7{Q_{\mathcal{P}}}^2}{2^c} + \frac{3{Q_{\mathcal{P}}}^2}{2^b} + \frac{(3(b-c)\ln(2)+4)\,Q_{\mathcal{P}}}{2^c}\,.$$

That said, as already mentioned in Section 1, the gains of our `sponge-pi` construction over the sponge become more pronounced when considering larger fields, such as fields with $p \approx 2^{256}$ as for Reinforced Concrete [GKL+22]. The `sponge-pi$` variant gives additional freedom over `sponge-pi` in that it allows to hash additional data into the initial inner part. This can be useful if both bits and field elements have to be hashed, or if a random-looking initialization value is needed for cross-protocol security, akin to SAFE [AKMQ23, KBM23].

Applications of `fks-pi`. The `fks-pi` construction can be seen to generalize the idea of domain separation between plaintext absorption and tag generation in authenticated encryption, as in Ascon-PRF [DEMS24], with the immediate avoidance of redundant padding if the plaintext is of size a multiple of the rate. Not surprisingly, the bound is comparable to that of Ascon-PRF [Men23], however, with subtle differences, namely that in our case the initial values cannot be chosen freely and that we use a different approach to tame multicollisions.

Applications of `duplex-pi` and `duplex-pi$`. The duplex variants `duplex-pi` and `duplex-pi$` significantly generalize these applications. That said, these constructions are described for a general amount of 2ζ NCPs, but the actual amount needed highly depends on the actual use case of the duplex. For example, one can describe `sponge-pi` (Algorithm 2) when π^p has no fixed points in terms of `duplex-pi`, as we demonstrate in Algorithm 10. In this case, the `duplex-pi` is always called with second input 1 for a full data block, and with second input 2 for either full or partial data block, which means that it invokes 3 NCPs at most. A comparable reduction from `duplex-pi$` to `sponge-pi$` applies.

One can also describe a SpongeWrap [BDPV11a] style authenticated encryption scheme. At a high level, in authenticated encryption, we get as input a key $K \in \mathbb{F}_p^\kappa$, nonce $N \in \mathbb{F}_p^\nu$, associated data $A \in \mathbb{F}_p^*$, and plaintext $M \in \mathbb{F}_p^*$, which get turned into a ciphertext $C \in \mathbb{F}_p^{|M|}$ and tag $T \in \mathbb{F}_p^\tau$. Such a function can be naively[2] implemented using `duplex-pi`, as we demonstrate in Algorithm 11. The decryption functionality goes the straightforward way, with the main difference that ciphertexts are now overwritten into the state, which is addressed by our indifferentiability proof. In this case, one requires $2\zeta = 4$ NCPs.

[2] More clever implementations may exist, for instance ones that make use of the freedom one has in the index *id* for the initial value.

Algorithm 10 `sponge-pi` using `duplex-pi`

Function `sponge-pi`
Input: $(id, M, n) \in [\mu] \times \mathbb{F}_p^* \times \mathbb{N}_+$
Output: $Z \in \mathbb{F}_p^*$

$(M_1, \ldots, M_\ell) \leftarrow \mathtt{pad}_{r_I, r_A}(M)$
if $\ell = 0$ **then**
  $Z \leftarrow \mathtt{duplex\text{-}pi.init}(\epsilon, 2)$
else if $\ell = 1$ **then**
  $Z \leftarrow \mathtt{duplex\text{-}pi.init}(M_1, 2)$
else
  $\mathtt{duplex\text{-}pi.init}(IV, M_1, 1)$ // discard output
  for $2 \leq i \leq \ell - 1$ **do**
    $\mathtt{duplex\text{-}pi.next}(M_i, 1)$ // discard output
  end for
  $Z \leftarrow \mathtt{duplex\text{-}pi.next}(M_\ell, 2)$
end if
for $2 \leq i \leq \lceil \frac{n}{r} \rceil$ **do**
  $Z \leftarrow Z \| \mathtt{duplex\text{-}pi.next}(0^{r_A}, 1)$
end for
return $Z[1:n]$

Security of the function immediately follows from the indifferentiability of the `duplex-pi` construction. The `SpongeWrap-pi` scheme can also be generalized to the SCHWAEMM authenticated encryption scheme of above-mentioned SPARKLE submission [BBC+19], noting that it (just like ESCH) adds a constant to the inner part to separate between full and partial data. (We note that SCHWAEMM absorbs data in a Beetle-style fashion [CDNY18] but this is captured by our analysis because we basically prove blockwise adaptive security.)

Again, the true power of `duplex-pi` and `duplex-pi$` becomes only more apparent if arbitrary fields are considered. Even beyond that, the flexibility of these duplexes allows them to be applicable in many more protocols. To be more precise, as these functionalities extend the power of the SAFE API [AKMQ23, KBM23] in the sense that no input-output pattern encoding has to be hashed prior to the evaluation, all applications that SAFE API has (commitment schemes, multi-round interactive protocols, Fiat-Shamir, etc.) would fare well with `duplex-pi` and `duplex-pi$`, too.

8 Conclusion

In this work, we introduced arithmetization-oriented permutation-based sponges and duplexes, and a full keyed sponge, all with length-preserving padding using NCPs, without sacrificing security. In a bit more detail, in a bit-oriented setting, the security degradation is similar to padding into the inner part of the state (basically reducing the capacity by 1), but the gains of our constructions become more pronounced when they are used on large finite fields. In this case, depending on the size of the field, and the choice of permutation size, capacity, and rate, the existing security proofs may be void but our bounds still yield good security.

We also demonstrated the potential of our `duplex-pi`, as it can be used to describe plain hashing (notably, `sponge-pi`) as well as authenticated encryption (`SpongeWrap-pi`). It can also be used in many other applications such as interactive protocols. That said, we wish to stress that the resulting bounds will always be birthday bound in the inner part. This is because we proved `duplex-pi` and `duplex-pi$` in the indifferentiability framework

Algorithm 11 `SpongeWrap-pi`

Function `SpongeWrap-pi`
Input: $(K, N, A, M) \in \mathbb{F}_p^\kappa \times \mathbb{F}_p^\nu \times \mathbb{F}_p^* \times \mathbb{F}_p^*$
Output: $(C, T) \in \mathbb{F}_p^{|M|} \times \mathbb{F}_p^\tau$

$B \leftarrow K \| N \| A$
$(B_1, \ldots, B_\ell) \leftarrow \mathtt{pad}_{r_I, r_A}(B)$
if $\ell = 1$ **then**
  $Z \leftarrow \mathtt{duplex\text{-}pi.init}(B_1, 2)$
else
  $\mathtt{duplex\text{-}pi.init}(IV, B_1, 1)$ // discard output
  for $2 \leq i \leq \ell - 1$ **do**
    $\mathtt{duplex\text{-}pi.next}(B_i, 1)$ // discard output
  end for
  $Z \leftarrow \mathtt{duplex\text{-}pi.next}(B_\ell, 2)$
end if
$(M_1, \ldots, M_\ell), \leftarrow \mathtt{pad}_{r_A}(M)$
if $\ell = 0$ **then**
  $Z \leftarrow Z \| \mathtt{duplex\text{-}pi.next}(\epsilon, 2)$
else
  for $1 \leq i \leq \ell - 1$ **do**
    $Z \leftarrow Z \| \mathtt{duplex\text{-}pi.next}(M_i, 1)$
  end for
  $Z \leftarrow Z \| \mathtt{duplex\text{-}pi.next}(M_\ell, 2)$
end if
for $2 \leq i \leq \lceil \tau / r_A \rceil$ **do**
  $Z \leftarrow Z \| \mathtt{duplex\text{-}pi.next}(0^{r_A}, 1)$
end for
return $Z[1 : |M| + \tau] \oplus (M \| 0^\tau)$

(as in the original analysis [BDPV11a] and that of Degabriele et al. [DFG23]). It has been observed before that a keyed version of the duplex typically yields better security as the offline and online complexity can be separated [ADMV15, MRV15, DMV17] (similar to how Theorem 3 achieves a better bound than Theorem 1). However, integrating the technique of using NCPs into keyed duplexes seems to require a non-trivial analysis.

Acknowledgments

The authors would like to thank the anonymous reviewers for their helpful and insightful comments. Charlotte Lefevre is supported by the Netherlands Organisation for Scientific Research (NWO) under grant OCENW.KLEIN.435. Mario Marhuenda Beltrán and Bart Mennink are supported by the Netherlands Organisation for Scientific Research (NWO) under grant VI.Vidi.203.099.

References

[AB24] Tomer Ashur and Amit Singh Bhati. Generalized Indifferentiable Sponge and its Application to Polygon Miden VM. Cryptology ePrint Archive, Paper 2024/911, 2024.

[ABK+23] Tomer Ashur, Amit Singh Bhati, Al Kindi, Mohammad Mahzoun, and Léo Perrin. XHash: Efficient STARK-friendly Hash Function. Cryptology ePrint Archive, Paper 2023/1045, 2023.

[ADMV15] Elena Andreeva, Joan Daemen, Bart Mennink, and Gilles Van Assche. Security of Keyed Sponge Constructions Using a Modular Proof Approach. In Gregor Leander, editor, *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, volume 9054 of *Lecture Notes in Computer Science*, pages 364–384. Springer, 2015.

[AGR+16] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 191–219, 2016.

[AKMQ23] Jean-Philippe Aumasson, Dmitry Khovratovich, Bart Mennink, and Porçu Quine. SAFE: Sponge API for Field Elements. Cryptology ePrint Archive, Paper 2023/522, 2023.

[BBC+19] Christof Beierle, Alex Biryukov, Luan Cardoso dos Santos, Johann Großschädl, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, and Qingju Wang. Schwaemm and Esch: Lightweight Authenticated Encryption and Hashing using the Sparkle Permutation Family. Submission to NIST Lightweight Cryptography, 2019.

[BBC+23] Clémence Bouvier, Pierre Briaud, Pyrros Chaidos, Léo Perrin, Robin Salen, Vesselin Velichkov, and Danny Willems. New Design Techniques for Efficient Arithmetization-Oriented Hash Functions: Anemoi Permutations and Jive Compression Mode. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part III*, volume 14083 of *Lecture Notes in Computer Science*, pages 507–539. Springer, 2023.

[BBLT18] Subhadeep Banik, Andrey Bogdanov, Atul Luykx, and Elmar Tischhauser. SUNDAE: Small Universal Deterministic Authenticated Encryption for the Internet of Things. *IACR Trans. Symmetric Cryptol.*, 2018(3):1–35, 2018.

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

[BDPV08] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the Indifferentiability of the Sponge Construction. In Nigel P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*, pages 181–197. Springer, 2008.

[BDPV11a] 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.

[BDPV11b] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak reference, January 2011.

[BDPV12] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Permutation-based encryption, authentication and authenticated encryption. Directions in Authenticated Ciphers, July 2012.

[BR93] Mihir Bellare and Phillip Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993*, pages 62–73. ACM, 1993.

[BR06] Mihir Bellare and Phillip Rogaway. The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426. Springer, 2006.

[CDMP05] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-Damgård Revisited: How to Construct a Hash Function. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 430–448. Springer, 2005.

[CDNY18] Avik Chakraborti, Nilanjan Datta, Mridul Nandi, and Kan Yasuda. Beetle Family of Lightweight and Secure Authenticated Encryption Ciphers. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):218–241, 2018.

[CLL19] Wonseok Choi, ByeongHak Lee, and Jooyoung Lee. Indifferentiability of Truncated Random Permutations. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 175–195. Springer, 2019.

[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.

[DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2. Winning Submission to NIST Lightweight Cryptography, 2021.

[DEMS24] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon MAC, PRF, and Short-Input PRF - Lightweight, Fast, and Efficient Pseudorandom Functions. In Elisabeth Oswald, editor, *Topics in Cryptology - CT-RSA 2024 - Cryptographers' Track at the RSA Conference 2024, San Francisco, CA, USA, May 6-9, 2024, Proceedings*, volume 14643 of *Lecture Notes in Computer Science*, pages 381–403. Springer, 2024.

[DFG23] Jean Paul Degabriele, Marc Fischlin, and Jérôme Govinden. The Indifferentiability of the Duplex and Its Practical Applications. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4-8, 2023, Proceedings, Part VIII*, volume 14445 of *Lecture Notes in Computer Science*, pages 237–269. Springer, 2023.

[DM19] Christoph Dobraunig and Bart Mennink. Leakage Resilience of the Duplex Construction. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part III*, volume 11923 of *Lecture Notes in Computer Science*, pages 225–255. Springer, 2019.

[DMV17] 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.

[DRS09] Yevgeniy Dodis, Thomas Ristenpart, and Thomas Shrimpton. Salvaging Merkle-Damgård for Practical Applications. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 of *Lecture Notes in Computer Science*, pages 371–388. Springer, 2009.

[Dwo05] Morris Dworkin. Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication, 2005-05-01 2005.

[FS86] Amos Fiat and Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Crypto*, volume 86, pages 186–194. Springer, 1986.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, USA, 2009.

[GKL+22] Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, Markus Schofnegger, and Roman Walch. Reinforced Concrete: A Fast Hash Function for Verifiable Computation. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 1323–1335, New York, NY, USA, 2022. Association for Computing Machinery.

[GKL+24] Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, Markus Schofnegger, and Roman Walch. Monolith: Circuit-Friendly Hash Functions with New Nonlinear Layers for Fast and Constant-Time Implementations. *IACR Trans. Symmetric Cryptol.*, 2024(3):44–83, 2024.

[GKR+21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In Michael Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 519–535. USENIX Association, 2021.

[GKS23] Lorenzo Grassi, Dmitry Khovratovich, and Markus Schofnegger. Poseidon2: A Faster Version of the Poseidon Hash Function. In Nadia El Mrabet, Luca De Feo, and Sylvain Duquesne, editors, *Progress in Cryptology - AFRICACRYPT 2023 - 14th International Conference on Cryptology in Africa, Sousse, Tunisia, July 19-21, 2023, Proceedings*, volume 14064 of *Lecture Notes in Computer Science*, pages 177–203. Springer, 2023.

[GM22] Lorenzo Grassi and Bart Mennink. Security of Truncated Permutation Without Initial Value. 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 620–650. Springer, 2022.

[HBHW23] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. ZCash protocol specification, 2023. https://github.com/zcash/zips/blob/master/protocol/protocol.pdf.

[Hir18] Shoichi Hirose. Sequential Hashing with Minimum Padding. *Cryptogr.*, 2(2):11, 2018.

[HPY07] Shoichi Hirose, Je Hong Park, and Aaram Yun. A Simple Variant of the Merkle-Damgård Scheme with a Permutation. In Kaoru Kurosawa, editor, *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, volume 4833 of *Lecture Notes in Computer Science*, pages 113–129. Springer, 2007.

[IK03] Tetsu Iwata and Kaoru Kurosawa. OMAC: One-Key CBC MAC. In Thomas Johansson, editor, *Fast Software Encryption, 10th International Workshop, FSE 2003, Lund, Sweden, February 24-26, 2003, Revised Papers*, volume 2887 of *Lecture Notes in Computer Science*, pages 129–153. Springer, 2003.

[KBM23] Dmitry Khovratovich, Mario Marhuenda Beltrán, and Bart Mennink. Generic Security of the SAFE API and Its Applications. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4-8, 2023, Proceedings, Part VIII*, volume 14445 of *Lecture Notes in Computer Science*, pages 301–327. Springer, 2023.

[KKW18] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 525–537. ACM, 2018.

[Lef24] Charlotte Lefevre. A Note on Adversarial Online Complexity in Security Proofs of Duplex-Based Authenticated Encryption Modes. Cryptology ePrint Archive, Paper 2024/213, 2024.

[LM24] Charlotte Lefevre and Bart Mennink. Generic Security of the Ascon Mode: On the Power of Key Blinding. In Maria Eichlseder and Sébastien Gambs, editors, *Selected Areas in Cryptography, 31st International Workshop, SAC 2024, Montréal, Quebec, Canada, August 26-27, Revised Selected Papers*, Lecture Notes in Computer Science. Springer, 2024. to appear.

[Men23] Bart Mennink. Understanding the Duplex and Its Security. *IACR Trans. Symmetric Cryptol.*, 2023(2):1–46, 2023.

[MK22] Mary Maller and Dmitry Khovratovich. Baloo: open source implementation, 2022. https://github.com/mmaller/caulk-dev/tree/main/baloo.

[MPS12] Avradip Mandal, Jacques Patarin, and Yannick Seurin. On the Public Indifferentiability and Correlation Intractability of the 6-Round Feistel Construction. In Ronald Cramer, editor, *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*, volume 7194 of *Lecture Notes in Computer Science*, pages 285–302. Springer, 2012.

[MRH04] Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In Moni Naor, editor, *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2004.

[MRV15] Bart Mennink, Reza Reyhanitabar, and Damian Vizár. Security of Full-State Keyed Sponge and Duplex: Applications to Authenticated Encryption. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 465–489. Springer, 2015.

[Nat15] National Institute of Standards and Technology. FIPS PUB 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, August 2015.

[NIS19] NIST. Lightweight Cryptography, February 2019. https://csrc.nist.gov/Projects/Lightweight-Cryptography.

[NO14] Yusuke Naito and Kazuo Ohta. Improved Indifferentiable Security Analysis of PHOTON. In Michel Abdalla and Roberto De Prisco, editors, *Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings*, volume 8642 of *Lecture Notes in Computer Science*, pages 340–357. Springer, 2014.

[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.

[Set22] Srinath Setty. Nova: open source implementation, 2022. https://github.com/microsoft/Nova.

[SLS+23] Alan Szepieniec, Alexander Lemmens, Jan Ferdinand Sauer, Bobbin Threadbare, and Al-Kindi. The Tip5 Hash Function for Recursive STARKs. Cryptology ePrint Archive, Paper 2023/107, 2023.

[YMO09] Kazuki Yoneyama, Satoshi Miyagawa, and Kazuo Ohta. Leaky Random Oracle. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 92-A(8):1795–1807, 2009.