CS 655 Project: Preprocessing Analysis on Duplex Constructions, with Applications to Ascon

Jacob White*, Adithya Bhat

May 23rd, 2023

1 Introduction

Sponge constructions and the closely-related duplex constructions (together, *sponge paradigm*) are an intriguing alternative to block cipher-based modes of operation. First introduced by [BDPV07, BDPV08] and later popularized by both the winner of NIST's SHA-3 competition Keccak [BDPV12c], the sponge paradigm has seen an explosion of academic work (e.g. [BKL+11, CHS19, Nai16, DJS19, BDPV08, Saa14, SY15, BDPV10]). Since sponges and duplexes can be constructed with only a permutation or transformation function and one can use the sponge paradigm to construct a wide range functions that are useful in symmetric cryptography (including hashes, authenticated encryption, MACs, PRFs, and PRNGs) [BDPV11], it serves as an extremely versatile basis for cryptosystems. Furthermore, incorporating the sponge paradigm in many ways eases the burden of analyzing the security of sanely-constructed cryptosuites, in part due to its potential as a reference model. Essentially, the sponge paradigm can act as a near-equivalent to a random oracle (ROM) up to the negligible potential for inner collisions [BDPV08] (also see e.g. [Saa14]).

Indeed, many proposed candidates in the NIST Lightweight Cryptography (LWC) competition were constructed using the sponge paradigm due to the ease of creating efficient constructions for a multitude of cryptographic functions. One candidate in particular, Ascon [DEMS21], very recently won the NIST LWC competition and is currently in the process of standardization. Unlike most constructions in the sponge paradigm, Ascon defines logically distinct a- and b-round random permutations p^a, p^b, respectively, but otherwise builds on techniques learned from analyses of similar constructions in the sponge paradigm.

However, most formal analyses of both the security bounds and attacks on the sponge paradigm in the literature have presumed a generic sponge using a random function (transformation or permutation), which does not adequately capture real-life scenarios where the function description is public and can be evaluated in finite-memory by the attacker. As

*Lead author

many analyses and attacks against the sponge paradigm treat its permutations as a (mostly) black-box random oracle, many authors such as [CGH98, MRH04, FGK22] have shed doubt on the idea that true random oracles can be implemented or even provide tight bounds on the advantage of some applications in this light. Indeed, the Ascon authors themselves explicitly acknowledge that their scheme's permutations need not be random or ideal [DEMS21], lending credibility to the idea that tight analyses on duplex constructions should consider additional models which assume an adversary has access to non-uniform hints about the distribution of these functions, such as in the bit-fixing and auxiliary-input preprocessing models.

While models capturing attackers' capabilities for preprocessing attacks have recently received more attention from the literature on block ciphers, only a few recent papers have attempted to analyze the impact of preprocessing attacks against the sponge paradigm itself [CDG18, FGK22], and analyses on duplexes specifically are even more sparse. For this project, we were inspired by a similar line of work specifically on sponge constructions and their applications to hash functions, aiming to "bridge the gap" in these analyses by extending formal analyses of duplex constructions (e.g., for AEAD security) into an appropriate preprocessing attack model as well.

2 Preliminaries

In this section, we will more formally introduce the necessary primitives and models which we will be using in this project.

2.1 Sponge Paradigm

First introduced by the Keccak team [BDPV07, BDPV08], constructions in the sponge paradigm employ a fixed-length, invertible random permutation (or transformation[1]) f as an underlying primitive. The inputs and outputs of these permutations are split into two data streams: an r-bit *rate* which will process the desired input and output blocks—interleaved by applications of f—and a c-bit *capacity* which exists to deter attackers from inverting the permutation. Sponges are typically defined as having an internal b-bit state, where $b = r + c$ and the initial state is $\sigma_0 = 0^b$. While sponges can take on many forms, a common approach is to update the state σ by XORing and then permuting the internal state with the next r-bit message block m_i. That is, for a given permutation π:

$$\begin{aligned}\sigma_0^{(1)} \| \sigma_0^{(2)} &\coloneqq 0^{r+c} \\ \sigma_i^{(1)} \| \sigma_i^{(2)} &\coloneqq \pi\left((\sigma_{i-1}^{(1)} \oplus m_i) \| \sigma_{i-1}^{(2)}\right).\end{aligned}$$

However, Ascon defines their sponge constructions slightly differently. Most notably, they permute *before* XORing to update the internal state. Furthermore, the initial state is defined as being a fixed IV, key, and random nonce (in essence, operating as a salted and keyed

[1]For simplicity of analysis, we will only consider permutation-based constructions in this report.

sponge) instead of 0^b. That is:

$$\sigma_0^{(1)} \| \sigma_0^{(2)} \coloneqq IV \| K \| N \tag{1}$$

$$\sigma_i^{(1)} \| \sigma_i^{(2)} \coloneqq \pi \left(\sigma_{i-1}^{(1)} \| \sigma_{i-1}^{(2)} \right) \oplus (m_i \| 0^c) \tag{2}$$

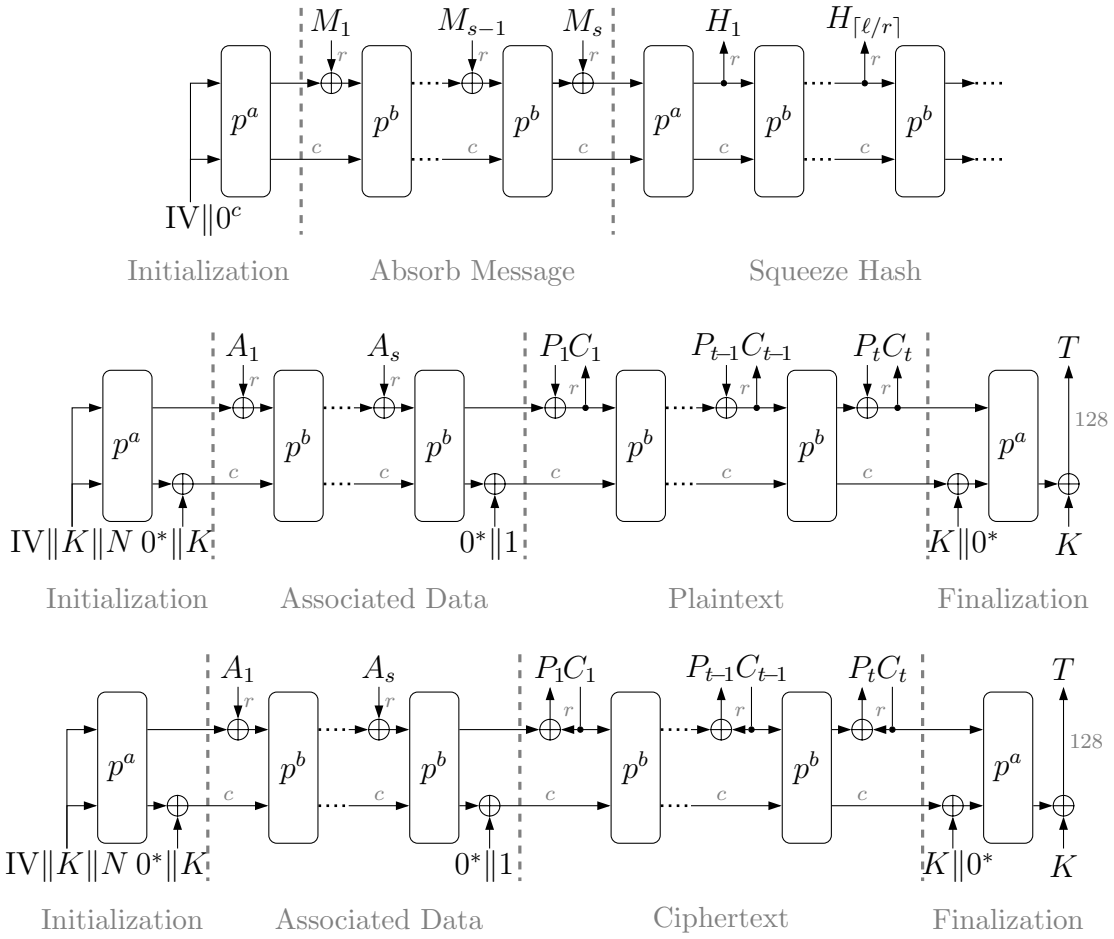

Figure 1: An example of the sponge paradigm as used in the Ascon ciphersuite. Ascon hashes use the sponge construction (shown above), and Ascon AEAD uses duplex constructions for both encryption and decryption (shown below). Diagrams courtesy of [EDMS].

There are two largely equivalent ways to represent constructions in the sponge paradigm. The original *sponge* construction splits data processing into distinct phases, *absorbing*/injecting input blocks then *squeezing*/extracting output blocks from the r-bit rate of the internal state using XOR. The Keccak authors later generalized this construction into a so-called *duplex* sponge construction where these data processing operations can be arbitrarily interleaved [BDPV12a] (i.e., blocks can be absorbed or squeezed in any order). See Figure 1 for a concrete example of how these constructions differ.

To aid in our analysis, we also use the definitions in [BDPV12a] for common patterns of representing permutation calls within the sponge. Let a *blank call* denote a permutation call

which takes as input the internal stated XORed with a preceding input block of all-zeroes. That is, for some permutation π and internal state $\sigma \in \{0,1\}^b$, we have: $\sigma_i \coloneqq \pi(\sigma_{i-1} \oplus 0^b) = \pi(\sigma_{i-1})$. Similarly, let a *mute call* denote a permutation call which has an output bit length of $\ell_i = 0$: i.e., the call does not expose any bits of the intermediate state as output.

Duplex as cascading sponges. The initial work on duplex constructions [BDPV12a] also formally proved that one can simulate an arbitrary duplex construction by representing it as a *cascade* of sponges. As such, duplex constructions benefit from analysis on sponge constructions as well, informally inheriting the same security properties of sponges via reduction. Informally, the reduction to simulate a duplex with sponges is as follows:

Lemma 1 (Duplex-to-sponge reduction)**.** *By the original (*KECCAK*) definition of duplexes, a duplex which makes n permutation calls can be simulated as a cascade of sponges with only $O(n^2)$ permutation calls and minimal memory overhead.*

Proof (Sketch). Let D be a duplex construction with a b-bit state, where $b = r + c$. Let the input string be denoted as r-bit blocks with $m = m_1 \| m_2 \| \dots \| m_s$, where all m_i are padded as necessary to ensure that $|m_i| = r$ for all blocks of the input. By the **duplexing-sponge lemma** [BDPV12a], the i-th output block of the duplex $D.\mathsf{duplexing}(m_i, \ell_i)$ is equivalent to the ℓ_i-bit output of $\mathsf{sponge}(m_0 \| m_1 \| \dots \| m_i, \ell_i)$. Equivalence of the duplex and sponge simulation immediately follows by induction on the input blocks m_i. Intuitively, we can represent the output of a duplex construction D by considering a cascade of sponges that are evaluated on the first i input blocks before returning the resulting r-bit output block. For n calls to a permutation f in an arbitrary duplex D, simulating the duplex purely with sponges requires at most $\sum_{i=1}^{n} i = n(n+1)/2 \in O(n^2)$ calls to f.

Additionally, this reduction is memory-tight. This is because the first part of the computation of every sponge in the cascade is identical, so one only needs to keep track of the index i (i.e., $O(\log n)$ memory) to the next output block and count permutation calls as necessary as additional memory, rewinding back to the initial constant-size state after each sponge has finished computing. □

2.2 Modelling Preprocessing Attacks

Bit-fixing model (BF). Unruh [Unr07] gave a model for analyzing the lower bounds of security of the random oracle model (ROM) and related models under preprocessing attacks. In this model, a computationally-unbounded attacker $\mathcal{A}_1$ fixes at most P input/output pairs of the oracle $\mathcal{O}$ and outputs an S-bit hint: the remaining inputs are assumed to be uniformly random. The online attacker $\mathcal{A}_2$ takes this S-bit hint and can make at most T queries.

Auxiliary-input model (AI). Starting with Unruh [Unr07] and later expanded upon by e.g. [CDG18, FGK22], the auxiliary-model relaxes the assumption that $\mathcal{A}_1$ fixes P input/output pairs of the oracle, and instead can freely analyze the oracle function in a non-uniform manner to arrive at an S-bit hint for $\mathcal{A}_2$. While this makes it much more difficult to analyze directly compared to e.g. BF, there are easy methods for converting upper-bounds

on advantages in the (much easier to analyze) BF-ROM/RPM/ICM/GGM models to the corresponding AI-ROM/RPM/ICM/GGM models.

Transformation. The seminal result **??** describes how to convert concrete analysis in the Bit-fixing model to the auxiliary input model, which we will explore further later in our analysis. Concretely, since we are analyzing the authentication and integrity application of ASCON-AEAD, we looked the unpredictability aspect of the theorem in particular.

Theorem 1 (Bit-fixing to Auxiliary Input [CDG18])**.** *Let $P, K, N, M \in \mathbb{N}$, $N \geq 16$, and $\gamma > 0$. Moreover, let*

$$(\mathtt{AI}, \mathtt{BF}) \in \{(\mathtt{AI} - \mathtt{IC}(K, N), \mathtt{BF} - \mathtt{IC}(P, K, N)), (\mathtt{AI} - \mathtt{GG}(N, M), \mathtt{BF} - \mathtt{GG}(P, N, M))\}.$$

Then,

1. *if an application G is $((S, T, p), \epsilon')$-secure in the $\mathtt{BF}$-model, it is $((S, T, p), \epsilon)$-secure in the $\mathtt{AI}$-model, where*
$$\epsilon \leq \epsilon' + \frac{12(S + \log \gamma^{-1}) \cdot T_G^{\mathtt{comb}}}{P} + 2\gamma;$$
2. *if an unpredictability application G is $((S, T, p), \epsilon')$-secure in the $\mathtt{BF}$-model for*
$$P \geq 6(S + 2\log \gamma^{-1}) \cdot T_G^{\mathtt{comb}},$$
it is $((S, T, p), \epsilon)$-secure in the $\mathtt{AI}$-model for
$$\epsilon \leq 2\epsilon' + 2\gamma,$$

where $T_G^{\mathtt{comb}}$ is the combined query complexity corresponding to G.

3 ASCON analysis under nonce-reuse

Formalization. The Ascon scheme defined as $ASCON_{\pi,IV,K,N}(\cdot)$ uses the sponge as an ideal permutation π, with key K, initialization vector IV, key K, and the nonce N. We further denote by $\sigma_{-1} := \sigma_{-1}^{(1)}||\sigma_{-1}^{(2)}$ where $|\sigma_{-1}^{(1)}| = r$ where r is the sponge rate, and $|\sigma_{-1}^{(2)}| = c$ where c is the capacity of the sponge. The σ_{-1} is initialized to $IV||K||N$. The initial state of the sponge is calculated as $\sigma_0 := \pi(\sigma_{-1}) \oplus 0^r||0^*||K$.

It's input consists of messages $M := (m_1, m_2, \ldots, m_t)$ and associated data $A := (A_1, \ldots, A_s)$. The encryption scheme then computes $\sigma_i := \pi\left(\left(\sigma_{i-1}^{(1)} \oplus A_{i-1}\right)||\sigma_{i-1}^{(2)}\right)$ for $i \in [s]$ where the associated data is absorbed into the sponge. In the end of the absorb phase, a bit is added to $\sigma_s := \sigma_s||0^*||1$ to indicate the start of the squeezing phase. The squeezing phase then computes $\sigma_i := \pi\left(\left(\sigma_{i-1}^{(1)} \oplus m_{i-s}\right)||\sigma_{i-1}^{(2)}\right)$ for $i \in [s+1, t]$. Finally the output cipher text is $\sigma_{s+1}^{(1)} \oplus m_1, \ldots, \sigma_{s+t}^{(1)} \oplus m_t$ and the tag $T := \sigma_{s+t+2} \oplus K$.

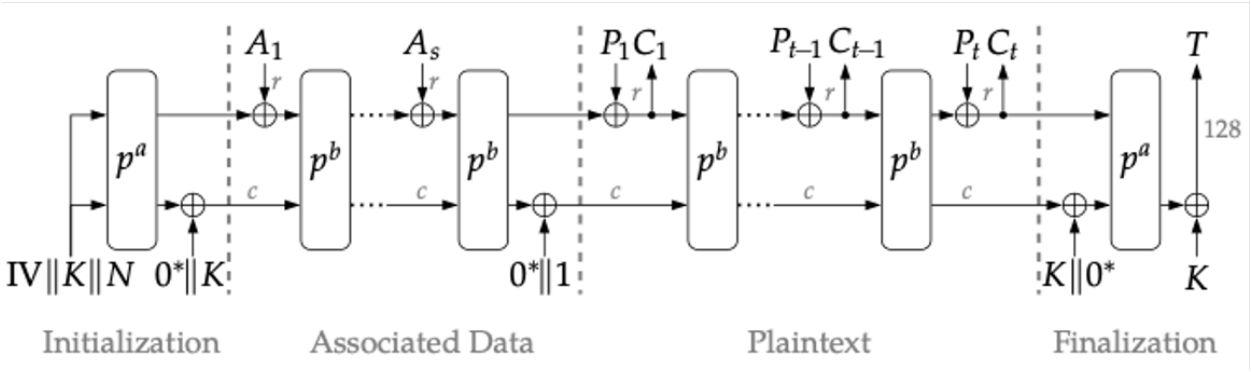

Figure 2: Ascon encryption scheme $ASCON_{\pi,IV,K,N}(\cdot)$

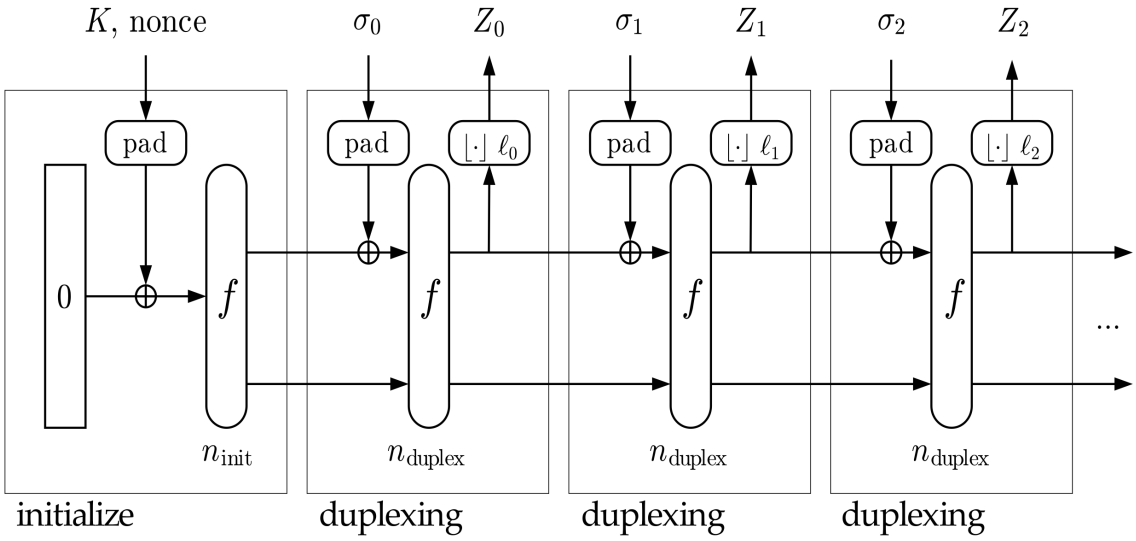

Figure 3: MonkeyDuplex construction.

Nonce-reuse. MonkeyDuplex [BDPV12b] is a reduced-round variant of Keccak, similar to Ascon. One key difference with this scheme, however, is that if the associated data and the nonce are the same for two messages, then an attacker can distinguish between a random string and the encrypted contents in the unpredictability game. Figure 3 presents their construction. It is clear that the nonce cannot be reused from their construction since the σ_s will be the same, and therefore $C_1 \oplus C_1^* = P_1 \oplus P_1^*$ for two messages P_1 and P_1^* with the same associated data. If associated data are added to MonkeyDuplex naïvely, then the associated data will also behave like the nonce, and security mandates that $A_1 \neq A_1^*$ for two authenticated data of any two messages. The modifications in Ascon (in fig. 2), weaken the nonce-reuse attack requirements. Ascon only requires that the nonce is not re-used, but allows the authenticated data to be re–used.

4 ASCON AEAD preprocessing analysis via duplex-to-sponge reduction

4.1 Extensions of the duplex definition

One crucial difference between MonkeyDuplex duplexes and other duplex constructions and analyses in the literature is that different absorb input blocks, permute, and squeeze output blocks from the state in a different order within each duplexing call. And, while many of these papers imply or implicitly assume that these different representations of duplex constructions are effectively equivalent for the purposes of analysis we wanted to try and formalize this notion.

And so, we first prove that regardless of the order of injecting input and permuting the state, these duple x constructions are very closely related and, in many cases, admit essentially equivalent constructions.

Lemma 2. *Full-state duplexes, as originally introduced by [BDPV12a], are invariant in the order of input injection (*absorbing*), permutation calls, and output extraction (*squeezing*). Changing the order into a regular form requires either 0 or 2 permutation calls per sponge, with no effect on the computation itself.*

Proof (Sketch). The only operations which affect the internal state of the sponge are input injection and permutation calls: thus, we need not consider the order of output extraction for the purposes of this proof.

Repeated input extractions (via XOR, by definition) of arbitrary input blocks $m_1, m_2, \ldots$ can be represented as a single XOR of $m_1 \oplus m_2 \oplus \ldots$ (padded as appropriate to a multiple of b) with the state σ by associativity:

$$(\ldots((\sigma_i \oplus m_{i,1}) \oplus m_{i,2})\ldots) \oplus m_{i,j} = \sigma_i \oplus (m_{i,1} \oplus m_{i,2} \oplus \ldots \oplus m_{i,j})$$

Furthermore, permutation calls with no preceding input extraction since the previous permutation call can be represented as injecting the all-zeroes block 0^b, leaving the state unaffected since $\sigma_i \oplus 0^b = \sigma_i$. Furthermore, since the r-bit rate and c-bit capacity of the state is *fixed*, we can (up to resp. padding) distribute over XOR and concatenation if the input $m_i = m_i^{(1)} \| m_i^{(2)}$ is injected into both the rate and capacity channels:

$$\left(\sigma_i^{(1)} \oplus m_i^{(1)}\right) \| \left(\sigma_i^{(2)} \oplus m_i^{(2)}\right) = \left(\sigma_i^{(1)} \| \sigma_i^{(2)}\right) \oplus \left(m_i^{(1)} \| m_i^{(2)}\right)$$

Thus, we can represent arbitrarily repeated logical XORs on the r-bit rate and c-bit capacity to inject input into the full-state duplex with *exactly* 1 XOR between each permutation call.

Lastly, it suffices to prove that whether input injection or permutation calls are first (respectively, last), the duplex can be transformed into an equivalent construction in which the other operation is first (last). If a duplex begins (ends) with a permutation, it can be represented with an injected input 0^b before the first (after the last) permutation. Conversely, if

a duplex begins (ends) with an input injection, it can be represented with a permutation on the end by prepending (appending after input) the permutation call π followed by a blank call to its inverse π^{-1}. By the properties of XOR, permutation inverses, and induction, the state—and thus all output blocks—remain unaffected at every output block.

Therefore, WLOG an arbitrary duplex which injects inputs via XOR on any part of its b-bit state (assuming fixed $b = r + c$) and calls permutation(s) π in any order can be equivalently represented as one that alternates between permutation and input injection (or vice versa) with at most 2 additional permutation calls. □

This supports conjectures provided in other duplex analyses (such as [DMA17]) that one can re-phase the original "XOR-then-permute" definitions of keyed duplexes from the KECCAK authors, extending the definition and scope of existing duplex definitions to other modes of input extraction and permutation (such as the one used by ASCON).

It directly follows as a special case of Lemma 2, then, that simulating a permute-then-XOR duplex (ASCON-like) as a standard XOR-then-permute duplex (KECCAK-like) is trivial: simply treat the first permutation call on the initial state as a blank call, XORed with all zeroes (equivalently, with initial state 0^b and first block $IV\|K\|N$). As a result, many existing theoretical analyeses of KECCAK-family duplex constructions can transfer nearly directly to ASCON with no additional overhead.

4.2 Simulating duplexes with sponges

In the remainder of this section, we will analyze the duplex construction used by ASCON for AEAD encryption. As described in Section 2.1, it is a relatively straightforward process to simulate a duplex with sponges. In particular, each sponge constructs some i-th block of output by re-computing the absorbing phase on each sponge and muting all output blocks except for the desired i-th block. By repeating this process for each block, one can generate duplexes using separate sponges for each output block, each with their own absorbing and squeezing phases.

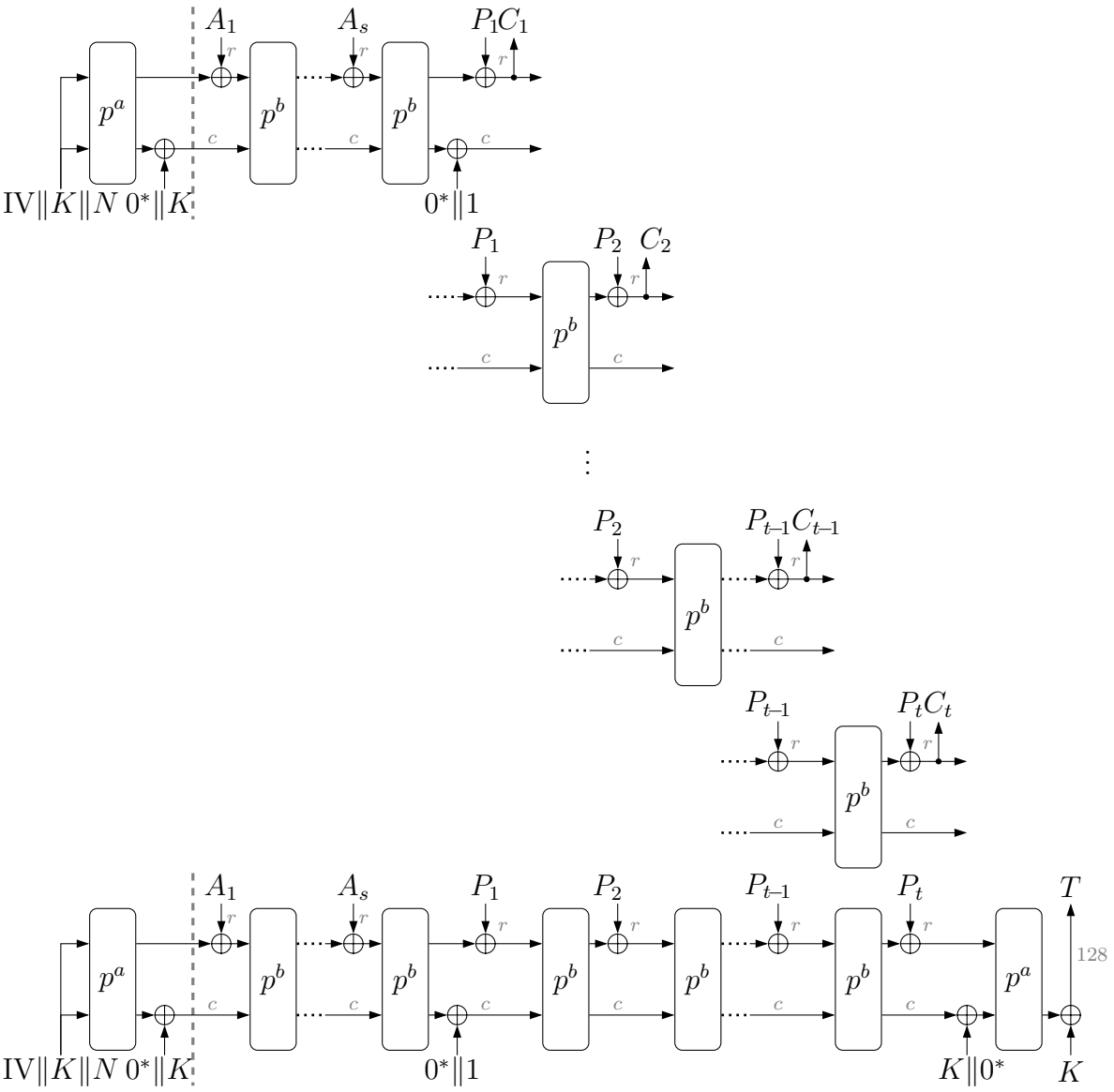

Figure 4: Ascon AEAD encryption, represented as a cascade of sponges.

Without loss of generality, consider the duplex construction given for Ascon encryption. Given s blocks of associated data A_k and t blocks of plaintext P_i, a simple combinatorial argument shows that the original Ascon encryption duplex requires $1 + s + t$ permutation calls. It follows by the duplex simulation algorithm, then, that the cascade of sponges simulating Ascon duplexes requires $\frac{t^2}{2} + st + \frac{3t}{2} + s + 1 \in O(t^2 + st)$ permutation calls. Note that this result is consistent with Lemma 1 for all $t, s \geq 0$.

4.3 Applying preprocessing analyses to sponge simulation

Now that we have a way to simulate any duplex with a cascade of sponges, the key insight for our attempt at analyzing Ascon AEADs with this method is that we can directly apply

existing results such as [CDG18] to analyze the unpredictability of sponge constructions (in particular, the bit-fixing to auxiliary-input transformation described in Theorem 1 above).

If we take the random permutation model (RPM) as our point of comparison, we can take the PRF security of the AEAD cipher using this analysis directly, even for sponge constructions. Using Appendix C.2 of [CDG18] as our point of reference, we can define the unpredictability game $G^{\mathsf{PRF}-\mathsf{S}'}$ and corresponding challenger $C^{\mathsf{PRF}-\mathsf{S}'}$ as follows, this time on a *duplex* of multiple simulated sponges:

1. Challenger $C^{\mathsf{PRF}-\mathsf{S}'}$ picks a random bit $b \leftarrow \{0,1\}$ and a key $k \leftarrow \{0,1\}^c$ (chosen as such since the r-bit rate is known to the adversary through much of the game).
2. When the attacker queries the r-bit blocks of the message $m = m_1 \cdot m_t$
   (a) If $b = 0$, the challenger answers with the output of the sponge on that corresponding key k and message m.
   (b) If $b = 1$, the challenger answers with a randomly-chosen value for each m.
3. The attacker wins iff they correctly guess b.

As proven in [CDG18], they first analyze in the bit-fixing model, making a distinction between *construction queries* (here, to the duplex, the complexity of which remains unchanged from the original proof) and between *primitive queries* to the underlying permutation(s) and their inverses p^a, p^{-a}, p^b, p^{-b} (for simplicity, consider only a single permutation π, π^{-1}).

Incrementally, adversary $\mathcal{A}_2$ can build node and supernode graphs with edges corresponding to primitive queries, fixing at most P query/answer pairs as edges for π. In the event that there is a unique path from the supernode containing key k to any other reachable supernode after the i-th edge is added and this clashes with primitive queries to the permutation oracle. With all other parameters considered constant (e.g. upper bound on number of queries q, upper bound on length of queries messages ℓ, space S), we conjecture that the bit-fixing analysis remains the same with the cascade of similar sponges with the sole exception that now T is $O(\ell^2)$ in the number of permutation calls (where ℓ is proportion to the maximum number of blocks in the associated data and message $s + t$) instead of $O(\ell)$. Otherwise, we conjecture that the same analysis holds. That is, the game $G^{\mathsf{PRF}-\mathsf{S}'}$ should remain secure in the same manner for the bit-fixing model, with upper bound:

$$O\left(\frac{(T + q\ell)^2 + (T + q\ell)P}{2^c}\right)$$

. Then, applying Theorem 1 as described before, we get the following upper bound in the auxiliary input model as well, which should also hold with this caveat of the increase of the number of permutation calls assumed in T:

$$\tilde{O}\left(\frac{(T + q\ell)^2}{2^c} + \sqrt{\frac{S(T + q\ell)^2}{2^c}}\right)$$

5 Related Work

[CDG18] proved tight (but impractical) bounds of $\Theta(ST^2/2^c + T^2/2^r)$ for the advantage of finding large collisions of length $\Omega(T)$ in the sponge construction, as well as an optimal attack with $\Theta(T^2/2^c + T^2/2^r)$ for finding short hashing collisions of $B = 1$ blocks in the Merkle-Damgård (MD) paradigm. Along with the observation that shorter collisions are harder to find, these results inspired a very recent paper which used bit-fixing in the AI-RPM model as well as compression techniques to provide more optimal attacks and advantage bounds for finding (small) B-block sponge collisions [FGK22]. The authors did not close the gap between the best attacks and upper bounds on advantage, but suggested sponge compression techniques to further tighten advantage bounds for $B = 1, 2$, noting that short collisions such as these are much more difficult to analyze and produce tight bounds for.

On the other hand, Daemen et al. [DMA17] consider an adaptive multi-user scenario that demonstrates strong security bounds for duplex constructions such as Ascon AEAD, accounting for adversaries either misusing or respecting nonces. However, no formal analyses we could find on the security of duplex constructions looked at the impact of preprocessing attacks or time-space tradeoffs analogous to [FGK22] for Ascon (et al.) sponge hashing. It is well-known that duplex constructions can be easily simulated by evaluating sponge functions on the same parameters [BDPV12a] though, which motivated the work in this paper as an easy way to extend analyses of sponge constructions to duplex constructions.

Lastly, Ascon's AEAD construction is incredibly similar to that of MonkeyDuplex [BDPV12b], with the primary difference that Ascon uses stronger keyed initialization and finalization stages to prevent leakage of internal state from revealing the key or winning the unpredictability game. In our work, we wanted to gain some insight into why Ascon fared better compared to e.g. MonkeyDuplex from the perspective of nonce reuse / misuse.

6 Future Work

The work we presented in this report involved an exploratory analysis into various aspects of nonce collision and preprocessing attacks on duplex constructions which—to the best of our knowledge—were not comprehensively explored by the literature, especially in relation to the Ascon AEAD cryptosystem. With this in mind, there are other approaches which we had considered for future analysis but did not have time to address:

1. Based on prior work analyzing the the security of Ascon AEAD and other duplex constructions under more standard assumptions, we considered adapting preprocessing analyses to duplexes directly—ala [CDG18] with sponges—without the intermediate step of simulating the duplex with a cascade of sponges.

2. Use a similar technique to analyze the preprocessing security of various other cryptographic primitives and cryptosuites that use duplexes besides Ascon and Keccak, as they not only work for AEAD but also PRFs, PRNGs, etc. One particular area which we noticed didn't receive much attention in the literature are constructions which employ more general *transformations* and not just permutations.

3. One could attempt to produce tighter bounds in the preprocessing security model by leveraging compression techniques instead of using the bit-fixing and auxiliary input models. However, we concur with the observations noted by [CDG18] that analyses using this technique may not transfer as broadly into other schemes which require additional security assumptions (e.g., discrete-log) and are likely much more difficult.

References

[BDPV07] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge functions. Ecrypt Hash Workshop 2007, 2007. `https://keccak.team/files/SpongeFunctions.pdf`.

[BDPV08] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 181–197. Springer, Heidelberg, April 2008.

[BDPV10] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge-based pseudo-random number generators. In Stefan Mangard and François-Xavier Standaert, editors, *CHES 2010*, volume 6225 of *LNCS*, pages 33–47. Springer, Heidelberg, August 2010.

[BDPV11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Cryptographic sponge functions, v0.1, 2011. `https://keccak.team/files/CSF-0.1.pdf`.

[BDPV12a] 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, *SAC 2011*, volume 7118 of *LNCS*, pages 320–337. Springer, Heidelberg, August 2012.

[BDPV12b] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Permutation-based encryption, authentication and authenticated encryption. DIAC 2012, 2012. `https://keccak.team/files/KeccakDIAC2012.pdf`.

[BDPV12c] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak Reference, 2012. `https://keccak.team`.

[BKL+11] Andrey Bogdanov, Miroslav Knežević, Gregor Leander, Deniz Toz, Kerem Varici, and Ingrid Verbauwhede. Spongent: A lightweight hash function. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES 2011*, volume 6917 of *LNCS*, pages 312–325. Springer, Heidelberg, September / October 2011.

[CDG18] Sandro Coretti, Yevgeniy Dodis, and Siyao Guo. Non-uniform bounds in the random-permutation, ideal-cipher, and generic-group models. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 693–721. Springer, Heidelberg, August 2018.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th ACM STOC*, pages 209–218. ACM Press, May 1998.

[CHS19] Jan Czajkowski, Andreas Hülsing, and Christian Schaffner. Quantum indistinguishability of random sponges. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 296–325. Springer, Heidelberg, August 2019.

[DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2: Lightweight authenticated encryption and hashing. *Journal of Cryptology*, 34(3):33, July 2021.

[DJS19] Jean Paul Degabriele, Christian Janson, and Patrick Struck. Sponges resist leakage: The case of authenticated encryption. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part II*, volume 11922 of *LNCS*, pages 209–240. Springer, Heidelberg, December 2019.

[DMA17] Joan Daemen, Bart Mennink, and Gilles Van Assche. Full-state keyed duplex with built-in multi-user support. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 606–637. Springer, Heidelberg, December 2017.

[EDMS] Maria Eichlseder, Christoph Dobraunig, Florian Mendel, and Martin Schläffer. Ascon specification. `https://ascon.iaik.tugraz.at/specification.html`.

[FGK22] Cody Freitag, Ashrujit Ghoshal, and Ilan Komargodski. Time-space tradeoffs for sponge hashing: Attacks and limitations for short collisions. Cryptology ePrint Archive, Paper 2022/1009, 2022. `https://eprint.iacr.org/2022/1009`.

[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, *TCC 2004*, volume 2951 of *LNCS*, pages 21–39. Springer, Heidelberg, February 2004.

[Nai16] Yusuke Naito. Sandwich construction for keyed sponges: Independence between capacity and online queries. In Sara Foresti and Giuseppe Persiano, editors, *CANS 16*, volume 10052 of *LNCS*, pages 245–261. Springer, Heidelberg, November 2016.

[Saa14] Markku-Juhani O. Saarinen. Beyond modes: Building a secure record protocol from a cryptographic sponge permutation. In Josh Benaloh, editor, *CT-RSA 2014*, volume 8366 of *LNCS*, pages 270–285. Springer, Heidelberg, February 2014.

[SY15] Yu Sasaki and Kan Yasuda. How to incorporate associated data in sponge-based authenticated encryption. In Kaisa Nyberg, editor, *CT-RSA 2015*, volume 9048 of *LNCS*, pages 353–370. Springer, Heidelberg, April 2015.

[Unr07] Dominique Unruh. Random oracles and auxiliary input. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 205–223. Springer, Heidelberg, August 2007.