Sponges Resist Leakage: The Case of Authenticated Encryption*

Jean Paul Degabriele[1], Christian Janson[2], and Patrick Struck[3]

[1] CNS, Technische Universität Darmstadt, Germany
jeanpaul.degabriele@cysec.de
[2] Cryptoplexity, Technische Universität Darmstadt, Germany
christian.janson@cryptoplexity.de
[3] CDC, Technische Universität Darmstadt, Germany
pstruck@cdc.tu-darmstadt.de

Abstract. In this work we advance the study of leakage-resilient Authenticated Encryption with Associated Data (AEAD) and lay the theoretical groundwork for building such schemes from sponges. Building on the work of Barwell et al. (ASIACRYPT 2017), we reduce the problem of constructing leakage-resilient AEAD schemes to that of building fixed-input-length function families that retain pseudorandomness and unpredictability in the presence of leakage. Notably, neither property is implied by the other in the leakage-resilient setting. We then show that such a function family can be combined with standard primitives, namely a pseudorandom generator and a collision-resistant hash, to yield a nonce-based AEAD scheme. In addition, our construction is quite efficient in that it requires only two calls to this leakage-resilient function per encryption or decryption call. This construction can be instantiated entirely from the T-sponge to yield a concrete AEAD scheme which we call Slae. We prove this sponge-based instantiation secure in the non-adaptive leakage setting. Slae bears many similarities and is indeed inspired by Isap, which was proposed by Dobraunig et al. at FSE 2017. However, while retaining most of the practical advantages of Isap, Slae additionally benefits from a formal security treatment.

Keywords: AEAD · Leakage Resilience · Side Channels · SLAE · ISAP

1 Introduction

The oldest and most fundamental application of cryptography is concerned with securing the communication between two parties who already share a secret key. The modern cryptographic construct for this application is authenticated encryption with associated data (AEAD), which was the topic of the recent CAESAR competition [6]. Most of the effort in this competition has been directed towards exploring new designs, optimising performance, and offering robust security guarantees. However, there has not been much progress in the development of AEAD constructions that, by design, protect against side-channel attacks. This is a challenging problem that is likely to become a primary focus in the area of AEAD design.

Recently, a handful of AEAD designs with this exact goal have emerged. Each of these is based on a different approach with varying trade-offs between complexity, efficiency, and security guarantees. One notable example is the work of Barwell et al. [4], which proposes AEAD constructions with strong security guarantees but pays a relatively high price in terms of complexity and efficiency. Specifically, their constructions achieve security against adaptive leakage but resort to elliptic-curve pairings and secret sharing in order to realise implementations of a leakage-resilient MAC and a leakage-resilient pseudorandom function (employed in a block-wise fashion for encryption) for instantiating their scheme. A more hands-on approach was adopted by Dobraunig et al. in the design of their proposed AEAD scheme Isap. It was conceived with the intent to protect against Differential Power Analysis (DPA) [12]. Isap is entirely sponge-based and follows a fairly conventional design, augmented with a rekeying strategy. Arguably, this simpler approach, employing readily-available symmetric primitives, is more likely to lead to a pragmatic solution. However, Isap's design rationale is predominantly heuristic, lacking any formal security analysis to justify its claims. As such the efficacy of Isap's approach in resisting side-channel attacks is unclear, both qualitatively and quantitatively, curtailing any objective comparison with the constructions from [4] and others.

In light of the practical advantages that the sponge-based approach offers, we remedy this state of affairs as follows. We propose SLAE, a derivative of ISAP which retains its main structure and benefits but includes certain modifications to admit a formal security proof. We analyse its security in the framework of leakage-resilient cryptography introduced by Dziembowski and Pietrzak [16], adapted to the random transformation model. Specifically, we prove it secure with respect to the leakage-resilient AEAD definition, put forward in [4] by Barwell et al., in the non-adaptive leakage setting. That is, we assume a leakage function that is fixed a priori and whose output is limited to some number of bits λ.

Admittedly, SLAE achieves qualitatively weaker security than the schemes of Barwell et al., since it only achieves non-adaptive leakage resilience. Nevertheless, we contend that SLAE strikes a more pragmatic balance by improving on efficiency and ease of implementation while still benefiting from a provably-secure design. Indeed, several other works [1, 15, 17, 25, 27] have settled for and argued that non-adaptive leakage security often suffices in practice. Moreover, as discussed in [27], the syntax of primitives like pseudorandom functions makes adaptive-leakage security impossible to achieve. In fact Barwell et al. achieve security against adaptive leakage by resorting to a specialised implementation of a pseudorandom function which requires an additional random input per invocation. In contrast, SLAE adheres to the standard nonce-based AEAD syntax and requires no source of randomness.

When viewed as sponge-based constructions, SLAE and ISAP look very similar and we do not claim any particular novelty in that respect. Nevertheless, the rationale behind their design is rather different. ISAP was conceived as augmenting a standard sponge-based AEAD design with a rekeying strategy, where the rekeying function is in turn also built from sponges, followed by some optimisations. The rekeying is intended to frustrate Differential Power Analysis (which requires several power traces on the same key but distinct inputs) by running the AEAD scheme with a distinct session key each time its inputs change. In turn, the session key is produced by combining a hash of the inputs and the master key through a rekeying function. Ostensibly, the rekeying function is itself strengthened against DPA by reducing its input data complexity through a low sponge absorption rate. In contrast SLAE is understood through a top-down design where we gradually decompose a leakage-resilient AEAD scheme into smaller components which we then instantiate using sponges. In particular there is no mention of rekeying or session keys. Note that there is more to this distinction than mere renaming. For instance, if we compare the MAC components in ISAP and SLAE we notice that the same value that serves as the MAC session key in ISAP is used directly as the MAC tag in SLAE.

At a more general level, the key premise made in [12] is that sponges offer a promising and practical solution to protect against side-channel attacks. Our work serves to provide formal justification to this claim and allows one to calculate concrete parameters for a desired security level.

1.1 Contribution

Below is an outline of our contributions highlighting how we improve on prior works and some of the challenges we face in our analysis.

A Generic Construction (FGHF′). The composition theorem in [4] reconsiders the N2 construction from [22] in the setting of leakage resilience. Specifically they show that given a MAC that is both *leakage-resilient strongly unforgeable* and a *leakage-resilient pseudorandom function*, together with an encryption scheme that is *leakage-resilient against augmented chosen plaintext attacks*, the N2 construction yields a leakage-resilient AEAD scheme. We extend this result, in the non-adaptive setting, by further decomposing the MAC and the encryption scheme into simpler lower-level primitives, ultimately giving rise to the FGHF′ construction. In turn this constructs a leakage-resilient AEAD scheme from two *fixed-size* leakage-resilient functions $\mathcal{F}$ and $\mathcal{F}'$, a standard pseudorandom generator $\mathcal{G}$, and a collision-resistant vector hash $\mathcal{H}$. The construction requires that both $\mathcal{F}$ and $\mathcal{F}'$ be leakage-resilient pseudorandom functions and that $\mathcal{F}'$ additionally be a leakage-resilient unpredictable function. The latter is a notion that we introduce.

As pointed out in [4], in the adaptive leakage setting any MAC whose verification algorithm recomputes the tag and checks for equality with the candidate tag, simply cannot be strongly unforgeable. They overcome this issue through an ingenious MAC implementation. However this requires three pairing evaluations per verification and a source of randomness. In the FGHF′ construction we show that by settling for non-adaptive leakage security the canonical MAC construction, which recomputes the tag and checks for equality, can be rescued. Specifically, we show that any leakage-resilient unpredictable function gives rise to a canonical MAC which is strongly unforgeable. In contrast to the leakage-free setting, not every pseudorandom function is an unpredictable function. This has to do with the fact that in unpredictability

we give the adversary more freedom in what it can query to its oracles, which is in turn a necessary requirement for composition to hold. In addition, we prove that one can combine a collision-resistant hash function with fixed-input-length leakage-resilient pseudorandom and unpredictable functions to obtain corresponding primitives with extended input domains.

For the encryption part, Barwell et al. use Counter Feedback Mode instantiated with a leakage-resilient pseudorandom function and an additional extra call to generate the initial vector from the nonce. Thus multiple calls to the leakage-resilient pseudorandom function are required for each encryption call. In contrast we show that to meet the required security notion, one can do with just one call to the leakage-resilient pseudorandom function and a pseudorandom generator, thereby resulting in a considerably more efficient scheme. Thus, if one is content with non-adaptive leakage security then the FGHF$'$ construction constitutes a simpler recipe yielding a more efficient AEAD scheme.

All the results needed to prove the security of the FGHF$'$ construction hold in the general adaptive setting. The limitation to the non-adaptive leakage setting comes from the fact that leakage-resilient unpredictable functions are unattainable in the adaptive-leakage setting if no further restriction is imposed on the set of leakage functions.

Non-Adaptively Leakage-Resilient Functions from Sponges. Having reduced the task of constructing a leakage-resilient AEAD scheme to that of constructing suitable leakage-resilient function families, we turn our attention to the latter problem. We instantiate both $\mathcal{F}$ and $\mathcal{F}'$ with the same sponge-based construction, which we refer to as SlFunc. This construction is essentially the rekeying function employed in Isap [12] instantiated with a random transformation (T-sponge) instead of a random permutation (P-sponge). In [12] this was proposed without proof, instead its security was argued based on its apparent similarity to the GGM construction [18] and the corresponding results in [17,25] for it yielding a leakage-resilient pseudorandom function family. However, there are clear differences between the sponge construction and the GGM construction and we do not see a way to make a direct connection between the security of the two. In fact our proof follows a fairly different strategy from the ones presented in [17, 18, 25] – which all rely on a hybrid argument whereas ours does not. Moreover, for the overall security of Slae we need this function family to additionally be leakage-resilient unpredictable, which, as was discussed above, does not follow from it being leakage-resilient pseudorandom. We show that this construction achieves both security notions when the absorption rate is set to a sufficiently small value.

Another technical challenge that we face here is that we cannot employ the H-coefficient technique which is commonly used to prove the security of various sponge-based constructions. Like most other works on leakage resilience, we resort to arguments based on min-entropy and its chain rule in order to deal with leakage. Unfortunately, such arguments do not combine well with the H-coefficient technique, which precludes us from using it. In turn, this renders the security proof more challenging, as we have to deal with an adversary that may choose its queries (not the leakage function) adaptively. In contrast, the H-coefficient technique would automatically bypass this issue by reducing the security proof to a counting problem.

A Concrete Sponge-Based AEAD Scheme (Slae). Finally, by instantiating the FGHF$'$ construction with the above sponge-based construction for $\mathcal{F}$ and $\mathcal{F}'$ and matching sponge-based constructions for $\mathcal{G}$ and $\mathcal{H}$ we obtain Slae. We also present security proofs for the T-sponge instantiations of the pseudorandom generator and the vector hash, which we were unable to readily find in the literature. Slae is perhaps our most practical contribution – an entirely sponge-based leakage-resilient nonce-based AEAD scheme with provable security guarantees that is simple to implement and reasonably efficient. The efficiency of Slae could be further optimised using similar techniques to the ones described in [12] for Isap. Furthermore our security proofs are conducted in the concrete security setting thereby allowing practitioners to easily derive parameter estimates for their desired security level.

1.2 Related Work

To the best of our knowledge, the first authenticated encryption scheme claimed to be leakage-resilient was RCB [3], but it was broken soon after [2].

A series of works [7, 8, 19, 23] have proposed a number of leakage-resilient symmetric encryption schemes, message authentication codes, and authenticated encryption schemes. These constructions assume that a subset of their components (block cipher instances) are leakage-free and that the leakage

in the other components is simulatable, an assumption that is somewhat contentious [21, 26]. Based on these assumptions, they show that the security of their encryption schemes reduces to the security of a single-block variant of the same scheme. However, the security of the corresponding single-block schemes remains an open question that is implicitly assumed to hold.

Abdalla, Belaïd, and Fouque [1] construct a symmetric encryption scheme that is non-adaptively leakage-resilient against chosen-plaintext attacks. Interestingly, their scheme employs a rekeying function that is not a leakage-resilient pseudorandom function. However their encryption scheme is not nonce-based as it necessitates a source of randomness.

In independent and concurrent work [13] Dobraunig and Mennink analyse the leakage resilience of the duplex sponge construction. While their leakage model is closer to ours, they prove something different. Namely they show that the duplex is indistinguishable from an *adjusted ideal extendible input function* (AIXIF) which is an ideal functionality incorporating leakage. In contrast we show that SLFUNC is both a leakage-resilient PRF (LPRF) and a leakage-resilient unpredictable function (LUF), and then leverage this to construct a leakage-resilient AEAD scheme.

Other independent and concurrent work by Guo et al. [20] proposes an AEAD design, TETSPonge, that combines a sponge construction with two tweakable block cipher instances. While their work and ours share the goal of constructing leakage-resilient AEAD schemes, the two works adopt very different approaches. Both the security definitions and the assumptions on which the security of the schemes rely on are significantly different. One notable difference, is that the leakage resilience of TETSPonge relies crucially on the tweakable block cipher instances being leak-free, presumably due to a hardened implementation, whereas our treatment exploits and exposes the inherent leakage resilience of the sponge construction.

1.3 Organization of the Paper

In Section 2 we review the basic concepts and security definitions that we require in the rest of the paper. This is followed by a detailed description of SLAE in Section 3. In Section 4 we cover the security analysis of the generic FGHF′ construction. In Section 5 we cover the security of the sponge-based primitives used to instantiate FGHF′ and thereby obtain SLAE. We conclude in Section 6 with some remarks on implementing SLAE. The full details of the proofs can be found in the Appendix.

2 Preliminaries

We start by reviewing the basic tools and definitions that we require for our results. We begin by establishing some notation. Some additional standard definitions appear in Appendix A.

2.1 Notation

For any non-negative integer $n \in \mathbb{N}$ we use $[n]$ to denote the set $\{1, \ldots, n\}$, where $[n] = \emptyset$ when $n = 0$. For any two strings s_1 and s_2, $|s_1|$ denotes the size of s_1 and $s_1 \| s_2$ denotes their concatenation. For a positive integer $k \leq |s_1|$, we use $\lfloor s_1 \rfloor_k$ to denote the string obtained by truncating s_1 to its leftmost k bits. The empty string is denoted by ε, $\{0,1\}^n$ denotes the set of bit strings of size n, and $\{0,1\}^*$ denotes the set of all strings of finite length. We write $x \twoheadleftarrow \mathcal{S}$ to denote the process of uniformly sampling a value from the finite set $\mathcal{S}$ and assigning it to x.

We make use of the code-based game-playing framework by Bellare and Rogaway [5], where the interaction between a game and the adversary is implicit. In all games, the adversary is given as its input the output of the initialize procedure, it has oracle access to the other procedures described in the game, and its output is fed into the finalize procedure. The output of the finalize procedure is the output of the game. For a game G and an adversary $\mathcal{A}$, $\mathsf{G}^{\mathcal{A}} \Rightarrow y$ denotes the event that G outputs y when interacting with $\mathcal{A}$. Similarly, $\mathcal{A}^{\mathsf{G}} \Rightarrow x$ denotes the event that $\mathcal{A}$ outputs x when interacting with G. By convention all boolean variables Bad are initialized to false, and for any table $p[\,]$ its entries are all initialized to $\perp$. When lazy-sampling a random function with domain $\mathcal{X}$ and co-domain $\mathcal{Y}$ into a table $p[\,]$, we use $\mathsf{inset}(p)$ and $\mathsf{outset}(p)$ to denote respectively the sets of input and output values defined up to that point. That is, $\mathsf{inset}(p) = \{X : p[X] \neq \perp \wedge X \in \mathcal{X}\}$ and $\mathsf{outset}(p) = \{p[X] : p[X] \neq \perp \wedge X \in \mathcal{X}\}$. If G_1 and G_2 are games and $\mathcal{A}$ is an adversary we define the corresponding *adversarial advantage* as

$$\mathbf{Adv}\left(\mathcal{A}^{\mathsf{G}_1}, \mathcal{A}^{\mathsf{G}_2}\right) = \Pr[\mathcal{A}^{\mathsf{G}_1} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathsf{G}_2} \Rightarrow 1]\,,$$

and the corresponding *game advantage* as

$$\mathbf{Adv}\left(\mathsf{G}_1^{\mathcal{A}}, \mathsf{G}_2^{\mathcal{A}}\right) = \Pr[\mathsf{G}_1^{\mathcal{A}} \Rightarrow \mathsf{true}] - \Pr[\mathsf{G}_2^{\mathcal{A}} \Rightarrow \mathsf{true}]\,.$$

We operate in the random transformation model, where ρ is an idealised random transformation mapping n-bit strings to n-bit strings. For any algorithm $\mathcal{F}$ that uses ρ as a subroutine, we use $Q_{\mathcal{F}}(q, \mu)$ to denote the number of calls to ρ required when evaluating $\mathcal{F}$ q times on a total of μ bits.

2.2 Syntax

Encryption. An *authenticated encryption scheme with associated data* $\textsc{Aead} = (\mathcal{E}, \mathcal{D})$ is a pair of efficient algorithms such that:

- The deterministic encryption algorithm $\mathcal{E} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \to \{0,1\}^*$ takes as input a secret key K, a nonce N, associated data A, and a message M to return a ciphertext C.
- The deterministic decryption algorithm $\mathcal{D} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \{0,1\}^* \to \mathcal{M} \cup \{\perp\}$ takes as input a secret key K, a nonce N, associated data A, and a ciphertext C to return either a message in $\mathcal{M}$ or $\perp$ indicating that the ciphertext is invalid.

Sets $\mathcal{K}$, $\mathcal{N}$, $\mathcal{A}$, and $\mathcal{M}$ denote respectively the key space, the nonce space, the associated data space, and the message space associated to the scheme. We assume throughout that $\mathcal{E}$ and $\mathcal{D}$ are never queried on inputs outside of these sets. An authenticated encryption scheme is required to be *correct* and *tidy*. Correctness requires that for all K, N, A, M if $\mathcal{E}(K, N, A, M) = C$ then $\mathcal{D}(K, N, A, C) = M$. Analogously, tidiness requires that for all K, N, A, C if $\mathcal{D}(K, N, A, C) = M \neq \perp$ then $\mathcal{E}(K, N, A, M) = C$. Furthermore we demand that encryption be length regular, i.e for all K, N, A, M it should hold that $|\mathcal{E}(K, N, A, M)|$ is entirely determined by $|N|$, $|A|$, and $|M|$.

We will use the terms *authenticated encryption scheme* and *symmetric encryption scheme* to refer to the analogously defined encryption scheme which does not admit associated data as part of its input. For such schemes, A is implicitly set to the empty string in the security games.

Message Authentication. A *message authentication code* $\textsc{Mac} = (\mathcal{T}, \mathcal{V})$ is a pair of efficient algorithms with an associated key space $\mathcal{K}$, domain $\mathcal{X}$, and tag length t such that:

- The deterministic tagging algorithm $\mathcal{T} : \mathcal{K} \times \mathcal{X} \to \{0,1\}^t$ takes as input a key K and a value X to return a tag T of size t.
- The deterministic verification algorithm $\mathcal{V} : \mathcal{K} \times \mathcal{X} \times \{0,1\}^t \to \{\top, \perp\}$ takes as input a key K, a value X, and a tag T to return either $\top$ indicating a valid input or $\perp$ otherwise.

We require that for any key $K \in \mathcal{K}$ and any admissible input $X \in \mathcal{X}$, if $T \leftarrow \mathcal{T}(K, X)$, then $\mathcal{V}(K, X, T) = \top$. When $\mathcal{X} = \{0,1\}^*$ we end up with the usual MAC definition, however we will also consider MACs over tuples of strings, e.g. $\mathcal{X} = \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^*$. Such MACs where considered in [22] and we follow suit in referring to such MACs as vector MACs.

We say that a MAC is *canonical* if it is implicitly defined by $\mathcal{T}$, where $\mathcal{V}(K, X, T)$ consists of running $T' \leftarrow \mathcal{T}(K, X)$ and returning $\top$ if $T' = T$ and $\perp$ otherwise.

2.3 The Sponge Construction

The sponge construction is a versatile object that can be used to realise various cryptographic primitives. Several variations of the sponge exist, Fig. 1 illustrates the plain version of the sponge as originally introduced by Bertoni et al. [9]. We give here only a brief overview of its operation and the associated nomenclature that we will use throughout this paper.

The sponge operates iteratively on its inputs through a transformation ρ, and generally includes an *absorbing* phase and a *squeezing* phase. The transformation ρ maps strings of size n to strings of size n. Associated to the sponge are two other values called the rate r and the capacity c, where $n = r + c$. At any given iteration we refer to the output of the transformation as the state, which we denote by S. Furthermore, we denote the leftmost r bits of S by $\bar{S}$ and the remaining c bits by $\hat{S}$. We will at times refer to $\bar{S}$ and $\hat{S}$ as the outer and inner parts of the state, respectively. In the absorbing phase an input M is "absorbed" iteratively r bits at a time. At iteration i input M_i is absorbed by letting $Y_i \leftarrow (M_i \oplus \bar{S}_i) \parallel \hat{S}_i$ and setting $S_{i+1} \leftarrow \rho(Y_i)$. The initial value of S may generally be set to a constant, a concatenation of a secret key and a constant, or by applying the transformation to either of these values.

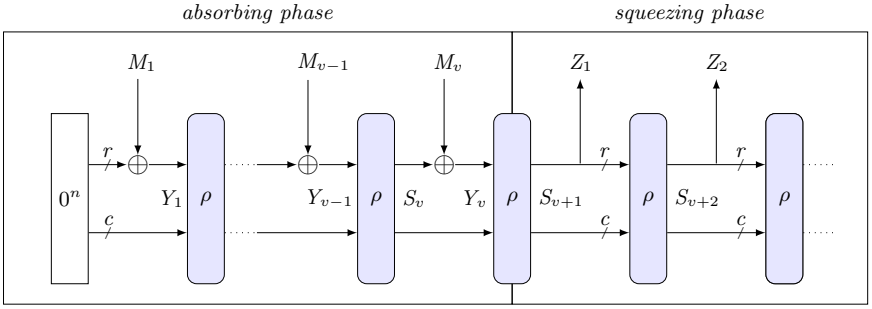

Fig. 1: Illustration of the plain sponge construction.

Output is produced from the sponge during the squeezing phase in one or more iterations, r bits at a time. At iteration i output Z_j is produced by setting $Z_j \leftarrow \bar{S}_i$ and $S_{i+1} \leftarrow \rho(S_i)$. The above variant is normally referred to as the T-sponge, as it employs a fixed-size random transformation. An alternative instantiation, known as the P-sponge, replaces this random transformation with a random permutation.

2.4 The Leakage Model

Our leakage model is based on *leakage resilience* as defined in [16]. This assumes that only computation leaks, and in particular, that only the data that is accessed during computation can leak information. It allows for continuous adaptive leakage, where in each query to a leakage oracle the adversary can specify a leakage function from some predefined set $\mathcal{L}$ that it can chose adaptively based on prior outputs and leakage. Throughout, we restrict ourselves to leakage functions that are deterministic and efficiently computable. While our security definitions are formulated in this general setting, our main results will be in the weaker *granular non-adaptive* leakage setting proposed in [17]. We view the non-adaptive leakage setting as the special case where the leakage set $\mathcal{L}$ is restricted to be a singleton, fixed at the start of the game. In granular leakage, a single time step is with respect to a single computation of some underlying primitive, in our case, the transformation ρ. Correspondingly, in this case the adversary specifies a vector of leakage functions and gets in return the aggregate leakage from the entire evaluation of the higher-level construction. Note that in the granular setting the leakage sets for each iteration can be distinct. Similarly, when studying the leakage resilience of composite constructions we have to consider compositions of leakage functions. For instance, if construction C is composed of primitives A and B with associated leakage sets $\mathcal{L}_A$ and $\mathcal{L}_B$, then we associate to C the Cartesian product of the two leakage sets, i.e. $\mathcal{L}_C = \mathcal{L}_A \times \mathcal{L}_B$. The actual inputs that get fed to the leakage functions are implicitly defined by the construction and its inputs, whereas the combined output is the aggregate output of all function evaluations.

An analysis of sponge-based constructions compels us to consider leakage resilience in the random transformation model. A similar setting, albeit in the random oracle model, was already considered by Standaert et al. in [25]. A central question that arises in idealised settings like this is whether the leakage function should be given access to the ideal primitive. As in [25], we will not give this access to the leakage function. On the one hand, providing the leakage function with unlimited access to the random oracle gives rise to artificial attacks, such as the "future computation attack" discussed in [25], that would not arise in practice. On the other hand, depriving the leakage function from accessing the ideal primitive, means that the leakage function cannot leak any bits of the ideal primitive's output, which may seem overly restrictive. However, for the case of sponge-based constructions this is less problematic because from the adversary's perspective the full output of a transformation call is completely determined by the input to the next transformation call. As such, information about the output of one transformation call can leak as part of the leakage in the next transformation call. Combined with the fact that the only restriction that we will impose on the leakage function is to limit its output length, we think that this leads to a fairly realistic leakage model.

We conclude our discussion on the leakage model by offering our interpretation of the significance of leakage resilience security with respect to practical side channel attacks. One might object that we model leakage by a deterministic function whose output is of a fixed bit-length whereas in practice the leakage is noisy. However through the leakage function we are really trying to capture the maximum amount of information that an adversary may obtain from evaluating the scheme on a single input. Hence, the underlying assumption is that no matter how many times the scheme is run on the same

input, in order to even out the noise, the information that the adversary can obtain is limited. Put in more practical terms, this roughly translates to assuming that the scheme's implementation resists Simple Power Analysis (SPA). On the other hand, if the scheme is proven to be leakage-resilient then we are guaranteed that an adversary cannot do much better even if it can observe and accumulate leakage on multiple other (differing) inputs. Thus a proof of leakage resilience can be interpreted as saying that if the scheme's implementation is secure against SPA then, by the inherent properties of the scheme, it is also secure against Differential Power Analysis (DPA). However, a proof of leakage resilience is of course no guarantee that a scheme's implementation will be secure against SPA.

2.5 Authenticated Encryption and Leakage Resilience

Recently, Barwell et al. [4] provided a definitional framework augmenting nonce-based authenticated encryption with leakage. Their security notions capture the leakage resilience setting as defined in [16]. Furthermore, they prove composition theorems analogous to [22] that additionally take leakage into account. Below we reproduce their security definitions and composition result which we will employ in this work, with some minor adaptations. We recast their definitions in a style that admits code-based proofs [5]. Unlike [4] we make no distinction between a scheme and its implementation since we are interested in proving security for the actual scheme. When defining these security notions, we only describe the game and the corresponding adversarial advantage. A scheme is understood to be secure if the adversarial advantage is bounded by a sufficiently small value for all reasonably-resourced adversaries. Our security theorems will then establish a bound on the adversarial advantage in terms of the adversary's resources, without drawing judgement as to what constitutes "small" and "reasonable" since that is a rather subjective matter.

Classifying Adversarial Queries. As usual, the adversary has to be forbidden from making certain queries in order to avoid trivial win conditions. Following the terminology of [4], if an adversary makes a query (N, A, M) to an encryption oracle that returns C, then repeating this query to one of the encryption oracles or querying (N, A, C) to one of the decryption oracles, is considered to be an *equivalent* query. Note that any additional components of a query, such as the leakage function, are ignored for the purpose of determining equivalence between two queries. If an adversary makes equivalent queries across two oracles, it is said to *forward* that query from one oracle to the other. Note that the two oracles do not need to be distinct, and thus forwarded queries include repeated queries to the same oracle.[4]

Let an encryption query refer to any query made to either a challenge encryption oracle or a leakage encryption oracle. Then an adversary against an (authenticated) encryption scheme is said to be *nonce respecting* if it never repeats a nonce in two distinct encryption queries.

Chosen-Plaintext Security with Leakage. Barwell et al. introduce an augmented variant of leakage-resilient chosen-plaintext security called IND-aCPLA, that is required by their composition theorem. Here the adversary is given access to three oracles. A challenge oracle that returns either a valid encryption of a message or a random string of appropriate length. A leakage encryption oracle that, upon being queried on a message and a leakage function, returns the corresponding ciphertext and the evaluated leakage. The adversary is not allowed to forward queries between the two encryption oracles. In addition, it has limited access to a leakage decryption oracle which returns the decryption of the queried ciphertext and the leakage corresponding to the queried leakage function. However, it can only query this oracle on inputs forwarded from the leakage encryption oracle. Thus the adversary can obtain decryption leakage, but only on ciphertexts for which it already knows the corresponding message. Below is the formal definition.

Definition 1 (IND-aCPLA **Security).** *Let* $\mathrm{SE} = (\mathcal{E}, \mathcal{D})$ *be a symmetric encryption scheme and the* INDaCPLA *game be as defined in Fig. 2. Then for any nonce-respecting adversary* $\mathcal{A}$ *that never forwards queries to or from the* `Enc` *oracle, only makes queries to* `LDec` *that are forwarded from* `LEnc`*, and only makes encryption and decryption queries containing leakage functions in the respective sets* $\mathcal{L}_E$ *and* $\mathcal{L}_D$*, its corresponding* IND-aCPLA *advantage is given by:*

$$\mathbf{Adv}^{\text{ind-acpla}}_{\mathrm{SE}}(\mathcal{A}, \mathcal{L}_E, \mathcal{L}_D) = 2\Pr\left[\mathsf{INDaCPLA}^{\mathcal{A}} \Rightarrow \mathsf{true}\right] - 1\,.$$

[4] This is not really required, since contrary to [4] the challenge oracles are not forgetful in our case. Nevertheless we conform to the original definition of forwarded queries.

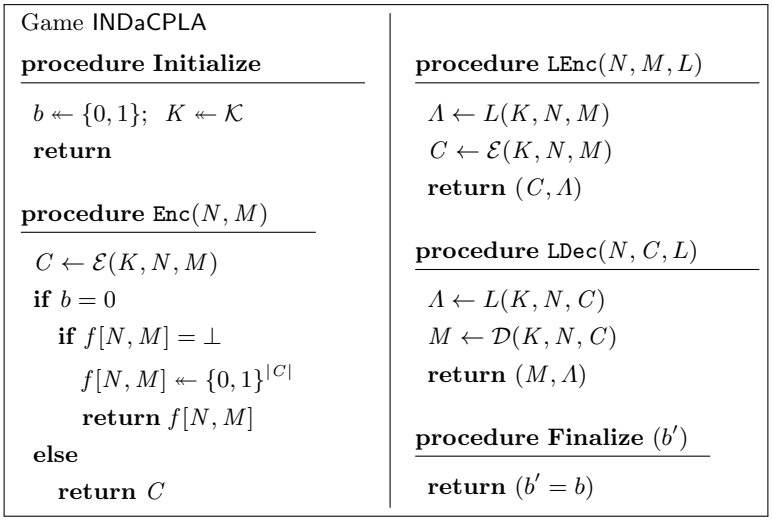

Fig. 2: Game used to define IND-aCPLA security.

Leakage-Resilient Function Families. We will distinguish among function families based on their domain $\mathcal{X}$. We will use the terms *fixed-input-length* function when $\mathcal{X} = \{0,1\}^l$ for some $l \in \mathbb{N}$, *variable-input-length* function when $\mathcal{X} = \{0,1\}^*$, and *vector* function when the domain is a cartesian product of string sets, e.g. $\mathcal{X} = \{0,1\}^* \times \{0,1\}^*$.

For such function families we will consider two security notions: leakage-resilient pseudorandom functions (LPRF) and leakage-resilient unpredictable functions (LUF). While LPRF security is well-established in the literature, LUF security is new. Below are the formal definitions. Note that no restriction is made on the adversary's queries in the LUF security definition.

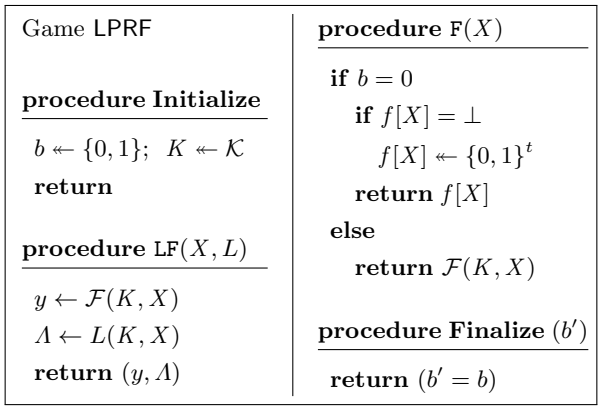

Fig. 3: Game used to define LPRF security.

Definition 2 (LPRF **Security).** *Let $\mathcal{F}\colon \mathcal{K} \times \mathcal{X} \to \{0,1\}^t$ be a function family over the domain $\mathcal{X}$ and indexed by $\mathcal{K}$, and the* LPRF *game be as defined in Fig. 3. Then for any adversary $\mathcal{A}$ that never forwards queries to or from the* F *oracle and only queries leakage functions in the set $\mathcal{L}_F$, its corresponding* LPRF *advantage is given by:*

$$\mathbf{Adv}^{\mathrm{lprf}}_{\mathcal{F}}(\mathcal{A}, \mathcal{L}_F) = 2\Pr\left[\mathsf{LPRF}^{\mathcal{A}} \Rightarrow \mathsf{true}\right] - 1\,.$$

Definition 3 (LUF **Security).** *Let $\mathcal{F}\colon \mathcal{K} \times \mathcal{X} \to \{0,1\}^t$ be a function family over the domain $\mathcal{X}$ and indexed by $\mathcal{K}$, and the* LUF *game be as defined in Fig. 4. Then for any adversary $\mathcal{A}$ its corresponding* LUF *advantage is given by:*

$$\mathbf{Adv}^{\mathrm{luf}}_{\mathcal{F}}(\mathcal{A}, \mathcal{L}_F) = \Pr\left[\mathsf{LUF}^{\mathcal{A}} \Rightarrow \mathsf{true}\right].$$

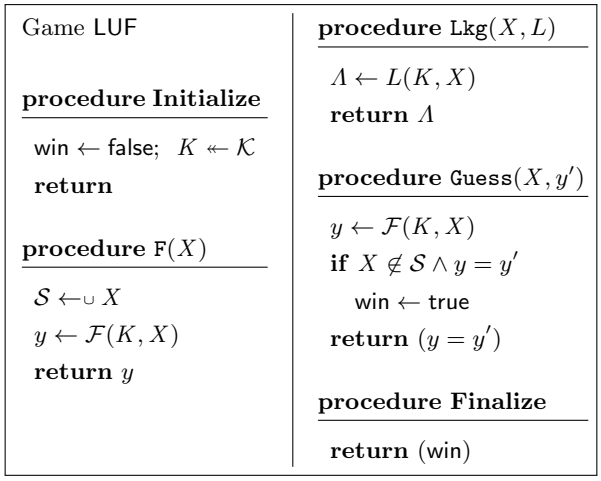

Fig. 4: Game used to define LUF security.

Unforgeability in the Presence of Leakage. For message authentication we will require the analogue of strong unforgeability in the leakage setting (SUF-CMLA) put forth in [4]. This is essentially strong unforgeability (SUF-CMA) formulated as a distinguishing game, with a challenge verification oracle and additional tagging and verification oracles that leak. Below is the formal definition.

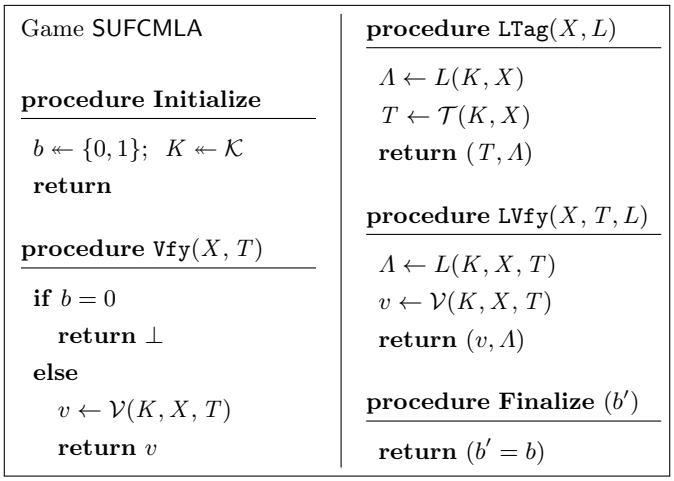

Fig. 5: Game used to define SUF-CMLA security.

Definition 4 (SUF-CMLA Security). *Let* $\textsc{Mac} = (\mathcal{T}, \mathcal{V})$ *be a message authentication code and the* SUFCMLA *game be as defined in Fig. 5. For any adversary* $\mathcal{A}$ *that never forwards queries from* `LTag` *to* `Vfy`*, and only queries leakage functions to its tagging and verification oracles in the respective sets* $\mathcal{L}_T$ *and* $\mathcal{L}_V$*, its corresponding* SUF-CMLA *advantage is given by:*

$$\mathbf{Adv}^{\text{suf-cmla}}_{\textsc{Mac}}(\mathcal{A}, \mathcal{L}_T, \mathcal{L}_V) = 2\Pr\left[\mathsf{SUFCMLA}^{\mathcal{A}} \Rightarrow \mathsf{true}\right] - 1\,.$$

Authenticated Encryption with Leakage. For an authenticated encryption scheme with associated data our target will be LAE security, which is a natural extension of the classical security notion put forth by Rogaway [24] to the leakage setting. This is defined formally below.

Definition 5 (LAE Security). *Let* $\textsc{Aead} = (\mathcal{E}, \mathcal{D})$ *be an authenticated encryption scheme with associated data and the* LAE *game be as defined in Fig. 6. Then for any adversary* $\mathcal{A}$ *that never forwards queries to or from the* `Enc` *and* `Dec` *oracles and only makes encryption and decryption queries containing*

Game LAE

procedure Initialize

$b \twoheadleftarrow \{0,1\}$; $K \twoheadleftarrow \mathcal{K}$
return

procedure $\texttt{Enc}(N, A, M)$

$C \leftarrow \mathcal{E}(K, N, A, M)$
if $b = 0$
  if $f[N, A, M] = \perp$
    $f[N, A, M] \twoheadleftarrow \{0,1\}^{|C|}$
    return $f[N, A, M]$
else
  return C

procedure Finalize (b')

return $(b' = b)$

procedure $\texttt{Dec}(N, A, C)$

$M \leftarrow \mathcal{D}(K, N, A, C)$
if $b = 0$
  return $\perp$
else
  return M

procedure $\texttt{LEnc}(N, A, M, L)$

$\Lambda \leftarrow L(K, N, A, M)$
$C \leftarrow \mathcal{E}(K, N, A, M)$
return (C, Λ)

procedure $\texttt{LDec}(N, A, C, L)$

$\Lambda \leftarrow L(K, N, A, C)$
$M \leftarrow \mathcal{D}(K, N, A, C)$
return (M, Λ)

Fig. 6: Game used to define LAE security.

leakage functions in the respective sets $\mathcal{L}_{AE}$ and $\mathcal{L}_{VD}$, its corresponding LAE *advantage is given by:*

$$\mathbf{Adv}^{\text{lae}}_{\text{AEAD}}(\mathcal{A}, \mathcal{L}_{AE}, \mathcal{L}_{VD}) = 2\Pr\left[\mathsf{LAE}^{\mathcal{A}} \Rightarrow \mathsf{true}\right] - 1\,.$$

Generic Composition in the Leakage Setting. The N2 construction was introduced in [22] and is depicted pictorially in Fig. 7. In [4] Barwell et al. prove a composition theorem for this construction that holds in the leakage setting. We will make use of this theorem and for completeness we reproduce it below, adapted to the random transformation model.

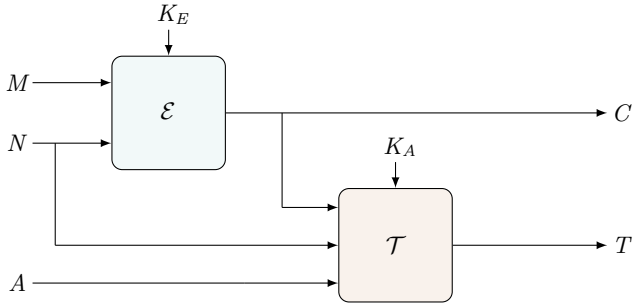

Fig. 7: Graphical representation of the N2 construction.

Theorem 1 (LAE **Security of the** N2 **Construction** [4])**.** *Let* $\text{SE} = (\mathcal{E}, \mathcal{D})$ *be a symmetric encryption scheme with associated leakage sets* $(\mathcal{L}_E, \mathcal{L}_D)$ *and* $\text{MAC} = (\mathcal{T}, \mathcal{V})$ *be a MAC with associated leakage sets* $(\mathcal{L}_T, \mathcal{L}_V)$*. Further let* N2 *be the composition of* SE *and* MAC *described in Fig. 7, with associated leakage sets* $(\mathcal{L}_{AE}, \mathcal{L}_{VD})$ *where* $\mathcal{L}_{AE} = \mathcal{L}_E \times \mathcal{L}_T$ *and* $\mathcal{L}_{VD} = \mathcal{L}_D \times \mathcal{L}_V$*. Then for any* LAE *adversary* $\mathcal{A}_{ae}$ *against* N2 *there exist adversaries* $\mathcal{A}_{se}$*,* $\mathcal{A}_{prf}$*, and* $\mathcal{A}_{mac}$ *such that:*

$$\mathbf{Adv}^{\text{lae}}_{\text{N2}}(\mathcal{A}_{ae}, \mathcal{L}_{AE}, \mathcal{L}_{VD}) \leq \mathbf{Adv}^{\text{ind-acpla}}_{\text{SE}}(\mathcal{A}_{se}, \mathcal{L}_E, \mathcal{L}_D) + \mathbf{Adv}^{\text{lprf}}_{\mathcal{T}}(\mathcal{A}_{prf}, \mathcal{L}_T) + 2\mathbf{Adv}^{\text{suf-cmla}}_{\text{MAC}}(\mathcal{A}_{mac}, \mathcal{L}_T, \mathcal{L}_V)\,.$$

The resources of the above adversaries are quantified as follows. Let q and μ be such that $\mathcal{A}_{ae}$ makes at most q queries to any of its oracles `Enc`, `Dec`, `LEnc`, and `LDec`, up to a total of μ bits for each oracle. Further let q_ρ denote the number of queries that $\mathcal{A}_{ae}$ makes to ρ. Then $\mathcal{A}_{se}$ queries each of `Enc`, `LEnc`, and `LDec` at most q times up to a total of μ bits, and queries ρ at most $Q_{\mathcal{T}}(2q, 2\mu) + Q_{\mathcal{V}}(q, \mu) + q_\rho$ times. As for $\mathcal{A}_{lprf}$, it makes at most q and $2q$ queries to `F` and `LF`, totalling μ and 2μ bits respectively, and $Q_{\mathcal{E}}(q, \mu) + Q_{\mathcal{D}}(q, \mu) + q_\rho$ queries to ρ. Finally, $\mathcal{A}_{mac}$ queries each its oracles `Vfy`, `LTag`, and `LVfy` at most $2q$ times, totalling 2μ bits in each case, and queries its ρ oracle $Q_{\mathcal{E}}(2q, 2\mu) + Q_{\mathcal{D}}(q, \mu) + q_\rho$ times.

3 SLAE: A Sponge-Based LAE Construction

SLAE, pronounced "sleigh", is a **S**ponge-based non-adaptive **L**eakage-resilient **AE**AD scheme. It is based on, and is closely related to, a prior sponge-based AEAD scheme called ISAP [12]. ISAP is a nonce-based AEAD scheme intended to inherently resist side-channel attacks while simultaneously fitting the well-established syntax of AEAD schemes. More specifically, it claims security against Differential Power Analysis (DPA) by employing a rekeying mechanism. An important challenge that ISAP overcomes, is to avoid decrypting distinct ciphertexts under the same key without maintaining a state. Furthermore, as noted by ISAP's designers, the sponge construction seems markedly well-suited to protect against side-channels. Typically, the sponge employs a large state that is continually evolving, which intuitively endows it with an intrinsic resilience to information leakage. Thus, in contrast to other designs, ISAP potentially offers a fairly efficient LAE solution that can be instantiated with off-the-shelf primitives. However, as we already noted, ISAP's biggest limitation is that its design is not backed by any formal security analysis, not even in the absence of leakage.

ISAP is composed of a symmetric encryption scheme ISAPENC and a MAC ISAPMAC combined according to the N2 construction. These components were conceived by augmenting established sponge constructs with a rekeying function. In particular the design rationale behind ISAPMAC is to augment a sponge-based suffix MAC with a rekeying function. The rekeying is such that the key fed into the suffix MAC itself depends on the inputs being authenticated and a master authentication key. Similarly ISAPENC is a standard sponge-based encryption scheme whose key is derived from a master encryption key and the nonce. Throughout, the rekeying function is realised from the sponge by setting the absorption rate to be one. Intuitively, ISAP's resistance to DPA comes from the fact that encryption and authentication never use the same key more than once, and the slow absorption rate employed in the rekeying function. Both of these factors limit the so-called data complexity of computations involving secret values, which in turn encumbers DPA attacks. See [12] for more details on ISAP.

SLAE retains the main structure of ISAP, as well as its benefits, but it includes some changes and restrictions that facilitate its security analysis. While the majority of these differences are conceptual, they are substantial enough, however, to *invalidate* any claim that our security proof applies to ISAP. The design of SLAE can be understood across *three* different levels of abstraction. At the highest level, like ISAP, it is the N2 composition of a symmetric encryption scheme SLENC and a MAC SLMAC. At the second abstraction level, SLMAC and SLENC can be viewed in terms of smaller components. Specifically, we view SLMAC as combining a collision-resistant vector hash function $\mathcal{H}$ and a fixed-input-length function $\mathcal{F}'$, and we decompose SLENC into a fixed-input-length function $\mathcal{F}$ and a pseudorandom generator with variable output length $\mathcal{G}$. Indeed this view corresponds to our generic construction of a non-adaptively leakage-resilient AEAD scheme which we refer to as the FGHF$'$ construction.

Note that there is no explicit idea of rekeying in the FGHF$'$ construction. The only leakage-resilient primitives are $\mathcal{F}$ and $\mathcal{F}'$. For security we will require both to be LPRF secure and $\mathcal{F}'$ to additionally be LUF secure. Thus LAE schemes are easy to construct once we have such primitives. Moreover, $\mathcal{F}$ is invoked once for encryption, and likewise $\mathcal{F}'$ is invoked once for authentication, irrespective of the message length. SLAE is obtained by instantiating the four components in the FGHF$'$ construction with T-sponges. This is the third level view. While the design rationale behind the FGHF$'$ construction is quite distinct from that of ISAP, once instantiated, SLAE and ISAP suddenly look very similar.

We now describe SLAE in more detail and then elaborate on the differences between SLAE and ISAP in Section 3.4.

3.1 High-Level View of SLAE

As already noted, SLAE = (SLAE-$\mathcal{E}$, SLAE-$\mathcal{D}$) is a nonce-based AEAD scheme composed from a nonce-based symmetric encryption scheme SLENC = (SLENC-$\mathcal{E}$, SLENC-$\mathcal{D}$) and a MAC SLMAC = (SLMAC-$\mathcal{T}$, SLMAC-$\mathcal{V}$). These are combined according to the N2 composition, where the key is split into an encryption

key K_E and an authentication key K_A. During encryption, SLENC-$\mathcal{E}$ takes the nonce, message, and key K_E to return a ciphertext which is then fed together with the nonce, associated data, and key K_A, to SLMAC-$\mathcal{T}$ to produce a tag which is then appended onto the ciphertext. Decryption proceeds by reversing these operations in a *verify-then-decrypt* manner, whereby ciphertext decryption using SLENC-$\mathcal{D}$ proceeds *only if* tag verification under SLMAC-$\mathcal{V}$ was successful. The pseudocode for this composition is described in Fig. 8.

SLAE-$\mathcal{E}(K, N, A, M)$	SLAE-$\mathcal{D}(K, N, A, \overline{C})$
parse K **as** $K_E \parallel K_A$	**parse** K **as** $K_E \parallel K_A$
$C \leftarrow$ SLENC-$\mathcal{E}(K_E, N, M)$	**parse** $\overline{C}$ **as** $C \parallel T$
$T \leftarrow$ SLMAC-$\mathcal{T}(K_A, (N, A, C))$	$v \leftarrow$ SLMAC-$\mathcal{V}(K_A, (N, A, C), T)$
$\overline{C} \leftarrow C \parallel T$	**if** $v = \top$
return $\overline{C}$	$M \leftarrow$ SLENC-$\mathcal{D}(K_E, N, C)$
	return M
	else
	return $\perp$

Fig. 8: High-level description of SLAE in terms of SLMAC and SLENC.

3.2 The SLMAC Construction

A pseudocode description of SLMAC = (SLMAC-$\mathcal{T}$, SLMAC-$\mathcal{V}$) can be found in Fig. 9. It is a vector MAC operating on the triple (N, A, C), where verification works by recomputing the tag for the given triple and checking that it is identical to the given tag. As such, the core functionality of SLMAC is captured in the tagging algorithm SLMAC-$\mathcal{T}$, which is additionally depicted in Fig. 10. The tagging algorithm can be understood as being composed of a (sponge-based) vector hash function compressing the triple (N, A, C) into a digest of size w bits, which is then fed to the unpredictable function SLFUNC to produce a tag of size t bits. The nonce N is required to be m bits long, whereas A and C can be of arbitrary length. Accordingly, SLMAC-$\mathcal{T}$ starts by padding both A and C so that their lengths are integer multiples of the sponge rate r. Note that the padding function, lpad, always returns at least a single bit of padding and is always applied, even if the input string is already an integer multiple of r.

To compute the hash digest H, the internal state is initialised to $\rho(N \parallel IV)$, where IV is a constant string of size $n - m$, and the padded associated data $\boldsymbol{A}$ and the padded ciphertext $\boldsymbol{C}$ are then absorbed block by block. An input separation mechanism is employed in order to demarcate the boundary between $\boldsymbol{A}$ and $\boldsymbol{C}$. This involves XORing the string $1\|0^{c-1}$ to the inner part of the state once $\boldsymbol{A}$ has been absorbed, and ensures that distinct pairs $(\boldsymbol{A}, \boldsymbol{C}) \neq (\overline{\boldsymbol{A}}, \overline{\boldsymbol{C}})$ for which $\boldsymbol{A} \,\|\, \boldsymbol{C} = \overline{\boldsymbol{A}} \,\|\, \overline{\boldsymbol{C}}$ do not result in the same hash digest.

Once the hash digest is evaluated, it is fed into SLFUNC to compute the final tag. This is also a sponge-based construction for which a graphical representation appears in Fig. 11. Here the state is initialised to $\rho(K_A \parallel IV)$ and the hash digest is then absorbed at a *reduced rate* of rr bits. Once the complete digest has been absorbed the left most t bits of the state are output as the tag.

3.3 The SLENC Construction

This is the sponge-based symmetric encryption scheme SLENC = (SLENC-$\mathcal{E}$, SLENC-$\mathcal{D}$) described in Fig. 12 and depicted in Fig. 13. It is easy to see that SLENC-$\mathcal{D}(K_E, N, \cdot)$ = SLENC-$\mathcal{E}(K_E, N, \cdot)$, and consequently we only describe the operation of SLENC-$\mathcal{E}$. This algorithm can be viewed as being composed of a pseudorandom function SLFUNC, taking as input the pair (K_E, N), and whose output is then fed into a pseudorandom generator SPRG. The output of SPRG is then used to encrypt the message.

The nonce N is required to be m bits long and we do not require any additional padding for the message. The evaluation of SLFUNC proceeds by initialising the internal state to $\rho(K_E \parallel IV)$, with a constant IV of size $n - k$, and then absorbing the nonce at a reduced rate of rr bits. Once the nonce is absorbed, the output state S_{l+1} serves as the seed to the pseudorandom generator SPRG. A separate

SLMAC-$\mathcal{T}(K_A, (N, A, C))$

$\boldsymbol{A} \leftarrow A \parallel \mathsf{lpad}(A, r)$
parse $\boldsymbol{A}$ **as** $\boldsymbol{A}_1 \parallel \ldots \parallel \boldsymbol{A}_u$
  st $\forall i\ |\boldsymbol{A}_i| = r$
$\boldsymbol{C} \leftarrow C \parallel \mathsf{lpad}(C, r)$
parse $\boldsymbol{C}$ **as** $\boldsymbol{C}_1 \parallel \ldots \parallel \boldsymbol{C}_v$
  st $\forall i\ |\boldsymbol{C}_i| = r$
$Y_0 \leftarrow N \parallel IV$
$S_1 \leftarrow \rho(Y_0)$
// Absorb Associated Data
for i **in** $\{1, \ldots, u\}$
  $Y_i \leftarrow (\bar{S}_i \oplus \boldsymbol{A}_i) \parallel \hat{S}_i$
  $S_{i+1} \leftarrow \rho(Y_i)$
// Separate Inputs
$S_{u+1} \leftarrow \bar{S}_{u+1} \parallel \left(\hat{S}_{u+1} \oplus\ (1 \parallel 0^{c-1})\right)$
// Absorb Ciphertext
for i **in** $\{u+1, \ldots, u+v\}$
  $Y_i \leftarrow (\bar{S}_i \oplus \boldsymbol{C}_{i-u}) \parallel \hat{S}_i$
  $S_{i+1} \leftarrow \rho(Y_i)$
// Generate Tag
$H \leftarrow \lfloor S_{u+v+1} \rfloor_w$
$T \leftarrow \lfloor \text{SLFUNC}(K_A, H) \rfloor_t$
return T

SLMAC-$\mathcal{V}(K_A, (N, A, C), T)$

$T' \leftarrow$ SLMAC-$\mathcal{T}(K_A, (N, A, C))$
if $T = T'$
  return $\top$
return $\bot$

SLFUNC(K_A, H)

parse H **as** $H_1 \parallel \ldots \parallel H_l$
  st $\forall i\ |H_i| = rr$
$Y_0 \leftarrow K_A \parallel IV$
$S_1 \leftarrow \rho(Y_0)$
for i **in** $\{1, \ldots, l\}$
  $Y_i \leftarrow (\bar{S}_i \oplus H_i) \parallel \hat{S}_i$
  $S_{i+1} \leftarrow \rho(Y_i)$
return $\lfloor S_{l+1} \rfloor_t$

$\mathsf{lpad}(A, r)$

$x \leftarrow |A| \bmod r$
return $1 \parallel 0^{r-x-1}$

Fig. 9: Pseudocode description of SLMAC and SLFUNC.

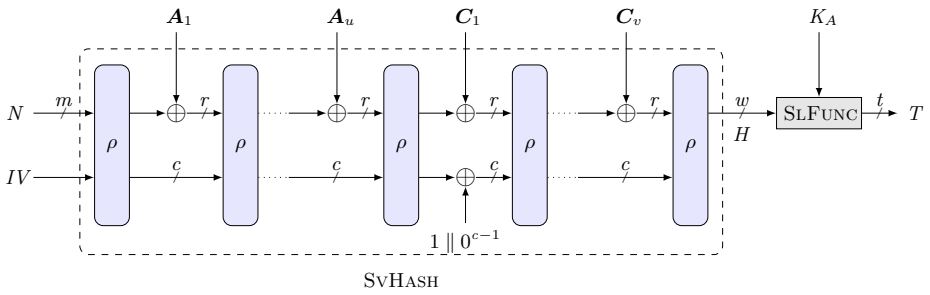

Fig. 10: Graphical illustration of SLMAC-$\mathcal{T}$.

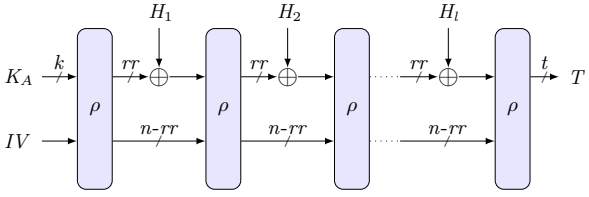

Fig. 11: Graphical illustration of SLFUNC.

pseudocode description of SPRG can be found in Fig. 14. The first ciphertext block is generated by XORing the outer part of this state with the first message block. Afterwards the initial state is given as input to the random transformation outputting a new state which is then used to derive the next ciphertext block by simply XORing again the outer state with the next message block. This process is repeated until the whole message has been processed. If the last message block is smaller than r bits, we simply truncate the outer state to the required size and XOR both parts to obtain the last ciphertext block.

SLENC-$\mathcal{E}(K_E, N, M)$	SLENC-$\mathcal{D}(K_E, N, C)$
parse N **as** $N_1 \parallel \ldots \parallel N_l$	**parse** N **as** $N_1 \parallel \ldots \parallel N_l$
st $\forall i\ \lvert N_i\rvert = rr$	**st** $\forall i\ \lvert N_i\rvert = rr$
parse M **as** $M_1 \parallel \ldots \parallel M_v$	**parse** C **as** $C_1 \parallel \ldots \parallel C_v$
st $\forall i < v\ \lvert M_i\rvert = r$ **and** $\lvert M_v\rvert \leq r$	**st** $\forall i < v\ \lvert C_i\rvert = r$ **and** $\lvert C_v\rvert \leq r$
// First Sponge Iteration	// First Sponge Iteration
$Y_0 \leftarrow K_E \parallel IV$	$Y_0 \leftarrow K_E \parallel IV$
$S_1 \leftarrow \rho(Y_0)$	$S_1 \leftarrow \rho(Y_0)$
// Absorb Nonce	// Absorb Nonce
for i **in** $\{1, \ldots, l\}$	**for** i **in** $\{1, \ldots, l\}$
$Y_i \leftarrow (\bar{S}_i \oplus N_i) \parallel \hat{S}_i$	$Y_i \leftarrow (\bar{S}_i \oplus N_i) \parallel \hat{S}_i$
$S_{i+1} \leftarrow \rho(Y_i)$	$S_{i+1} \leftarrow \rho(Y_i)$
// Squeeze and Encrypt	// Squeeze and Decrypt
for i **in** $\{l+1, \ldots, l+v-1\}$	**for** i **in** $\{l+1, \ldots, l+v-1\}$
$C_{i-l} \leftarrow \bar{S}_i \oplus M_{i-l}$	$M_{i-l} \leftarrow \bar{S}_i \oplus C_{i-l}$
$S_{i+1} \leftarrow \rho(S_i)$	$S_{i+1} \leftarrow \rho(S_i)$
$C_v \leftarrow \lfloor \bar{S}_{l+v} \rfloor_{\lvert M_v\rvert} \oplus M_v$	$M_v \leftarrow \lfloor \bar{S}_{l+v} \rfloor_{\lvert C_v\rvert} \oplus C_v$
return $C_1 \parallel \ldots \parallel C_v$	**return** $M_1 \parallel \ldots \parallel M_v$

Fig. 12: Pseudocode description of the SLENC encryption scheme.

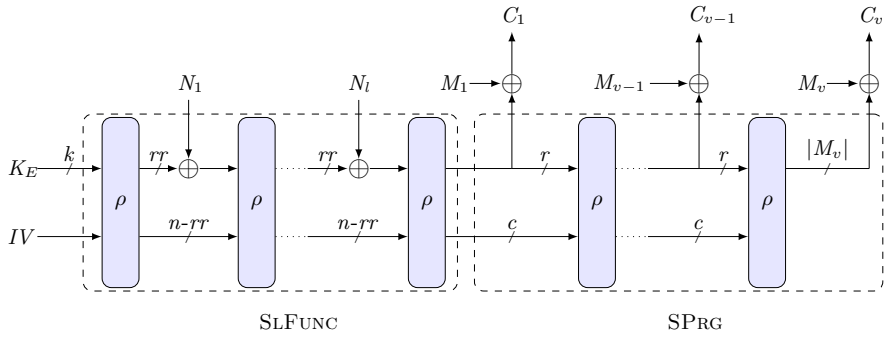

Fig. 13: Graphical illustration of SLENC-$\mathcal{E}$.

3.4 Differences Between SLAE and ISAP

We have already described in passing some of the differences between SLAE and ISAP, but for clarity, we summarise these distinctions below and discuss them in more detail.

The most prominent difference is that SLAE is based on the T-sponge whereas ISAP employs the P-sponge. In particular the security proofs of SLAE rely on treating ρ as a non-invertible transformation.

SPRG($seed, L$)

$v \leftarrow \lceil \frac{L}{r} \rceil$
$S_1 \leftarrow seed$
for i **in** $\{1, \ldots, v-1\}$
  $S_{i+1} \leftarrow \rho(S_i)$
$R \leftarrow \bar{S}_1 \parallel \ldots \parallel \bar{S}_v$
return $\lfloor R \rfloor_L$

Fig. 14: Pseudocode description of SPRG.

Treating ρ as an invertible random permutation would add another layer of complexity to the security analysis and we chose not to pursue this route at this point.

The description of ISAP actually specifies three distinct permutations, each obtained from the same round function but with a varying number of rounds. These are used in the different components of ISAP as a means of optimisation. Such heuristic optimisations could be employed in SLAE as well, but in our security analysis we instantiate SLAE with the same random transformation throughout. Indeed this is the more conservative assumption, since otherwise we would be treating these variants as being sampled independently at random when in fact they are intimately related.

Another difference between SLAE and ISAP can be seen in their MAC components SLMAC and ISAPMAC. The design of ISAPMAC is based on combining a rekeying function ISAPRK with a sponge-based suffix MAC. In turn, ISAPRK takes as input a hash of the MAC inputs. As a design optimisation, it is then noted that this hash is already being computed as part of the suffix MAC, at which point it is extracted, fed into ISAPRK, and its output (the session key) is fed back into the last permutation of the suffix MAC to yield the MAC tag. In contrast, in SLMAC, the value corresponding to the session key in ISAPMAC is output directly as the MAC tag.

Finally there are some differences in the way we set parameters in SLAE as opposed to ISAP. For instance, ISAP sets the size of the key and the nonce to be equal. On the other hand, our analysis indicates that the limiting factor in the security of SLAE is the key size. As such it makes sense to set the key size k equal to the width of the sponge n while setting the nonce to be much smaller, say between 64 and 128 bits.

4 The Security of FGHF′

In this section we establish the security of the FGHF′ construction which is depicted in Fig. 15. This is an abstraction of SLAE, and proving its security brings us halfway towards proving the security of SLAE. At the same time, we believe the FGHF′ construction to be of independent interest as it serves as a generic blueprint for constructing efficient AEAD schemes that are non-adaptively leakage-resilient.

The FGHF′ construction is a refinement of the N2 construction [22] which builds a nonce-based AEAD scheme from a nonce-based symmetric encryption scheme and a vector MAC. Barwell et al. [4] showed that the security of this construction extends to the setting of leakage resilience. Specifically they showed that if the encryption component is IND-aCPLA secure and the vector MAC is both LPRF and SUF-CMLA secure, then the composition is LAE secure. In turn the FGHF′ construction further breaks down the encryption component, denoted by SE$[\mathcal{F}, \mathcal{G}]$, and the vector MAC component, denoted by MAC$[\mathcal{H}, \mathcal{F}']$, of N2 into smaller parts. Namely encryption is realised from a fixed-input-length leakage-resilient PRF $\mathcal{F}$ and a standard PRG $\mathcal{G}$, whereas the vector MAC is built from a vector hash function $\mathcal{H}$, and a fixed-input-length function $\mathcal{F}'$ that is both leakage-resilient pseudorandom and leakage-resilient unpredictable.

Since FGHF′ is an instance of N2 we can apply the composition theorem of Barwell et al. [4], which we reproduced and adapted to the random transformation model in Section 2.5. Moreover, since we can view non-adaptive leakage as a special case of adaptive leakage where the leakage set is a singleton, the theorem carries over to that setting which is what we are interested in here. Thus to prove that the FGHF′ construction is LAE secure we only need to show that the encryption and MAC components meet the requirements of Theorem 1.

As it turns out, we can realise an IND-aCPLA secure encryption directly from an LPRF and a variable-output-length PRG. Here the PRG serves only to extend the range of the LPRF in order for

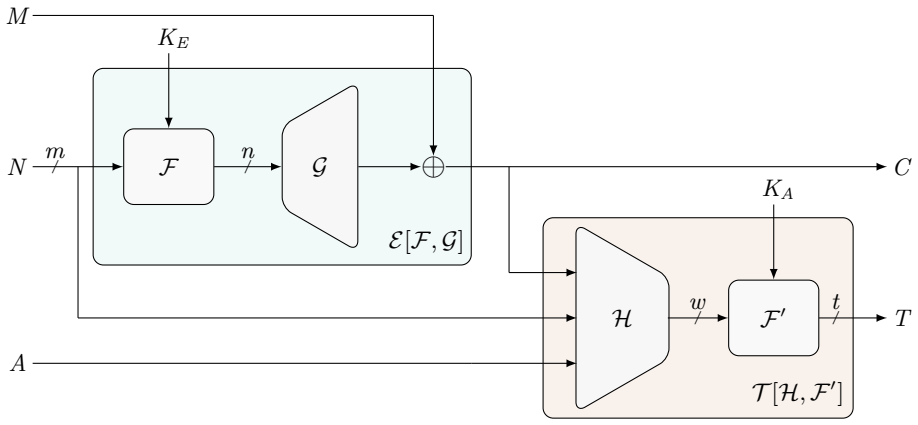

Fig. 15: Graphical representation of the FGHF′ construction. It corresponds to the N2 composition of $\textsc{Se}[\mathcal{F},\mathcal{G}] = (\mathcal{E},\mathcal{D})$ and $\textsc{Mac}[\mathcal{H},\mathcal{F}'] = (\mathcal{T},\mathcal{V})$ which are in turn composed of a fixed-input-length LPRF $\mathcal{F}$, a PRG $\mathcal{G}$, a vector hash $\mathcal{H}$, and a fixed-input-length function $\mathcal{F}'$ that is both a LUF and an LPRF.

the encryption scheme to accommodate variable-length messages. Surprisingly, a standard PRG without any leakage resilience suffices. As for the vector MAC component it needs to be an LPRF over a vector of strings and simultaneously satisfy SUF-CMLA security. Contrary to the leakage-free setting, the latter property is not automatically implied by the former when a MAC is constructed from an LPRF through the canonical construction. This is because the SUFCMLA game is more permissive than the LPRF game with respect to the adversary's queries. Namely, the adversary can forward queries from the `LVfy` to `Vfy`, whereas in the LPRF game the adversary is not allowed to forward queries from `LF` to `F`. This precludes reducing SUF-CMLA security to LPRF security due to our inability of simulating the verification oracles via the respective LPRF oracles. Note that SUF-CMLA needs to be defined this way for Theorem 1 to hold whereas lifting the restriction in the LPRF game would make it unsatisfiable. We overcome this problem by noting that, in the non-adaptive leakage setting, unpredictability suffices to achieve SUF-CMLA security, and at the same time we can allow the adversary to forward queries between its leakage and challenge oracles while maintaining satisfiability. This leads to our notion of a LUF which we prove to be sufficient to yield SUF-CMLA security. As we will see in the next section we can construct fixed-input-length function families satisfying both notions rather easily from sponges. Given such a function family $\mathcal{F}'$, we can turn it into the required vector MAC by composing it with a collision-resistant vector hash function. Specifically we show that we can extend the domain of LPRFs and LUFs, rather efficiently, by composing them with standard collision-resistant hash functions over appropriate domains.

Combining the results in this section, leads to the LAE security of the FGHF′ construction against non-adaptive leakage. We like this construction as it strikes a practical balance between security and efficiency. By settling for non-adaptive leakage, which seems to suffice for many practical applications, it only requires one call to each of the leakage-resilient primitives, $\mathcal{F}$ and $\mathcal{F}'$, per encryption query. In this work we focused on Slae which is a specific sponge-based instantiation of FGHF′, but other instantiations, possibly based on different techniques, are of course possible. Thus this construction essentially reduces the problem of designing non-adaptively leakage-resilient AEAD schemes to that of designing function families over small domains that are good LPRFs and LUFs, which conceptually is a much simpler target.

4.1 $\textsc{Se}[\mathcal{F},\mathcal{G}]$ is IND-aCPLA Secure

We begin by proving the security of the encryption component of FGHF′. Note that for this part security holds in more general setting of adaptive leakage. Below is the formal theorem statement and its proof is presented in Appendix B.1.

Theorem 2. *Let* $\textsc{Se}[\mathcal{F},\mathcal{G}]$ *be the encryption scheme depicted in Fig. 15, composed of the function family* $\mathcal{F}\colon \mathcal{K} \times \{0,1\}^m \to \{0,1\}^n$ *and the PRG* $\mathcal{G}\colon \{0,1\}^n \to \{0,1\}^*$ *with respective associated leakage sets* $\mathcal{L}_F$ *and* $\mathcal{L}_G$*. Then for any* IND-aCPLA *adversary* $\mathcal{A}_{se}$ *against* $\textsc{Se}[\mathcal{F},\mathcal{G}]$ *and associated leakage sets* $\mathcal{L}_E = \mathcal{L}_D = \mathcal{L}_F \times \mathcal{L}_G$*, there exist an* LPRF *adversary* $\mathcal{A}_{lprf}$ *against* $\mathcal{F}$ *and a* PRG *adversary* $\mathcal{A}_{prg}$

against $\mathcal{G}$ such that:

$$\mathbf{Adv}^{\text{ind-acpla}}_{\text{SE}[\mathcal{F},\mathcal{G}]}(\mathcal{A}_{se}, \mathcal{L}_E, \mathcal{L}_D) \leq 2\,\mathbf{Adv}^{\text{lprf}}_{\mathcal{F}}(\mathcal{A}_{lprf}, \mathcal{L}_F) + 2\,\mathbf{Adv}^{\text{prg}}_{\mathcal{G}}(\mathcal{A}_{prg})\ .$$

Let q and μ be such that $\mathcal{A}_{se}$ makes at most q queries totalling μ bits to each of its oracles `Enc`, `LEnc`, and `LDec`, and let q_ρ denote the number of queries it makes to ρ. Then $\mathcal{A}_{lprf}$ makes at most q and $2q$ queries to its oracles `F` and `LF`, totalling qm and $2qm$, respectively, and at most $Q_{\mathcal{G}}(2q, 2\mu) + q_\rho$ to ρ. As for $\mathcal{A}_{prg}$, it makes at most q queries to its oracle `G` totalling μ bits and $Q_{\mathcal{F}}(2q, 2qm) + Q_{\mathcal{G}}(2q, 2\mu) + q_\rho$ queries to ρ.

4.2 MAC$[\mathcal{H}, \mathcal{F}']$ is SUF-CMLA Secure

Next we reduce the SUF-CMLA security of MAC$[\mathcal{H}, \mathcal{F}']$ to the LUF security of $\mathcal{F}'$ and the collision resistance of $\mathcal{H}$. Towards this end, we first show that any LUF $\hat{\mathcal{F}}$ over domain $\mathcal{X}$ yields a SUF-CMLA secure MAC with message space $\mathcal{X}$ via the canonical construction. Then we show that such a function $\hat{\mathcal{F}}$ can be constructed from a fixed-input-length LUF $\mathcal{F}'$ and a collision-resistant hash function with domain $\mathcal{X}$. The formal theorem statements now follow. Their proofs can be found in Appendices B.2 and B.3 respectively.

Theorem 3. *Let $\hat{\mathcal{F}}\colon \mathcal{K}\times\mathcal{X} \to \{0,1\}^t$ be a function family with associated leakage set $\mathcal{L}_{\hat{F}}$, and let* MAC$[\hat{\mathcal{F}}]$ *be the corresponding canonical MAC with associated leakage sets $\mathcal{L}_T$, $\mathcal{L}_V$ where $\mathcal{L}_{\hat{F}} = \mathcal{L}_T = \mathcal{L}_V$. Then for any* SUF-CMLA *adversary $\mathcal{A}_{mac}$ against* MAC$[\hat{\mathcal{F}}]$*, there exists an adversary $\mathcal{A}_{luf}$ against $\hat{\mathcal{F}}$ such that:*

$$\mathbf{Adv}^{\text{suf-cmla}}_{\text{MAC}[\hat{\mathcal{F}}]}(\mathcal{A}_{mac}, \mathcal{L}_T, \mathcal{L}_V) \leq \mathbf{Adv}^{\text{luf}}_{\hat{\mathcal{F}}}\left(\mathcal{A}_{luf}, \mathcal{L}_{\hat{F}}\right)\ .$$

Let q and μ be such that $\mathcal{A}_{mac}$ makes at most q queries totalling μ bits to each of its oracles `Vfy`, `LTag`, and `LVfy`. Then $\mathcal{A}_{luf}$ makes at most q, $2q$, and $2q$ queries to `F`, `Lkg`, and `Guess`, totalling μ, 2μ, and 2μ bits, respectively.

Theorem 4. *Let $\mathcal{F}'\colon \mathcal{K}\times\{0,1\}^w \to \{0,1\}^t$ be a function family with associated leakage set $\mathcal{L}_{F'}$, and let $\mathcal{H}\colon \mathcal{X} \to \{0,1\}^w$ be a hash function over any domain $\mathcal{X}$. Further let their composition $\hat{\mathcal{F}}$ be defined as*

$$\hat{\mathcal{F}}(K, X) = \mathcal{F}'(K, \mathcal{H}(X))$$

where $X \in \mathcal{X}$, $K \in \mathcal{K}$, and $\mathcal{L}_{\hat{F}} = \mathcal{L}_{F'} \times \mathcal{L}_H$ for any set of efficiently computable functions $\mathcal{L}_H$. Then for any LUF *adversary $\mathcal{A}_{luf}$ against $\hat{\mathcal{F}}$, there exists a corresponding* LUF *adversary $\mathcal{A}'_{luf}$ against $\mathcal{F}'$ and an adversary $\mathcal{A}_{hash}$ against $\mathcal{H}$ such that:*

$$\mathbf{Adv}^{\text{luf}}_{\hat{\mathcal{F}}}\left(\mathcal{A}_{luf}, \mathcal{L}_{\hat{F}}\right) \leq 2\,\mathbf{Adv}^{\text{cr}}_{\mathcal{H}}(\mathcal{A}_{hash}) + \mathbf{Adv}^{\text{luf}}_{\mathcal{F}'}\left(\mathcal{A}'_{luf}, \mathcal{L}_{F'}\right)\ .$$

Let q and μ be such that $\mathcal{A}_{luf}$ makes at most q queries totalling μ bits to each of its oracles `F`, `Lkg`, and `Guess`, and let q_ρ denote the number of queries it makes to ρ. Then $\mathcal{A}'_{luf}$ makes at most q queries totalling qw bits to each of its oracles `F`, `Lkg`, and `Guess`, and at most $Q_{\mathcal{H}}(3q, 3\mu) + q_\rho$ queries to ρ. As for $\mathcal{A}_{hash}$, it requires at most $Q_{\mathcal{F}'}(3q, 3qw) + Q_{\mathcal{H}}(3q, 3\mu)$ queries to ρ in order to simulate $\mathcal{F}'$ and $\mathcal{H}$.

Combining both theorems, we obtain the following simple corollary reducing the SUF-CMLA security of MAC$[\mathcal{H}, \mathcal{F}']$ to that of its building blocks $\mathcal{H}$ and $\mathcal{F}'$.

Corollary 1. *Let* MAC$[\mathcal{H}, \mathcal{F}']$ *be the MAC component depicted in Fig. 15, composed of the hash function $\mathcal{H}$ and the function family $\mathcal{F}'$ with respective leakage sets $\mathcal{L}_H$ and $\mathcal{L}_{F'}$. Then for any* SUF-CMLA *adversary $\mathcal{A}_{mac}$ against* MAC$[\mathcal{H}, \mathcal{F}']$ *with associated leakage sets $\mathcal{L}_T = \mathcal{L}_V = \mathcal{L}_{F'} \times \mathcal{L}_H$, there exists a* LUF *adversary $\mathcal{A}_{luf}$ against $\mathcal{F}'$ and an adversary $\mathcal{A}_{hash}$ against $\mathcal{H}$ such that:*

$$\mathbf{Adv}^{\text{suf-cmla}}_{\text{MAC}[\mathcal{H},\mathcal{F}']}(\mathcal{A}_{mac}, \mathcal{L}_T, \mathcal{L}_V) \leq 2\,\mathbf{Adv}^{\text{cr}}_{\mathcal{H}}(\mathcal{A}_{hash}) + \mathbf{Adv}^{\text{luf}}_{\mathcal{F}'}(\mathcal{A}_{luf}, \mathcal{L}_{F'})\ .$$

Suppose $\mathcal{A}_{mac}$ makes at most q queries totalling at most μ bits to each of its oracles `Vfy`, `LTag`, and `LVfy`, and q_ρ to ρ. Then $\mathcal{A}_{luf}$ makes at most $2q$ queries totalling at most $2qw$ bits to each of the oracles in the LUF game, and $Q_{\mathcal{H}}(6q, 6\mu) + q_\rho$ queries to ρ. As for $\mathcal{A}_{hash}$ it needs at most $Q_{\mathcal{F}'}(6q, 6qw) + Q_{\mathcal{H}}(6q, 6\mu)$ queries to ρ to simulate $\mathcal{F}'$ and $\mathcal{H}$.

4.3 MAC$[\mathcal{H}, \mathcal{F}']$ is LPRF Secure

The final piece needed to apply Theorem 1 is to show that MAC$[\mathcal{H}, \mathcal{F}']$, or rather its tagging algorithm $\mathcal{T}[\mathcal{H}, \mathcal{F}']$, is a leakage-resilient PRF. Since by assumption $\mathcal{F}'$ is already an LPRF, this result is analogous to Theorem 4 in that it provides us a with simple technique for extending the domain of an LPRF. The proof can be found in Appendix B.4.

Theorem 5. *Let $\mathcal{F}' \colon \mathcal{K} \times \{0,1\}^w \to \{0,1\}^t$ be a function family with associated leakage set $\mathcal{L}_{F'}$, and let $\mathcal{H} \colon \mathcal{X} \to \{0,1\}^w$ be a hash function over the domain $\mathcal{X}$. Further let their composition $\hat{\mathcal{F}}$ be defined as*

$$\hat{\mathcal{F}}(K, X) = \mathcal{F}'(K, \mathcal{H}(X))$$

where $X \in \mathcal{X}$, $K \in \mathcal{K}$, and $\mathcal{L}_{\hat{F}} = \mathcal{L}_{F'} \times \mathcal{L}_H$ for any set of efficiently computable functions $\mathcal{L}_H$. Then for any LPRF *adversary $\mathcal{A}_{lprf}$ against $\hat{\mathcal{F}}$, there exists a corresponding* LPRF *adversary $\mathcal{A}'_{lprf}$ against $\mathcal{F}'$ and an adversary $\mathcal{A}_{hash}$ against $\mathcal{H}$ such that:*

$$\mathbf{Adv}^{\text{lprf}}_{\hat{\mathcal{F}}}\left(\mathcal{A}_{lprf}, \mathcal{L}_{\hat{F}}\right) \leq 2\,\mathbf{Adv}^{\text{cr}}_{\mathcal{H}}\left(\mathcal{A}_{hash}\right) + \mathbf{Adv}^{\text{lprf}}_{\mathcal{F}'}\left(\mathcal{A}'_{lprf}, \mathcal{L}_{F'}\right) .$$

Let q and μ be such that $\mathcal{A}_{lprf}$ makes at most q queries totalling μ bits to each of its oracles `F` and `LF`, and let q_ρ denote the number of queries it makes to ρ. Then $\mathcal{A}'_{lprf}$ makes at most q queries totalling qw bits to each of its oracles `F` and `LF`, and at most $Q_{\mathcal{H}}(2q, 2\mu) + q_\rho$ queries to ρ. As for $\mathcal{A}_{hash}$, it requires at most $Q_{\mathcal{F}'}(2q, 2qw) + Q_{\mathcal{H}}(2q, 2\mu)$ queries to ρ in order to simulate $\mathcal{F}'$ and $\mathcal{H}$.

4.4 The FGHF′ Composition Theorem

Collecting the results from this section and combining it with the N2 composition theorem we get the following composition theorem for the FGHF′ construction.

Theorem 6 (LAE Security of the FGHF′ Construction). *Let $\mathcal{F}$ be a fixed-input-length* LPRF*, $\mathcal{G}$ a* PRG*, $\mathcal{H}$ a vector hash function, and $\mathcal{F}'$ be a fixed-input-length function that is both an* LUF *and an* LPRF *with associated leakage sets $\mathcal{L}_F$, $\mathcal{L}_G$, $\mathcal{L}_H$, and $\mathcal{L}_{F'}$, respectively. Let* FGHF′ *be the composition of $\mathcal{F}$, $\mathcal{G}$, $\mathcal{H}$, and $\mathcal{F}'$ with associated leakage sets $\mathcal{L}_{AE} = \mathcal{L}_{VD} = \mathcal{L}_F \times \mathcal{L}_G \times \mathcal{L}_H \times \mathcal{L}_{F'}$. Then for any* LAE *adversary $\mathcal{A}_{ae}$ against* FGHF′ *there exist adversaries $\mathcal{A}_{lprf}$, $\mathcal{A}'_{lprf}$, $\mathcal{A}_{prg}$, $\mathcal{A}_{hash}$, and $\mathcal{A}_{luf}$ such that:*

$$\begin{aligned}\mathbf{Adv}^{\text{lae}}_{\text{FGHF}'}\left(\mathcal{A}_{ae}, \mathcal{L}_{AE}, \mathcal{L}_{VD}\right) \leq\ & 2\,\mathbf{Adv}^{\text{lprf}}_{\mathcal{F}}\left(\mathcal{A}_{lprf}, \mathcal{L}_F\right) + 2\,\mathbf{Adv}^{\text{lprf}}_{\mathcal{F}'}\left(\mathcal{A}'_{lprf}, \mathcal{L}_{F'}\right) \\ & + 2\,\mathbf{Adv}^{\text{prg}}_{\mathcal{G}}\left(\mathcal{A}_{prg}\right) + 6\,\mathbf{Adv}^{\text{cr}}_{\mathcal{H}}\left(\mathcal{A}_{hash}\right) \\ & + 2\,\mathbf{Adv}^{\text{luf}}_{\mathcal{F}'}\left(\mathcal{A}_{luf}, \mathcal{L}_{F'}\right) .\end{aligned}$$

Now suppose $\mathcal{A}_{ae}$ makes at most q queries totalling at most μ bits to each of its `Enc`, `LEnc`, `Dec`, and `LDec` oracles, and let q_ρ denote its number of queries to ρ. Then, $\mathcal{A}_{lprf}$ makes at most $2q$ queries totalling $2qm$ bits to each of its oracles `F` and `LF`, and at most $2Q_{\mathcal{H}}(2q, 2\mu) + 2Q_{\mathcal{F}'}(2q, 2qw) + Q_{\mathcal{G}}(2q, 2\mu)$ queries to ρ. Similarly, $\mathcal{A}'_{lprf}$ makes at most $2q$ queries totalling $2qw$ bits to each of its oracles `F` and `LF`, and at most $2Q_{\mathcal{F}}(q, qm) + 2Q_{\mathcal{G}}(q, \mu) + Q_{\mathcal{H}}(4q, 4\mu)$ queries to ρ. $\mathcal{A}_{luf}$ makes at most $4q$ queries, totalling $4qw$ bits to each of its oracles `F` and `Lkg`, and at most $2Q_{\mathcal{F}}(2q, 2qm) + 2Q_{\mathcal{G}}(2q, 2\mu) + Q_{\mathcal{H}}(12q, 12\mu)$ to ρ. As for $\mathcal{A}_{prg}$, it makes at most q queries, totalling μ bits, to its oracle `G` and at most $2Q_{\mathcal{H}}(2q, 2\mu) + 2Q_{\mathcal{F}'}(2q, 2qw) + Q_{\mathcal{F}}(2q, 2qm) + Q_{\mathcal{G}}(2q, 2\mu)$ queries to ρ. Finally, $\mathcal{A}_{hash}$ requires at most $2Q_{\mathcal{F}}(2q, 2qm) + 2Q_{\mathcal{G}}(2q, 2\mu) + Q_{\mathcal{F}'}(12q, 12qw) + Q_{\mathcal{H}}(12q, 12\mu)$ queries to ρ.

5 Security of Sponge-Based Primitives

We now turn our attention to instantiating the constituent blocks of the FGHF′ construction using sponge-based primitives. Specifically we prove the security of the vector hash function SvHash, the pseudorandom generator SPrg, and the leakage-resilient function family SlFunc for instantiating both $\mathcal{F}$ and $\mathcal{F}'$. All primitives are based on the T-sponge and this particular instantiation of the FGHF′ construction gives rise to Slae. The most interesting results are Theorems 7 and 8 which substantiate our claim that sponges offer an inherent resistance to non-adaptive leakage. Informally these two theorems state that by using a reduced absorption rate (e.g. $rr = 1$), the impact of leakage on security can be limited to $\lambda 2^{rr}$. In particular, note that the aggregate security loss is independent of the number of

queries, meaning that the effect of leakage cannot be amplified by repeated use. While sponge-based hash functions and pseudorandom generators have been studied quite extensively, SvHash and SPrg are non-standard constructions. Firstly, they are based on a transformation rather than a permutation which is not common in the literature. Secondly, unlike other constructions SPrg treats the whole initial state as the seed, and SvHash takes a triple of strings as its input. Thus while not particularly novel, we include their security proofs for completeness.

5.1 Revised Security Bounds

The security bounds presented in the proceedings version of this paper [11] were overly optimistic, and in fact, incorrect. This was pointed out to us by Bart Mennink and Krzysztof Pietrzak. The error was due to a wrong enumeration of the support when applying Lemma 1 in the proofs.

5.2 A Sponge-Based Leakage-Resilient Function Family

Although LPRF and LUF security are incomparable notions, it is still possible to meet both notions simultaneously through a single primitive. Indeed the FGHF′ construction requires that such a primitive exist since $\mathcal{F}'$ is required to satisfy both security notions. We now show that the SlFunc construction is well-suited for this role, and in fact that it can be used to instantiate both the $\mathcal{F}$ and $\mathcal{F}'$ components – as is the case in Slae. Moreover, the most extensively studied leakage-resilient object is that of a pseudorandom function due to its versatility in several potential applications. SlFunc yields a practical construction of this primitive against non-adaptive leakage and as such we think it may be of independent interest. The security of SlFunc is stated formally in the following two theorems. Their proofs can be found in Sections C.1 and C.2 of the Appendix.

Theorem 7. *Let* SlFunc *be the function family described in Fig. 9 taking as input strings of size* $(l \cdot rr)$ *bits and returning* t*-bit strings. Then for any* LPRF *adversary* $\mathcal{A}$ *against* SlFunc *and any vector of leakage functions* $[L_0, \ldots, L_l]$ *where each component maps* n *bits to* λ *bits such that* $\mathcal{L}_\lambda = \{[L_0, \ldots, L_l]\}$*, it holds that:*

$$\mathbf{Adv}^{\mathrm{lprf}}_{\mathrm{SlFunc}}(\mathcal{A}, \mathcal{L}_\lambda) \leq \frac{q_T(q_T+2) + (q_{\mathtt{F}} + q_{\mathtt{LF}})q_\rho}{2^{n-rr-1}} + \frac{2q_\rho}{2^{k-\lambda 2^{rr}}} + \frac{2l\,q_{\mathtt{F}}\,q_\rho}{2^{n-\lambda 2^{rr}}}\,.$$

In the above q_ρ*,* $q_{\mathtt{F}}$*, and* $q_{\mathtt{LF}}$ *denote respectively the number of queries* $\mathcal{A}$ *makes to its oracles* ρ*,* $\mathtt{F}$*, and* $\mathtt{LF}$ *and* $q_T = (l+1)(q_{\mathtt{LF}} + q_{\mathtt{F}}) + q_\rho$*. Moreover it is required that* $q_\rho + l(q_{\mathtt{F}} + q_{\mathtt{LF}}) \leq 2^{k-1}$ *and* $(2^{rr})q_\rho + l(q_{\mathtt{F}} + q_{\mathtt{LF}}) \leq 2^{n-1}$*.*

The next Theorem shows that SlFunc is a good LUF. Its proof bears some similarity to that of Theorem 7 as it uses similar ideas. However one important difference lies in the leakage model that is used in this theorem. Since the $\mathtt{Lkg}$ oracle returns only the leakage and no output, we add here an extra leakage function that returns the leakage on the output of SlFunc. In the LPRF case this was not required since in that game the leakage oracle returns the full output anyway.

Theorem 8. *Let* SlFunc *be the function family described in Fig. 9 taking as input strings of size* $(l \cdot rr)$ *bits and returning* t*-bit long strings. Then for any* LUF *adversary* $\mathcal{A}$ *against* SlFunc*, and any vector of leakage functions* $[L_0, \ldots, L_{l+1}]$ *where each component maps* n *bits to* λ *bits and letting* $\mathcal{L}_\lambda = \{[L_0, \ldots, L_{l+1}]\}$*, it holds that:*

$$\mathbf{Adv}^{\mathrm{luf}}_{\mathrm{SlFunc}}(\mathcal{A}, \mathcal{L}_\lambda) \leq \frac{q_T(q_T+2)}{2^{n-rr}} + \frac{2q_\rho}{2^{k-\lambda 2^{rr}}} + \frac{2l\,q_{\mathtt{Lkg}}\,q_\rho}{2^{n-\lambda 2^{rr}}} + \frac{q_{\mathtt{Guess}}}{2^{t-\lambda-1}}\,.$$

In the above q_ρ*,* $q_{\mathtt{F}}$*,* $q_{\mathtt{Lkg}}$ *and* $q_{\mathtt{Guess}}$ *denote respectively the number of queries* $\mathcal{A}$ *makes to its oracles* ρ*,* $\mathtt{F}$*,* $\mathtt{Lkg}$*, and* $\mathtt{Guess}$ *and* $q_T = (l+1)(q_{\mathtt{F}} + q_{\mathtt{Lkg}} + q_{\mathtt{Guess}}) + q_\rho$*. Moreover it is required that the following conditions be satisfied* $q_\rho + (l+1)(q_{\mathtt{F}} + q_{\mathtt{Lkg}} + q_{\mathtt{Guess}}) \leq 2^{k-1}$*,* $(2^{rr})q_\rho + (l+1)(q_{\mathtt{F}} + q_{\mathtt{Lkg}} + q_{\mathtt{Guess}}) \leq 2^{n-1}$*, and* $q_{\mathtt{Guess}} + (l+1)(q_{\mathtt{F}} + q_{\mathtt{Lkg}} + q_{\mathtt{Guess}}) \leq 2^{n-1}$*.*

5.3 The Security of SPrg

As explained in Section 3.3, SlEnc can be decomposed into the cascade of SlFunc and SPrg, matching the encryption component of the FGHF′ construction. A pseudocode description of the variable-output-length pseudorandom generator SPrg is given in Fig. 14. Decomposing SlEnc this way requires us

to treat all of SPRG's initial state as the seed, which deviates from the more conventional ways of constructing sponge-based pseudorandom generators. Moreover we consider a security definition which allows the adversary to make multiple queries to the PRG, each with differing output lengths.

The security of SPRG is stated formally in Theorem 9. Its proof follows from a standard hybrid argument and can be found in Appendix C.3.

Theorem 9. *Let* SPRG *be the pseudorandom generator described in Fig. 14. Then for any PRG adversary $\mathcal{A}$, it holds that:*

$$\mathbf{Adv}^{\mathrm{prg}}_{\mathrm{SPRG}}(\mathcal{A}) \leq \frac{v\,q_{\mathtt{G}}\,q_{\rho}}{2^c - q_{\rho}} + \frac{v\,q_{\rho}\,q_{\mathtt{G}} + v^2\,q_{\mathtt{G}}^2}{2^n} + \frac{q_{\mathtt{G}}^2}{2^n}\,.$$

In the above, $\mathcal{A}$ makes at most q_{ρ} queries to the random transformation ρ and $q_{\mathtt{G}}$ queries to the challenge oracle $\mathtt{G}$*. Moreover, $v = \lceil \frac{L_{max}}{r} \rceil$, where L_{max} is an upper bound on the adversary's inputs to* $\mathtt{G}$*.*

5.4 A Sponge-Based Vector Hash Function

The final building block is the sponge-based vector hash function SvHash which is graphically represented in Fig. 10. It takes as input a triple of strings, namely a nonce, associated data and a ciphertext to return a string digest. A salient feature of this construction is the xoring of $1 \parallel 0^{c-1}$ into the inner state in order to separate the (padded) associated data from the (padded) ciphertext. We prove the security of SvHash in a modular fashion, by first reducing its security to that of a plain hash function taking a single input and then prove the collision-resistance of this latter construction in the random transformation model. The collision-resistance of SvHash is stated formally in the following theorem, and the full proof details can be found in Appendix C.4.

Theorem 10. *Let* SvHash *be the vector hash function described in Fig. 10. Then for any adversary $\mathcal{A}$ making q queries to ρ, it holds that:*

$$\mathbf{Adv}^{\mathrm{cr}}_{\mathrm{SvHash}}(\mathcal{A}) \leq \frac{q(q-1)}{2^{w+1}} + \frac{q(q+2)}{2^{c-1}}\,.$$

5.5 Concrete Security of Slae

A bound for the security of Slae is obtained by directly combining the FGHF′ composition theorem (Theorem 6) with Theorems 7 –10. It then only remains to derive concrete bounds for the expressions $Q_{\mathcal{F}}, Q_{\mathcal{G}}, Q_{\mathcal{H}}, Q_{\mathcal{F}'}$ for the specific case of Slae. Assuming a nonce size of m bits and that the output of $\mathcal{H}$ is w bits long, the following expressions are easily derived from the algorithm definitions. Namely, we have that:

$$Q_{\mathcal{F}}(q, qm) = q\left\lceil \frac{m+1}{rr} \right\rceil \qquad Q_{\mathcal{F}'}(q, qw) = q\left\lceil \frac{w+1}{rr} \right\rceil$$

$$Q_{\mathcal{G}}(q, \mu) = \left\lceil \frac{\mu}{r} \right\rceil \qquad Q_{\mathcal{H}}(q, \mu) = \left\lceil \frac{\mu}{r} \right\rceil + 3q\,.$$

6 Concluding Remarks and Implementation Aspects

In this work we proposed the FGHF′ construction as a template for constructing non-adaptively leakage-resilient AEAD schemes from relatively simpler primitives – requiring only two calls to the leakage-resilient functions per encryption or decryption call. We then presented Slae as a sponge-based instantiation of this construction, offering good performance and simplicity. Our security analysis shows that if the absorption rate is set sufficiently low, the transformation sponge yields a leakage-resilient function with the desired properties. However some care is needed in interpreting these results. Like most treatments of leakage resilience we assume that the leakage per evaluation is limited and does not drain the entropy in the secret state. Thus it is implicitly assumed that an implementation is good enough to withstand basic side-channel attacks like Simple Power Analysis (SPA) attacks. The benefit of our leakage-resilience security proof is that resistance to basic attacks automatically translates to resistance against more sophisticated attacks like Differential Power Analysis (DPA).

In the FGHF′ construction and SLAE, authenticity is verified by recomputing the MAC tag and testing for equality between the recomputed tag and the one included in the ciphertext. While our leakage model accounts for the leakage that may take place during the tag recomputation, equality testing is assumed to be leak-free. Thus any implementation of SLAE (or any other realisation of the FGHF′ construction) needs to ensure that equality testing does not leak, or take additional measures, such as masking, to protect against leakage from this component.

Finally the security of SLAE relies on it being instantiated with a non-invertible transformation rather than a permutation. On the other hand, most practical schemes employ permutations, such as KECCAK-p and XOODOO-p. While in this work we did not specify any concrete transformation, a natural candidate is to use $\rho(x) = p(x) \oplus x$ for $p \in \{\text{KECCAK-}p, \text{XOODOO-}p\}$. Although this construction is known to be differentiable from a random transformation when given access to p, this should not preclude it from being a suitable candidate for instantiating constructions in the random transformation model. Indeed, KECCAK-p and XOODOO-p are also differentiable from a random permutation when given access to their underlying building blocks.

Acknowledgements

We are grateful to Bart Mennink and Krzysztof Pietrzak for pointing out an error in a previous version of this paper. We thank Daniel Baur and Christian Schuller for initial discussions during the early stages of this project, and our anonymous reviewers for their constructive comments. Degabriele was supported by the German Federal Ministry of Education and Research (BMBF) as well as by the Hessian State Ministry for Higher Education, Research and Arts (HMWK) within CRISP. Janson and Struck have been (partially) funded by the Deutsche Forschungsgemeinschaft (DFG) – SFB 1119 – 236615297.

References

1. M. Abdalla, S. Belaïd, and P.-A. Fouque. Leakage-resilient symmetric encryption via re-keying. In G. Bertoni and J.-S. Coron, editors, *CHES 2013*, volume 8086 of *LNCS*, pages 471–488. Springer, Heidelberg, Aug. 2013.
2. F. abed, F. Berti, and S. Lucks. Insecurity of RCB: Leakage-resilient authenticated encryption. Cryptology ePrint Archive, Report 2016/1121, 2016. http://eprint.iacr.org/2016/1121.
3. M. Agrawal, T. K. Bansal, D. Chang, A. K. Chauhan, S. Hong, J. Kang, and S. K. Sanadhya. Rcb: leakage-resilient authenticated encryption via re-keying. *The Journal of Supercomputing*, 74(9):4173–4198, Sep 2018.
4. G. Barwell, D. P. Martin, E. Oswald, and M. Stam. Authenticated encryption in the face of protocol and side channel leakage. In T. Takagi and T. Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 693–723. Springer, Heidelberg, Dec. 2017.
5. M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Heidelberg, May / June 2006.
6. D. J. Bernstein. CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness, 2014.
7. F. Berti, F. Koeune, O. Pereira, T. Peters, and F.-X. Standaert. Leakage-resilient and misuse-resistant authenticated encryption. Cryptology ePrint Archive, Report 2016/996, 2016. http://eprint.iacr.org/2016/996.
8. F. Berti, O. Pereira, T. Peters, and F.-X. Standaert. On leakage-resilient authenticated encryption with decryption leakages. *IACR Trans. Symm. Cryptol.*, 2017(3):271–293, 2017.
9. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Sponge functions. In *ECRYPT Hash Workshop*, 2007. https://keccak.team/files/SpongeFunctions.pdf.
10. D. Boneh and V. Shoup. *A Graduate Course in Applied Cryptography*. draft 0.4.
11. J. P. Degabriele, C. Janson, and P. Struck. Sponges resist leakage: The case of authenticated encryption. In S. D. Galbraith and S. Moriai, editors, *ASIACRYPT 2019, Part II*, volume 11922 of *LNCS*, pages 209–240. Springer, Heidelberg, Dec. 2019.
12. C. Dobraunig, M. Eichlseder, S. Mangard, F. Mendel, and T. Unterluggauer. ISAP – towards side-channel secure authenticated encryption. *IACR Trans. Symm. Cryptol.*, 2017(1):80–105, 2017.
13. C. Dobraunig and B. Mennink. Leakage resilience of the duplex construction. In S. D. Galbraith and S. Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 225–255. Springer, Heidelberg, Dec. 2019.
14. Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. Cryptology ePrint Archive, Report 2003/235, 2003. http://eprint.iacr.org/2003/235.
15. Y. Dodis and K. Pietrzak. Leakage-resilient pseudorandom functions and side-channel attacks on Feistel networks. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 21–40. Springer, Heidelberg, Aug. 2010.

16. S. Dziembowski and K. Pietrzak. Leakage-resilient cryptography. In *49th FOCS*, pages 293–302. IEEE Computer Society Press, Oct. 2008.
17. S. Faust, K. Pietrzak, and J. Schipper. Practical leakage-resilient symmetric cryptography. In E. Prouff and P. Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 213–232. Springer, Heidelberg, Sept. 2012.
18. O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, Oct. 1986.
19. C. Guo, O. Pereira, T. Peters, and F.-X. Standaert. Leakage-resilient authenticated encryption with misuse in the leveled leakage setting: Definitions, separation results, and constructions. Cryptology ePrint Archive, Report 2018/484, 2018. https://eprint.iacr.org/2018/484.
20. C. Guo, O. Pereira, T. Peters, and F.-X. Standaert. Towards lightweight side-channel security and the leakage-resilience of the duplex sponge. Cryptology ePrint Archive, Report 2019/193, 2019. https://eprint.iacr.org/2019/193.
21. J. Longo, D. P. Martin, E. Oswald, D. Page, M. Stam, and M. Tunstall. Simulatable leakage: Analysis, pitfalls, and new constructions. In P. Sarkar and T. Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 223–242. Springer, Heidelberg, Dec. 2014.
22. C. Namprempre, P. Rogaway, and T. Shrimpton. Reconsidering generic composition. In P. Q. Nguyen and E. Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 257–274. Springer, Heidelberg, May 2014.
23. O. Pereira, F.-X. Standaert, and S. Vivek. Leakage-resilient authentication and encryption from symmetric cryptographic primitives. In I. Ray, N. Li, and C. Kruegel, editors, *ACM CCS 2015*, pages 96–108. ACM Press, Oct. 2015.
24. P. Rogaway. Authenticated-encryption with associated-data. In V. Atluri, editor, *ACM CCS 2002*, pages 98–107. ACM Press, Nov. 2002.
25. F. Standaert, O. Pereira, Y. Yu, J. Quisquater, M. Yung, and E. Oswald. Leakage resilient cryptography in practice. In A. Sadeghi and D. Naccache, editors, *Towards Hardware-Intrinsic Security - Foundations and Practice*, Information Security and Cryptography, pages 99–134. Springer, 2010.
26. F.-X. Standaert, O. Pereira, and Y. Yu. Leakage-resilient symmetric cryptography under empirically verifiable assumptions. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 335–352. Springer, Heidelberg, Aug. 2013.
27. Y. Yu, F.-X. Standaert, O. Pereira, and M. Yung. Practical leakage-resilient pseudorandom generators. In E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, editors, *ACM CCS 2010*, pages 141–151. ACM Press, Oct. 2010.

A Additional Preliminary Material

A.1 Standard Cryptographic Primitives

We make use of the following definition of a pseudorandom generator. Note that our syntax defines a pseudorandom generator with variable output length, where the output length (in bits) is specified as part of the input. In addition, in our security definition we allow the adversary to make multiple queries to the challenge oracle $\mathtt{G}$.

Definition 6 (Pseudorandom Generators). *Let $\mathcal{G}\colon \mathcal{S} \times \mathbb{N} \to \{0,1\}^*$ be a pseudorandom generator with an associated seed space $\mathcal{S}$, and let the* PRG *game be as defined in Fig. 16. Then for any adversary $\mathcal{A}$, its corresponding* PRG *advantage is given by:*

$$\mathbf{Adv}^{\mathrm{prg}}_{\mathcal{G}}(\mathcal{A}) = 2\Pr\left[\mathsf{PRG}^{\mathcal{A}} \Rightarrow \mathsf{true}\right] - 1\,.$$

We define collision-resistant hash functions over a generic domain $\mathcal{X}$. Letting $\mathcal{X} = \{0,1\}^*$ results in the usual syntax but we can also, for instance, model a vector hash function over a triple of strings by setting $\mathcal{X} = \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^*$. For simplicity we only consider the random transformation model.

Definition 7 (Collision-Resistant Hash Functions). *Let $\mathcal{H}\colon \mathcal{X} \to \{0,1\}^w$ be a hash function constructed from a random transformation ρ, with domain $\mathcal{X}$ and output length w. Then for any adversary $\mathcal{A}$ with oracle access to ρ, its corresponding advantage is given by:*

$$\mathbf{Adv}^{\mathrm{cr}}_{\mathcal{H}}(\mathcal{A}) = 2\Pr[(X_0, X_1) \leftarrow \mathcal{A}^{\rho} \,:\, \mathcal{H}(X_0) = \mathcal{H}(X_1) \wedge X_0 \neq X_1 \wedge X_0, X_1 \in \mathcal{X}]\,.$$

Game PRG	**procedure** $\mathtt{G}(L)$
procedure Initialize	**if** $b = 0$
$b \twoheadleftarrow \{0,1\}$	$R \twoheadleftarrow \{0,1\}^L$
return	**return** R
	else
	$seed \twoheadleftarrow \mathcal{S}$
	$R \leftarrow \mathcal{G}(seed, L)$
	return R
	procedure Finalize (b')
	return $(b' = b)$

Fig. 16: Game used to define PRG security.

A.2 Min-Entropy

Below are some definitions and results on min-entropy that we will need in in some of the proofs.

Definition 8. *Let X, Y, and Z be random variables. The average-case guessing probability of X conditioned on Y and $Z = z$ is given by*

$$\mathrm{GP}(X|\ Y, Z = z) := \mathbb{E}_{y \leftarrow Y}\left[\max_x \Pr[X = x \mid Y = y \wedge Z = z\,]\right].$$

The average-case min-entropy of X conditioned on Y and $Z = z$ is defined to be:

$$\mathrm{H}_\infty(X \mid Y, Z = z) := -\log(\mathrm{GP}(X|\ Y, Z = z)).$$

Lemma 1 ([14]). *Let X, Y, and Z be random variables such that the support of Y is bounded above by 2^λ. Then*

$$\mathrm{H}_\infty(X \mid Y, Z = z) \geq \mathrm{H}_\infty(X, Y \mid Z = z) - \lambda \geq \mathrm{H}_\infty(X \mid Z = z) - \lambda.$$

B Security Proofs for Section 4

B.1 Proof of Theorem 2

We prove the theorem through the sequence of four games which are described in Fig. 17. The initialize and finalize procedure as well as the leakage oracles are identical for all games, hence we only list them once. Only the Enc oracle is changed in each game. Now, game G_0 corresponds to the INDaCPLA game instantiated with $\textsc{Se}[\mathcal{F}, \mathcal{G}]$, whereas game G_3 is defined so that for any adversary its probability of winning is exactly $\frac{1}{2}$.

It then follows that

$$\begin{aligned}\mathbf{Adv}^{\text{ind-acpla}}_{\textsc{Se}[\mathcal{F},\mathcal{G}]}(\mathcal{A}_{se}, \mathcal{L}_E, \mathcal{L}_D) &= 2\,\mathbf{Adv}\left(\mathsf{G}_0^{\mathcal{A}_{se}}, \mathsf{G}_3^{\mathcal{A}_{se}}\right) + 2\Pr\left[\mathsf{G}_3^{\mathcal{A}_{se}} \Rightarrow \mathsf{true}\right] - 1 \\ &= 2\sum_{i=0}^{2} \mathbf{Adv}\left(\mathsf{G}_i^{\mathcal{A}_{se}}, \mathsf{G}_{i+1}^{\mathcal{A}_{se}}\right).\end{aligned}$$

We will bound the IND-aCPLA advantage of $\mathcal{A}_{se}$ by providing bounds on the distinguishing advantages between subsequent games. We reduce the ability to distinguishing between games G_0 and G_1 to the LPRF security of $\mathcal{F}$, and that between games G_1 and G_2 to the PRG security of $\mathcal{G}$. As for G_2 and G_3, we will show that no adversary can distinguish between them, and therefore the corresponding advantage term is also zero.

Let us begin by bounding the distinguishing advantage between game G_0 and game G_1. Recall that G_0 is the INDaCPLA game instantiated with $\textsc{Se}[\mathcal{F}, \mathcal{G}]$. Game G_1 is the same, except that the challenge oracle samples *seed* at random instead of using the pseudorandom function $\mathcal{F}$. We now construct an

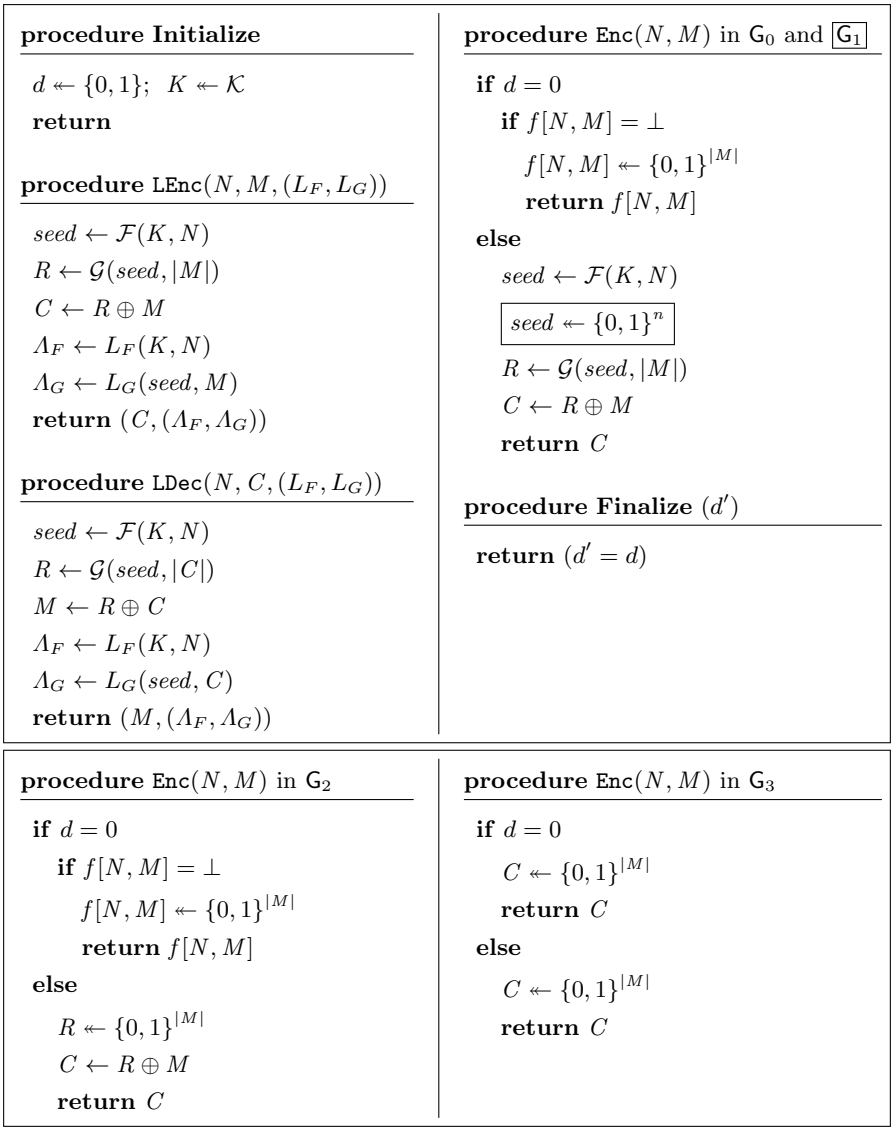

Fig. 17: Games G_0, G_1, G_2, and G_3 used to prove Theorem 2. Game G_1 contains the boxed code, G_0 does not. Games G_2 and G_3 are identical to Game G_1 except that oracle Enc is replaced with the one described.

adversary $\mathcal{A}_{lprf}$ against the LPRF security of $\mathcal{F}$ from any adversary $\mathcal{A}_{se}$ trying to distinguish between games G_0 and G_1. Note that there will be two secret bits at play, b and d. The former is the one used in the LPRF game (cf. Fig. 3) which $\mathcal{A}_{lprf}$ is trying to guess. Depending on this bit, $\mathcal{A}_{lprf}$ will simulate either G_0 or G_1 to $\mathcal{A}_{se}$. On the other hand, bit d is the secret bit sampled by $\mathcal{A}_{lprf}$ when simulating game G_0 or G_1 which $\mathcal{A}_{se}$ is trying to guess.

The adversary $\mathcal{A}_{lprf}$ starts by sampling d at random and then proceeds as follows. For any leakage query $(N, M, (L_F, L_G))$ that $\mathcal{A}_{se}$ makes to LEnc, $\mathcal{A}_{lprf}$ queries (N, L_F) to LF to obtain $(seed, \Lambda_F)$ and computes $\Lambda_G = L_G(seed, M)$. It then computes the ciphertext $C = \mathcal{G}(seed, |M|) \oplus M$ and forwards $(C, (\Lambda_F, \Lambda_G))$ to $\mathcal{A}_{se}$. Queries of the form $(N, C, (L_F, L_G))$ which $\mathcal{A}_{se}$ makes to LDec are answered just as queries to LEnc, except that the message M is obtained by XORing the ciphertext C to the output of the PRG $\mathcal{G}$ and that L_G takes C instead of M as input. For any challenge query (N, M) by $\mathcal{A}_{se}$, the behaviour of $\mathcal{A}_{lprf}$ depends on the bit d. If it is equal to 0, it returns a random bit string of length equal to the length of the message and records this bit string using the table f. If the bit d is equal to 1, $\mathcal{A}_{lprf}$ queries N to its own challenge oracle F obtaining the seed $seed$ for the pseudorandom generator $\mathcal{G}$. Subsequently, it computes $C = \mathcal{G}(seed, |M|) \oplus M$ and sends C to $\mathcal{A}_{se}$. Eventually, $\mathcal{A}_{se}$ outputs a bit d' to indicate whether it was in the ideal or in the real world. If $\mathcal{A}_{se}$ guessed correctly, i.e. $d' = d$, $\mathcal{A}_{lprf}$ outputs $b' \leftarrow 0$. Otherwise, $\mathcal{A}_{lprf}$ outputs $b' \leftarrow 1$.

From its leakage oracle, $\mathcal{A}_{lprf}$ obtains the leakage $\Lambda_F = L_F(K, N)$ and the seed for the pseudorandom generator. It then uses the seed to compute the ciphertext C or the message M together with the leakage Λ_G. When $d = 0$ (ideal world), $\mathcal{A}_{lprf}$ can easily simulate the challenge oracle by sampling random ciphertexts. However, when $d = 1$ (real world), $\mathcal{A}_{lprf}$ uses its own challenge oracle to obtain the seed for $\mathcal{G}$. Then depending on the bit b, the seed will be the output of a pseudorandom function ($b = 1$) or sampled randomly ($b = 0$). Since the games G_0 and G_1 only differ in how the seed for the PRG is generated, depending on the value of b, $\mathcal{A}_{lprf}$ provides $\mathcal{A}_{se}$ with a perfect simulation of either G_0 or G_1.

It remains to verify that $\mathcal{A}_{lprf}$ is a valid LPRF adversary. By construction, as long as $\mathcal{A}_{se}$ queries leakage functions from the permitted leakage set then so will $\mathcal{A}_{lprf}$. For any leakage query $(N, \cdot, L = (L_F, \cdot))$ to either $\texttt{LEnc}$ or $\texttt{LDec}$ by $\mathcal{A}_{se}$, $\mathcal{A}_{lprf}$ queries $\texttt{LF}$ on (N, L_F). It remains to argue that $\mathcal{A}_{lprf}$ does not forward queries to or from its $\texttt{F}$ oracle. Note that $\mathcal{A}_{lprf}$ only makes queries to $\texttt{F}$ to answer $\mathcal{A}_{se}$'s queries to $\texttt{Enc}$. Then since $\mathcal{A}_{se}$ must be nonce-respecting, it cannot reuse the same nonce across $\texttt{Enc}$ and $\texttt{LEnc}$. Moreover, all of its queries to $\texttt{LDec}$ must be forwarded from $\texttt{LEnc}$. Thus the nonces between $\texttt{Enc}$ and the leakage oracles have to be disjoint and therefore $\mathcal{A}_{lprf}$'s queries to $\texttt{F}$ and $\texttt{LF}$ will also be disjoint. We are now ready to derive our first bound. From the foregoing discussion we have that

$$\begin{aligned}\mathbf{Adv}\left(\mathsf{G}_0^{\mathcal{A}_{se}}, \mathsf{G}_1^{\mathcal{A}_{se}}\right) &= \Pr[\mathsf{G}_0^{\mathcal{A}_{se}} \Rightarrow \mathsf{true}] - \Pr[\mathsf{G}_1^{\mathcal{A}_{se}} \Rightarrow \mathsf{true}] \\ &\leq \Pr[\mathcal{A}_{lprf}^{\mathsf{LPRF}} \Rightarrow 0 \,|\, b = 1] - \Pr[\mathcal{A}_{lprf}^{\mathsf{LPRF}} \Rightarrow 0 \,|\, b = 0] \\ &\leq \mathbf{Adv}_{\mathcal{F}}^{\text{lprf}}\left(\mathcal{A}_{lprf}, \mathcal{L}_F\right) .\end{aligned}$$

Next, we bound the advantage in distinguishing G_1 from G_2. The only difference between these two games is how R is generated, in G_1 this is generated as $\mathcal{G}(\mathit{seed})$ whereas in G_2 it is sampled uniformly at random. For any adversary $\mathcal{A}_{se}$ trying to distinguish between G_1 and G_2 we construct a PRG adversary $\mathcal{A}_{prg}$ against $\mathcal{G}$ as follows. It starts by sampling a random key K for the PRF $\mathcal{F}$ and a random bit d. For any query $(N, M, (L_F, L_G))$ to $\texttt{LEnc}$ that $\mathcal{A}_{se}$ makes, $\mathcal{A}_{prg}$ computes locally $\mathit{seed} \leftarrow \mathcal{F}(K, N)$, $R \leftarrow \mathcal{G}(\mathit{seed}, |M|)$, and $C \leftarrow R \oplus M$. It then computes $\Lambda_F \leftarrow L_F(K, N)$ as well as $\Lambda_G \leftarrow L_G(\mathit{seed}, M)$ and returns $(C, (\Lambda_F, \Lambda_G))$ to $\mathcal{A}_{se}$. Leakage queries to $\texttt{LDec}$ are handled similarly, only XORing the ciphertext to the output of the PRG $\mathcal{G}$ and feeding the ciphertext as input to L_G. For any challenge query (N, M) by $\mathcal{A}_{se}$, when $d = 0$ $\mathcal{A}_{prg}$ returns a random bit string of length equal to $|M|$. Otherwise, if $d = 1$ $\mathcal{A}_{prg}$ queries its own challenge oracle $\texttt{G}$ on $|M|$ to obtain R and returns $C \leftarrow R \oplus M$ to $\mathcal{A}_{se}$. Then, when $\mathcal{A}_{se}$ outputs a guess d', $\mathcal{A}_{prg}$ outputs 0 if $d' = d$ and outputs 1 otherwise.

Now note that if $b = 0$ then $\mathcal{A}_{prg}$ provides $\mathcal{A}_{se}$ with simulation of G_2 and if $b = 1$ it simulates G_1. However when $b = 1$, $\mathcal{A}_{prg}$ evaluates $\mathcal{G}$ both locally and through its oracle $\texttt{G}$ on independently sampled seeds. Since $\mathcal{A}_{se}$ cannot reuse the same nonce across queries to $\texttt{Enc}$ and its leakage oracles, these two sets of queries are always answered using independently-sampled seeds and its simulation of G_2 is perfect. We then have that

$$\begin{aligned}\mathbf{Adv}\left(\mathsf{G}_1^{\mathcal{A}_{se}}, \mathsf{G}_2^{\mathcal{A}_{se}}\right) &= \Pr[\mathsf{G}_1^{\mathcal{A}_{se}} \Rightarrow \mathsf{true}] - \Pr[\mathsf{G}_2^{\mathcal{A}_{se}} \Rightarrow \mathsf{true}] \\ &\leq \Pr[\mathcal{A}_{prg}^{\mathsf{PRG}} \Rightarrow 0 \,|\, b = 1] - \Pr[\mathcal{A}_{prg}^{\mathsf{PRG}} \Rightarrow 0 \,|\, b = 0] \\ &\leq \mathbf{Adv}_{\mathcal{G}}^{\text{prg}}\left(\mathcal{A}_{prg}\right) .\end{aligned}$$

We now conclude the proof by bounding the distinguishing advantage between the games G_2 and G_3. These two games differ in two aspects. In the ideal world ($d = 0$), the former outputs a random bit string using a table f while the latter forgetfully outputs a random bit string of the same size. However as long as $\mathcal{A}_{se}$ is nonce-respecting it will not be able to distinguish these two cases since every output will be freshly sampled. The other difference is how C is generated in the real world ($d = 1$). However both methods yield identically distributed ciphertexts. Thus the two games are identical and therefore

$$\mathbf{Adv}\left(\mathsf{G}_2^{\mathcal{A}_{se}}, \mathsf{G}_3^{\mathcal{A}_{se}}\right) = 0 .$$

Collecting all of the above bounds together yields

$$\mathbf{Adv}_{\mathrm{SE}[\mathcal{F},\mathcal{G}]}^{\text{ind-acpla}}\left(\mathcal{A}_{se}, \mathcal{L}_E, \mathcal{L}_D\right) \leq 2\,\mathbf{Adv}_{\mathcal{F}}^{\text{lprf}}\left(\mathcal{A}_{lprf}, \mathcal{L}_F\right) + 2\,\mathbf{Adv}_{\mathcal{G}}^{\text{prg}}\left(\mathcal{A}_{prg}\right) ,$$

which proves the theorem. □

B.2 Proof of Theorem 3

For any SUF-CMLA adversary $\mathcal{A}_{mac}$ against $\textsc{Mac}[\hat{\mathcal{F}}]$ we construct a LUF adversary $\mathcal{A}_{luf}$ against $\hat{\mathcal{F}}$. We proceed in two steps, we first bound the SUF-CMLA advantage of $\mathcal{A}_{mac}$ by the probability of a particular event occurring and then we bound this probability by the LUF advantage of $\mathcal{A}_{luf}$. Let E be the event that the $\mathcal{A}_{mac}$ makes a query to `Vfy` which returns $\top$. Then

$$\begin{aligned}
\mathbf{Adv}^{\text{suf-cmla}}_{\textsc{Mac}[\hat{\mathcal{F}}]}(\mathcal{A}_{mac}, \mathcal{L}_T, \mathcal{L}_V) &= 2\Pr\left[\mathsf{SUFCMLA}^{\mathcal{A}_{mac}} \Rightarrow \mathsf{true}\right] - 1 \\
&= 2\Pr\left[\mathsf{SUFCMLA}^{\mathcal{A}_{mac}} \Rightarrow \mathsf{true} \cap E\right] \\
&\qquad + 2\Pr\left[\mathsf{SUFCMLA}^{\mathcal{A}_{mac}} \Rightarrow \mathsf{true} \cap \overline{E}\right] - 1 \\
&\leq 2\Pr[E] + 2\Pr\left[\mathsf{SUFCMLA}^{\mathcal{A}_{mac}} \Rightarrow \mathsf{true} \,|\, \overline{E}\right] - 1\,.
\end{aligned}$$

Now, conditioned on $\overline{E}$ the `Vfy` oracle will always returns $\perp$ irrespective of the value of b and hence the probability of $\mathcal{A}_{mac}$ winning is exactly one half. Thus the above can be re-written as

$$\leq 2\Pr[E \cap b = 0] + 2\Pr[E \cap b = 1]\ .$$

When $b = 0$ the `Vfy` oracle will always returns $\perp$ and thus E simply cannot occur, i.e. the first term in the above expression is zero. Thus

$$\begin{aligned}
&= 2\Pr[E \,|\, b = 1]\Pr[b = 1] \\
&= \Pr[E \,|\, b = 1]\ .
\end{aligned}$$

We now bound this last probability by the LUF advantage of $\mathcal{A}_{luf}$. By construction we have that $\mathcal{L}_T = \mathcal{L}_V = \mathcal{L}_{\hat{F}}$. Then $\mathcal{A}_{luf}$ runs $\mathcal{A}_{mac}$ and provides it with a simulation of the $\mathsf{SUFCMLA}$ game with the bit b fixed to 1. Whenever $\mathcal{A}_{mac}$ makes a query (X, L) to `LTag`, $\mathcal{A}_{luf}$ queries X to `F` and (X, L) to `Lkg` to obtain respectively $y = \hat{\mathcal{F}}(K, X)$ and $\Lambda = L(K, X)$, and returns (y, Λ) back to $\mathcal{A}_{mac}$. Similarly if $\mathcal{A}_{mac}$ queries (X, T, L) to its `LVfy` oracle, $\mathcal{A}_{luf}$ queries (X, L) to `Lkg` and (X, T) to `Guess` to obtain Λ and v respectively, and returns (v, Λ) to $\mathcal{A}_{mac}$. Every query (X, T) that $\mathcal{A}_{mac}$ makes to its `Vfy` oracle, is forwarded by $\mathcal{A}_{luf}$ to its own `Guess` oracle and returns $\top$ to $\mathcal{A}_{mac}$ if `Guess` returns `true` and $\perp$ otherwise.

Recall that $\mathcal{A}_{luf}$'s queries to `F` are recorded in the set $\mathcal{S}$, and it only wins the LUF game if it makes a query to `Guess` which returns `true` where the corresponding X value is not contained in $\mathcal{S}$. However since $\mathcal{A}_{mac}$ does not forward queries from `LTag` to `Vfy`, it is guaranteed that $\mathcal{A}_{luf}$'s queries to `Guess` are not contained in $\mathcal{S}$. It then follows that $\mathcal{A}_{luf}$ wins the LUF game whenever E occurs, and hence

$$\Pr[E \,|\, b = 1] \leq \mathbf{Adv}^{\text{luf}}_{\hat{\mathcal{F}}}\left(\mathcal{A}_{luf}, \mathcal{L}_{\hat{F}}\right)$$

which proves the theorem. □

B.3 Proof of Theorem 4

We prove the Theorem through the two games, G and $\boxed{\mathsf{G}}$, as described in Fig. 18. Namely:

- **Game** G is the LUF game instantiated with $\hat{\mathcal{F}}$ and its associated leakage set $\mathcal{L}_{\hat{F}} = \mathcal{L}_{F'} \times \mathcal{L}_H$, and some additional bookkeeping to maintain the set Ω and test for the event Bad.
- **Game** $\boxed{\mathsf{G}}$ is identical to game G except that the winning condition is restricted to only hold if Bad does not occur.

Moreover G and $\boxed{\mathsf{G}}$ are identical until Bad, and therefore by the game playing lemma we have that

$$\begin{aligned}
\mathbf{Adv}\left(\mathsf{G}^{\mathcal{A}_{luf}}, \boxed{\mathsf{G}}^{\mathcal{A}_{luf}}\right) &= \Pr[\mathsf{G}^{\mathcal{A}_{luf}} \Rightarrow \mathsf{true}] - \Pr[\boxed{\mathsf{G}}^{\mathcal{A}_{luf}} \Rightarrow \mathsf{true}] \\
&\leq \Pr[\mathsf{Bad}]\,.
\end{aligned}$$

Then, through some simple algebraic manipulation and substituting for the LUF advantage of $\hat{\mathcal{F}}$ we get

$$\mathbf{Adv}^{\text{luf}}_{\hat{\mathcal{F}}}\left(\mathcal{A}_{luf}, \mathcal{L}_{\hat{F}}\right) \leq 2\ \Pr[\mathsf{Bad}] + \left(2\ \Pr[\boxed{\mathsf{G}}^{\mathcal{A}_{luf}} \Rightarrow \mathsf{true}] - 1\right)\ . \tag{1}$$

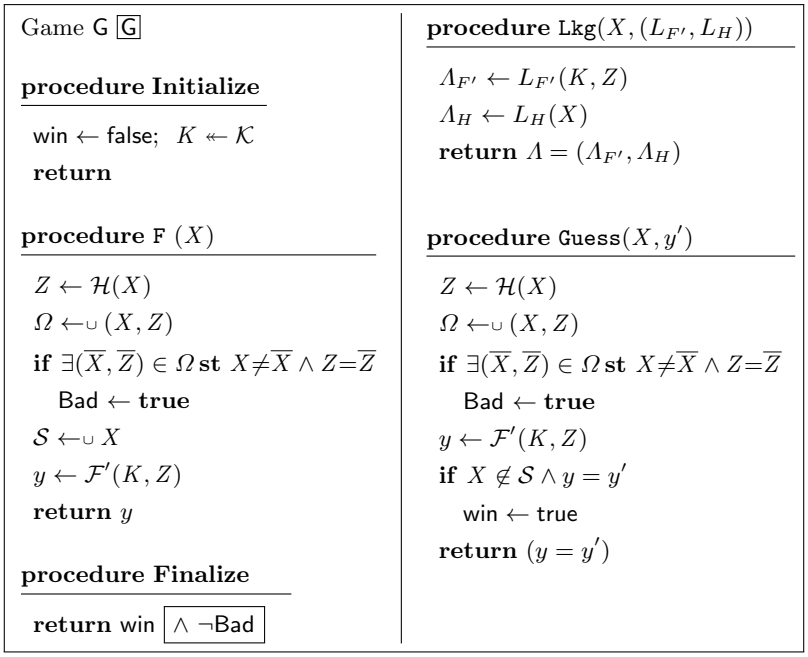

Fig. 18: Games G and [G] used to prove Theorem 4. Game [G] includes the boxed code whereas G does not.

We now bound the two terms on the right-hand side of the above inequality. We start by bounding the probability that $\mathcal{A}_{luf}$ sets the flag Bad to true in game G. Towards this end we construct from $\mathcal{A}_{luf}$ another adversary $\mathcal{A}_{hash}$ for breaking the collision resistance of $\mathcal{H}$. The adversary $\mathcal{A}_{hash}$ samples a key for $\mathcal{F}'$, runs $\mathcal{A}_{luf}$ and provides it with a simulation of G. Throughout the game it maintains the set Ω and constantly checks for colliding pairs in this set. As soon as it finds a collision, i.e. Bad is set to true, it halts and outputs this colliding pair. Clearly

$$\Pr[\mathsf{Bad}] \leq \mathbf{Adv}^{\mathrm{cr}}_{\mathcal{H}}(\mathcal{A}_{hash}) \ . \tag{2}$$

Now, to bound the second term of inequality (1), we construct from $\mathcal{A}_{luf}$ another adversary $\mathcal{A}'_{luf}$ against the LUF security of $\mathcal{F}'$. Specifically, $\mathcal{A}'_{luf}$ runs $\mathcal{A}_{luf}$ and provides it with a simulation of the [G] game. For any query that $\mathcal{A}_{luf}$ makes to any of its oracles which includes the value X, $\mathcal{A}'_{luf}$ simply replaces this value with $\mathcal{H}(X)$ and forwards this query to its corresponding oracle and then passes the reply back to $\mathcal{A}_{luf}$. In the case of a leakage query, $\mathcal{A}'_{luf}$ additionally evaluates $L_H(X)$ locally and combines it with the response it obtains from its oracle before forwarding it back to $\mathcal{A}_{luf}$. By construction, the leakage queries of $\mathcal{A}'_{luf}$ will be in the permitted leakage set as long as $\mathcal{A}_{luf}$'s queries are. Moreover, whenever $\mathcal{A}_{luf}$ wins the [G] game then $\mathcal{A}'_{luf}$ also wins the LUF game with respect to $\mathcal{F}'$. This is because if $\mathcal{A}_{luf}$ wins [G] then Bad must not have occurred, thus if the winning query (X, y') is such that $X \notin \mathcal{S}$ then $\mathcal{H}(X)$ is also not contained in the corresponding set in the LUF game of $\mathcal{A}'_{luf}$. It then follows that

$$\left(2\Pr[\boxed{\mathsf{G}}^{\mathcal{A}_{luf}} \Rightarrow \mathsf{true}] - 1\right) \leq \mathbf{Adv}^{\mathrm{luf}}_{\mathcal{F}'}(\mathcal{A}'_{luf}, \mathcal{L}_{F'}) \ . \tag{3}$$

Combining inequalities (1), (2), and (3) yields the desired result. □

B.4 Proof of Theorem 5

We prove the Theorem through the two games, G and [G], described in Fig. 19. Namely:

- **Game** G is the LPRF game instantiated with $\hat{\mathcal{F}}$ and its associated leakage set $\mathcal{L}_{\hat{F}} = \mathcal{L}_{F'} \times \mathcal{L}_H$, and some additional bookkeeping to maintain the sets Ω_1 and Ω_2 and test for events Bad^1 and Bad^2.
- **Game** [G] is identical to game G except that the F oracle returns the same output for any two inputs which hash to the same value, even when $d = 0$, and the winning condition is restricted to only hold if Bad^1 does not occur.

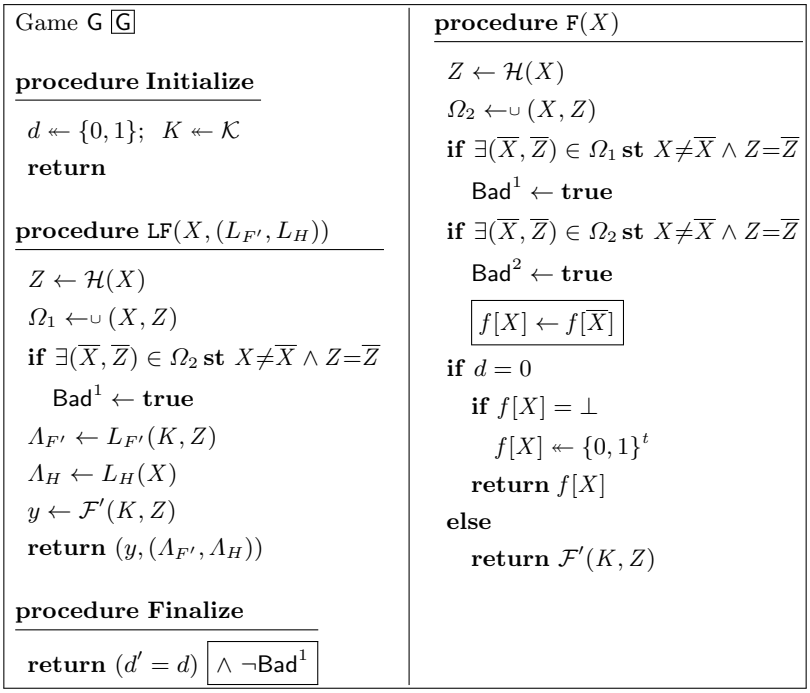

Fig. 19: Games G and $\boxed{\mathsf{G}}$ used to prove Theorem 5. Game $\boxed{\mathsf{G}}$ includes the boxed code whereas G does not.

Moreover G and $\boxed{\mathsf{G}}$ are identical until $\mathsf{Bad} = \mathsf{Bad}^1 \cup \mathsf{Bad}^2$, and therefore by the game playing lemma we have that

$$\begin{aligned}\mathbf{Adv}\left(\mathsf{G}^{\mathcal{A}_{lprf}}, \boxed{\mathsf{G}}^{\mathcal{A}_{lprf}}\right) &= \Pr[\mathsf{G}^{\mathcal{A}_{lprf}} \Rightarrow \mathsf{true}] - \Pr[\boxed{\mathsf{G}}^{\mathcal{A}_{lprf}} \Rightarrow \mathsf{true}] \\ &\leq \Pr[\mathsf{Bad}]\,.\end{aligned}$$

Then, through some simple algebraic manipulation and substituting for the LPRF advantage of $\hat{\mathcal{F}}$ we get

$$\mathbf{Adv}^{\mathrm{lprf}}_{\hat{\mathcal{F}}}\left(\mathcal{A}_{lprf}, \mathcal{L}_{\hat{F}}\right) \leq 2\,\Pr[\mathsf{Bad}^1 \cup \mathsf{Bad}^2] + \left(2\,\Pr[\boxed{\mathsf{G}}^{\mathcal{A}_{lprf}} \Rightarrow \mathsf{true}] - 1\right)\,. \tag{4}$$

We now bound the two terms on the right-hand side of the above inequality. We start by bounding the probability that Bad occurs, i.e. $\mathcal{A}_{lprf}$ sets Bad^1 or Bad^2 to true in game G. Towards this end we construct from $\mathcal{A}_{lprf}$ another adversary $\mathcal{A}_{hash}$ for breaking the collision resistance of $\mathcal{H}$. The adversary $\mathcal{A}_{hash}$ samples a key for $\mathcal{F}'$, runs $\mathcal{A}_{lprf}$ and provides it with a simulation of G. Throughout the game it maintains the sets Ω_1 and Ω_2 and constantly checks for the conditions that set Bad^1 or Bad^2 to true. When either of these conditions occur a collision has been found, at which point $\mathcal{A}_{hash}$ halts and outputs the colliding pair. It then follows that

$$\Pr[\mathsf{Bad}^1 \cup \mathsf{Bad}^2] \leq \mathbf{Adv}^{\mathrm{cr}}_{\mathcal{H}}\left(\mathcal{A}_{hash}\right)\,. \tag{5}$$

Now, to bound the second term of inequality (4), we construct from $\mathcal{A}_{lprf}$ another adversary $\mathcal{A}'_{lprf}$ against the LPRF security of $\mathcal{F}'$. Specifically, $\mathcal{A}'_{lprf}$ runs $\mathcal{A}_{lprf}$ and provides it with a simulation of the $\boxed{\mathsf{G}}$ game. For any query that $\mathcal{A}_{lprf}$ makes to either of its oracles with the value X, $\mathcal{A}'_{lprf}$ simply replaces this value with $\mathcal{H}(X)$ and forwards this query to its corresponding oracle and then passes the reply back to $\mathcal{A}_{lprf}$. In the case of a leakage query, $\mathcal{A}'_{lprf}$ additionally evaluates $L_H(X)$ locally and combines it with the response it obtains from its oracle before forwarding it back to $\mathcal{A}_{lprf}$. Now, since in $\boxed{\mathsf{G}}$ any two queries to F that hash to the same value will return the same output (irrespective of the value d) $\mathcal{A}'_{lprf}$'s simulation is perfect. The leakage queries of $\mathcal{A}'_{lprf}$ will be in the permitted leakage set as long as $\mathcal{A}_{lprf}$'s queries are. Moreover, as long as Bad^1 does not occur the queries of $\mathcal{A}'_{lprf}$ to its F and LF oracles will be disjoint as long as $\mathcal{A}_{lprf}$'s queries are disjoint. Thus whenever $\mathcal{A}_{lprf}$ wins game $\boxed{\mathsf{G}}$, $\mathcal{A}'_{lprf}$ will also win

the LPRF game with respect to $\mathcal{F}'$, because if $\mathcal{A}_{lprf}$ wins $\boxed{\mathsf{G}}$ then Bad^1 must not have occurred. It then follows that

$$\left(2\,\Pr[\boxed{\mathsf{G}}^{\mathcal{A}_{lprf}} \Rightarrow \mathsf{true}] - 1\right) \leq \mathbf{Adv}^{\mathrm{lprf}}_{\mathcal{F}'}\left(\mathcal{A}'_{lprf}, \mathcal{L}_{F'}\right) . \tag{6}$$

Combining inequalities (4), (5), and (6) yields the desired result. □

C Security Proofs for Section 5

C.1 Proof of Theorem 7

Game G_1	**procedure** $\mathtt{LF}(X, L_0, \ldots, L_l)$
Initialize	**parse** X **as** $X_1 \parallel \ldots \parallel X_l$ $\forall i\ \lvert X_i\rvert = rr$
$K \twoheadleftarrow \{0,1\}^k$	$Y_0 \leftarrow K \parallel IV$
procedure $\rho(Z)$	$\Lambda \leftarrow L_0(Y_0)$
if $p[Z] = \perp$	**if** $p[Y_0] = \perp$
$p[Z] \twoheadleftarrow \{0,1\}^n$	$p[Y_0] \twoheadleftarrow \{0,1\}^n$
return $p[Z]$	$S_1 \leftarrow p[Y_0]$
procedure $\mathtt{F}(X)$	**for** i **in** $\{1, \ldots, l\}$
parse X **as** $X_1 \parallel \ldots \parallel X_l$ $\forall i\ \lvert X_i\rvert = rr$	$Y_i \leftarrow (\bar{S}_i \oplus X_i) \parallel \hat{S}_i$
$Y_0 \leftarrow K \parallel IV$	$\Lambda \leftarrow \Lambda \parallel L_i(Y_i)$
if $p[Y_0] = \perp$	**if** $p[Y_i] = \perp$
$p[Y_0] \twoheadleftarrow \{0,1\}^n$	$p[Y_i] \twoheadleftarrow \{0,1\}^n$
$S_1 \leftarrow p[Y_0]$	$S_{i+1} \leftarrow p[Y_i]$
for i **in** $\{1, \ldots, l\}$	**return** (S_{l+1}, Λ)
$Y_i \leftarrow (\bar{S}_i \oplus X_i) \parallel \hat{S}_i$	
if $p[Y_i] = \perp$	
$p[Y_i] \twoheadleftarrow \{0,1\}^n$	
$S_{i+1} \leftarrow p[Y_i]$	
return S_{l+1}	

Fig. 20: Game G_1 used in the proof of Theorem 7.

We prove the theorem for the case where SLFUNC takes inputs of fixed length $(l \cdot rr)$ and returns outputs of length $t = n$. The general case then follows by means of a simple reduction which truncates the output of SLFUNC to t bits. The proof proceeds by considering the following sequence of games.

G_1 This game is described in Fig. 20. It is the LPRF game instantiated with SLFUNC and the bit b set to 1, i.e. $\mathtt{F}$ always returns real evaluations of SLFUNC. Furthermore the adversary is also given oracle access to the random transformation ρ which SLFUNC depends on. This random transformation is sampled lazily across all oracles and the corresponding values are stored in a global array $p[\,]$. All entries of $p[\,]$ are initialised to $\perp$. Thus,

$$\Pr[\mathcal{A}^{\mathsf{LPRF}} \Rightarrow 1 \mid b = 1] = \Pr[\mathcal{A}^{\mathsf{G}_1} \Rightarrow 1] . \tag{7}$$

$\boxed{\mathsf{G}_2}$ In this game, described in Fig. 21, we introduce three flags Bad^1_2, Bad^2_2, and Bad^3_2 and evaluate the last round of the $\mathtt{F}$ oracle in a different but equivalent way. Here the transformation used in the last round of $\mathtt{F}$ is considered to be special and its values are stored in $p^*[\,]$ instead. However, as we lazy-sample $p[\,]$ and $p^*[\,]$ we ensure that they are aligned on common input values. When one array is to be sampled on an input for which the other array has already been defined, then one of the three

Bad_2 flags will be set and the output for that array is copied from the one that has already been defined. Note that in comparison to G_1 the sampling of the transformation's outputs is unaltered, we are merely storing them across two arrays instead of one. Thus, the functionality of the game and the distribution of its random variables is unchanged, and hence

$$\Pr[\mathcal{A}^{\boxed{\mathsf{G}_2}} \Rightarrow 1] = \Pr[\mathcal{A}^{\mathsf{G}_1} \Rightarrow 1]\,. \tag{8}$$

G_2 This game is the game described in Fig. 21 without the boxed code. By removing the boxed code we no longer maintain consistency between $p[\,]$ and $p^*[\,]$ on common inputs. In particular, the last round of $\mathtt{F}$ is now evaluated using an independent random transformation. Clearly, G_2 and $\boxed{\mathsf{G}_2}$ are identical until the event $\mathsf{Bad}_2^1 \cup \mathsf{Bad}_2^2 \cup \mathsf{Bad}_2^3$ occurs. Then, by the fundamental lemma of game-playing it follows that:

$$\Pr[\mathcal{A}^{\mathsf{G}_2} \Rightarrow 1] - \Pr[\mathcal{A}^{\boxed{\mathsf{G}_2}} \Rightarrow 1] \leq \Pr[\mathsf{Bad}_2^1 \cup \mathsf{Bad}_2^2 \cup \mathsf{Bad}_2^3]\,.$$

Moreover, for events C and E such that $\mathsf{Bad}_2^3 \subseteq C$ and $\mathsf{Bad}_2^1 \subseteq E$, we have that

$$\begin{aligned} &\leq \Pr[E \cup \mathsf{Bad}_2^2 \cup C]\,, \\ &\leq \Pr[C] + \Pr[E \mid \overline{C}] + \Pr[\mathsf{Bad}_2^2 \mid \overline{C}, \overline{E}]\,. \end{aligned} \tag{9}$$

We now define event C appropriately and bound the above three probabilities. Let q_ρ, $q_{\mathtt{LF}}$, and $q_{\mathtt{F}}$ denote the number of queries $\mathcal{A}$ makes to its oracles ρ, $\mathtt{LF}$, and $\mathtt{F}$, respectively. We can further assume, without loss of generality, that $\mathcal{A}$ never repeats a query to any of its oracles.
If Bad_2^3 is set $\mathcal{A}$ must have made two distinct queries, X to $\mathtt{F}$ and X' to $\mathtt{LF}$, such that $\textsc{SlFunc}(K, X) = \textsc{SlFunc}(K, X')$. Then, note that $\textsc{SlFunc}(K, X)$ can be viewed as the sponge-based hash function from Section 5.4 evaluated on input $0 \,\|\, X$ with an initial state of $K \,\|\, IV$ (instead of 0^n) and capacity $n - rr$. We will use this observation to bound the probability of event C (and Bad_3^1) occurring, but let us first define event C. Let $\mathcal{H}^{K\|IV}$ be the variable-input-length hash function just described, then C is the event that one of the following conditions is satisfied.

(i) $\mathcal{A}$ makes queries X and X' across $\mathtt{F}$ and $\mathtt{LF}$ such that $\mathcal{H}^{K\|IV}(0 \,\|\, X) = \mathcal{H}^{K\|IV}(0 \,\|\, X')$ and $X \neq X'$.

(ii) $\mathcal{A}$ makes queries X and X' across $\mathtt{F}$ and $\mathtt{LF}$ such that $\mathcal{H}^{K\|IV}(0 \,\|\, X) = \mathcal{H}^{K\|IV}(0 \,\|\, X'')$ for some X'' that is a prefix of X' and $|X''| < |X'|$.

(iii) $\mathcal{A}$ makes a query X to $\mathtt{F}$ or $\mathtt{LF}$ such that $\mathcal{H}^{K\|IV}(0 \,\|\, X') = K \,\|\, IV$ for some X' that is a prefix of X.

(iv) $\mathcal{A}$ makes queries X and X' across $\mathtt{F}$ and $\mathtt{LF}$ such that $\mathcal{H}^{K\|IV}(0 \,\|\, X'') = \mathcal{H}^{K\|IV}(0 \,\|\, X''')$, where X'' is a prefix of X, X''' is a prefix of X', $X'' \neq \varepsilon \neq X'''$, and $X'' \neq X'''$.

Note that $\mathsf{Bad}_2^3 \subseteq C$ is implied by the first condition. Now for any adversary $\mathcal{A}$ that causes C, we can construct an adversary $\mathcal{B}$ that finds a collision in $\mathcal{H}^{K\|IV}$. Specifically, $\mathcal{B}$ knows K and uses this together with its access to ρ to simulate game G_2 to $\mathcal{A}$ while constantly checking if event C has occurred. Note that any of the sub-cases in C will directly enable $\mathcal{B}$ to produce a collision on $\mathcal{H}^{K\|IV}$. We can therefore use the security bound derived in Section 5.4 for $\mathcal{H}$ since it applies to any initial state that $\mathcal{H}$ is assigned, as long as it is fixed at the start of the game. Thus, applying inequality (49) from Appendix C.4 while setting $w = n$ and $c' = n - rr$ yields

$$\Pr[\mathsf{Bad}_2^3] \leq \Pr[C] \leq \frac{q_T(q_T + 2)}{2^{n-rr}}\,, \tag{10}$$

where $q_T = (l+1)(q_{\mathtt{LF}} + q_{\mathtt{F}}) + q_\rho$.
We now define event E and bound $\Pr[E \mid \overline{C}]$. At any point in time, let $\mathcal{V}_{\mathtt{F}}$ and $\mathcal{V}_{\mathtt{LF}}$ denote the set unions of intermediate values $\{Y_1, \ldots, Y_l\}$ evaluated in $\mathtt{F}$ and $\mathtt{LF}$ respectively over all queries made up to that point. Further let E_j be the event that the j^{th} query Z that $\mathcal{A}$ makes to ρ is such that $Z = Y_0$ or $Z \oplus A \,\|\, 0^{n-rr} \in \mathcal{V}_{\mathtt{F}}$ for some $A \in \{0,1\}^{rr}$. Then E represents the union of events E_j for $j \in \{1, \ldots, q_\rho\}$. Note that $\mathsf{Bad}_2^1 \subseteq E$ as required, and we also have that

$$\begin{aligned} \Pr[E \mid \overline{C}] &\leq \Pr[E_1 \cup E_2 \cup \ldots \cup E_{q_\rho} \mid \overline{C}] \\ &= \Pr[E_1 \mid \overline{C}] + \Pr[E_2 \mid \overline{E}_1, \overline{C}] + \ldots \\ &\qquad + \Pr[E_{q_\rho} \mid \overline{E}_1, \ldots, \overline{E}_{q_\rho - 1}, \overline{C}]\,. \end{aligned} \tag{11}$$

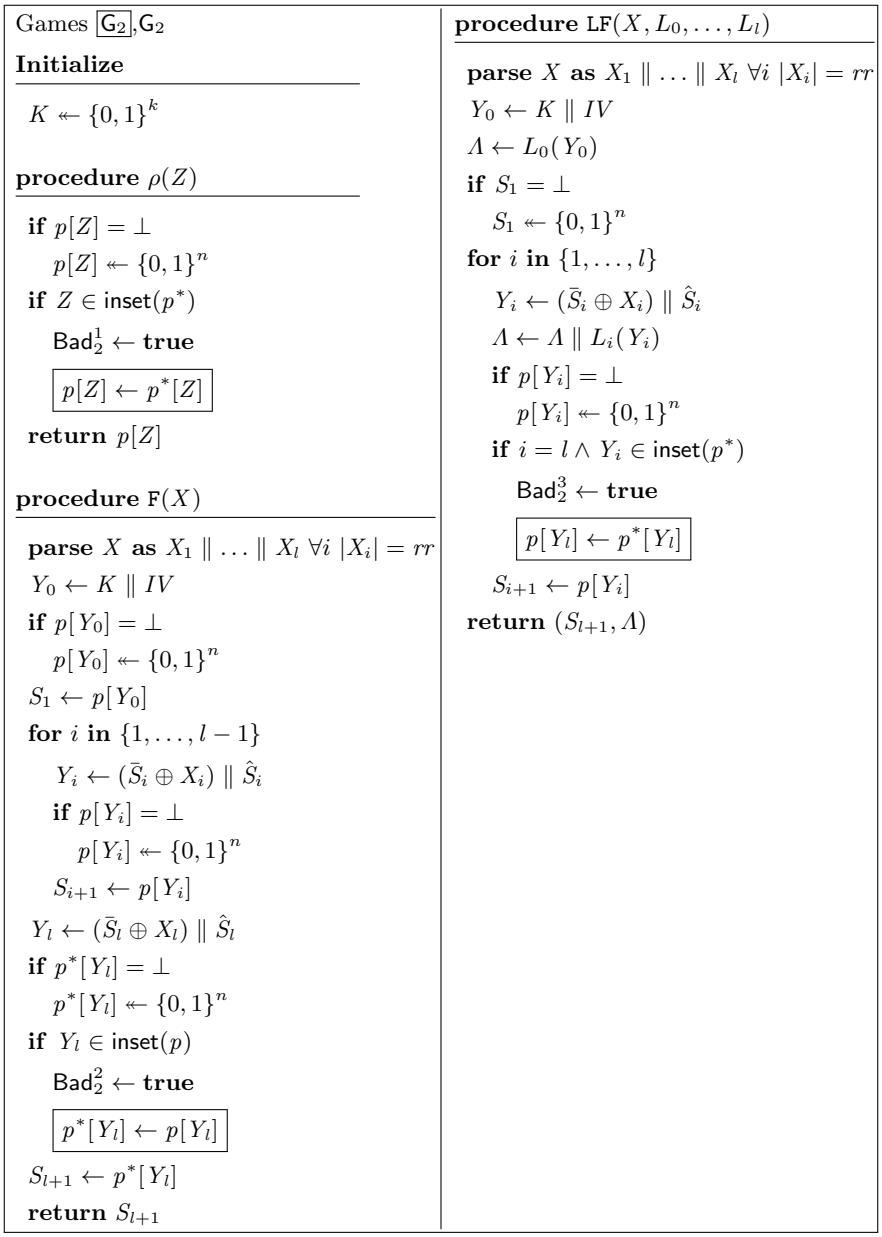

Fig. 21: Games $\boxed{\mathsf{G}_2}$ and G_2 used in the proof of Theorem 7. Game $\boxed{\mathsf{G}_2}$ includes the boxed code whereas G_2 does not.

Now as long as C does not occur no two queries will result in values Y_i and $Y_{i'}$ such that $Y_i = Y_{i'}$ and $i \neq i'$. Thus no state variable $p[Y_i]$ occurs in more than one position (when considering both $\mathtt{F}$ and $\mathtt{LF}$), meaning that no state variable is subject to more than one leakage function and that no internal state variable is ever exposed as the output of $\mathtt{F}$ or $\mathtt{LF}$. Moreover, as long as E does not occur, the outputs of ρ will never expose the internal state variables of $\mathtt{F}$ and $\mathtt{LF}$. Then, for $i \geq 1$, guessing a value $Y_i \in \mathcal{V}_{\mathtt{F}}$ is tantamount to guessing the corresponding S_i, since $Y_i = S_i \oplus (X_i \,\|\, 0^{n-rr})$ and X_i is known to the adversary. Then by the union bound and the above observations it follows that

$$\begin{aligned}\Pr[E_j \,|\, \overline{E}_1, \ldots, \overline{E}_{j-1}, \overline{C}] \leq & \sum_{Y_i \in \mathcal{V}_{\mathtt{F}}} 2^{-\mathrm{H}_\infty\left(Y_i \,|\, \Lambda, \overline{E}_1, \ldots, \overline{E}_{j-1}, \overline{C}\right)} \\ + & \quad 2^{-\mathrm{H}_\infty\left(K \,|\, \Lambda, \overline{E}_1, \ldots, \overline{E}_{j-1}, \overline{C}\right)} \\ = & \sum_{Y_i \in \mathcal{V}_{\mathtt{F}}} 2^{-\mathrm{H}_\infty\left(S_i \,|\, \Lambda, \overline{E}_1, \ldots, \overline{E}_{j-1}, \overline{C}\right)} \\ + & \quad 2^{-\mathrm{H}_\infty\left(K \,|\, \Lambda, \overline{E}_1, \ldots, \overline{E}_{j-1}, \overline{C}\right)}. \end{aligned} \tag{12}$$

Although $\mathtt{F}$ does not leak we have to take into account that the same state variable may result internally in a distinct query to $\mathtt{LF}$ and thereby be subject to leakage. With respect to the key the adversary only learns $L_0(K \,\|\, IV)$ from the leakage. However, for a given S_i the adversary may obtain (at most) the leakage $L_i(S_i \oplus (X_i \,\|\, 0^{n-rr}))$ for each possible value of X_i. If we consider the aggregate leakage for a given S_i over all possible values of X_i, i.e. $L_i(S_i \oplus 0^{rr} \,\|\, 0^{n-rr}) \,\|\, \ldots \,\|\, L_i(S_i \oplus 1^{rr} \,\|\, 0^{n-rr})$, the support of this aggregate random variable is bounded by $2^{\lambda 2^{rr}}$. Then by applying Lemma 1 we have that

$$\mathrm{H}_\infty\left(K \;\middle|\; \Lambda_0, \overline{E}_1, \ldots, \overline{E}_{j-1}, \overline{C}\right) \geq \mathrm{H}_\infty\left(K \;\middle|\; \overline{E}_1, \ldots, \overline{E}_{j-1}, \overline{C}\right) - \lambda 2^{rr}$$

and

$$\mathrm{H}_\infty\left(S_i \;\middle|\; \Lambda_i, \overline{E}_1, \ldots, \overline{E}_{j-1}, \overline{C}\right) \geq \mathrm{H}_\infty\left(S_i \;\middle|\; \overline{E}_1, \ldots, \overline{E}_{j-1}, \overline{C}\right) - \lambda 2^{rr}. \tag{13}$$

It now remains to bound the min-entropies of K and S_i conditioned on $E_1, \ldots, E_{j-1}$ and C not occurring. For each event E_j, conditioning on it not occurring excludes one value for the variable K and 2^{rr} possible values for the variable S_i. On the other hand, conditioning on $\overline{C}$ excludes (at most) a further $(l+1)(q_{\mathtt{F}} + q_{\mathtt{LF}})$ from the possible values that K and S_i may take. Then by constraining the adversary's queries such that $q_\rho + (l+1)(q_{\mathtt{F}} + q_{\mathtt{LF}}) \leq 2^{k-1}$ and $(2^{rr})q_\rho + (l+1)(q_{\mathtt{F}} + q_{\mathtt{LF}}) \leq 2^{n-1}$, we obtain, for all possible j, the following bounds

$$\mathrm{H}_\infty\left(K \;\middle|\; \overline{E}_1, \ldots, \overline{E}_{j-1}, \overline{C}\right) \geq k - 1$$

and

$$\mathrm{H}_\infty\left(S_i \;\middle|\; \overline{E}_1, \ldots, \overline{E}_{j-1}, \overline{C}\right) \geq n - 1. \tag{14}$$

Combining (12), (13), and (14) and applying the bound $|\mathcal{V}_{\mathtt{F}}| \leq l q_{\mathtt{F}}$ yields

$$\Pr[E_j \,|\, \overline{E}_1, \ldots, \overline{E}_{j-1}, \overline{C}] \leq \frac{1}{2^{k-\lambda 2^{rr}-1}} + \frac{l q_{\mathtt{F}}}{2^{n-\lambda 2^{rr}-1}}.$$

We then substitute the above in (11) to obtain

$$\Pr[\mathsf{Bad}_2^1 \,|\, \overline{C}] \leq \frac{q_\rho}{2^{k-\lambda 2^{rr}-1}} + \frac{l q_{\mathtt{F}} q_\rho}{2^{n-\lambda 2^{rr}-1}}. \tag{15}$$

Finally, we bound the last term of inequality (9). The flag Bad_2^2 is set if $\mathcal{A}$ makes some query to $\mathtt{F}$ such that the corresponding Y_l value was already contained in $\mathsf{inset}(p)$. Now if it were the case that Y_l was already in $\mathcal{V}_{\mathtt{F}} \cup \mathcal{V}_{\mathtt{LF}}$ when Bad_2^2 occurs it would mean that event C has also occurred. Thus, conditioned on C not occurring, the only way for Bad_2^2 to be set is if Y_l was already queried to ρ by $\mathcal{A}$. Let X^* represent $\mathcal{A}$'s query to $\mathtt{F}$ that sets Bad_2^2, resulting in the corresponding values $Y_i^* = S_i^* \oplus (X_i^* \,\|\, 0^{n-rr})$ for $1 \leq i \leq l$, such that Y_l^* was already queried to ρ. Then, conditioned on E not occurring, the variable S_l^* cannot be sampled during a query to ρ. That is, $p[Y_{l-1}^*]$ must be set during some query to $\mathtt{F}$ or $\mathtt{LF}$ after that Y_l^* was queried to ρ. Now let F be the event that $\mathcal{A}$

makes a query to $\mathtt{F}$ or $\mathtt{LF}$ resulting in S_l^* such that for some $A \in \{0,1\}^{rr}$ the value $S_l^* \oplus (A \,\|\, 0^{n-rr})$ was already queried to ρ. It then follows that

$$\Pr[\mathsf{Bad}_2^2 \,|\, \overline{C}, \overline{E}] \leq \Pr[F \,|\, \overline{C}, \overline{E}]\,. \tag{16}$$

We now bound $\Pr[F \,|\, \overline{C}, \overline{E}]$. Let F_j denote the event that F occurs at the j^{th} query that $\mathcal{A}$ makes to $\mathtt{F}$ or $\mathtt{LF}$. Then, by the union bound we have that

$$\Pr[F \,|\, \overline{C}] \leq \Pr[F_1 \,|\, \overline{C}, \overline{E}] + \Pr[F_2 \,|\, \overline{C}, \overline{E}] + \ldots + \Pr[F_{q_{\mathtt{F}}+q_{\mathtt{LF}}} \,|\, \overline{C}, \overline{E}]\,.$$

Then for each query there are at most $(2^{rr})q_\rho$ values that S_l^* can take to set F out of $2^n - (l+1)(q_{\mathtt{F}} + q_{\mathtt{LF}}) - (2^{rr})q_\rho$ (due to conditioning on $\overline{C}$ and $\overline{E}$). Thus

$$\begin{aligned}\Pr[F \,|\, \overline{C}] &\leq \sum_{j=1}^{q_{\mathtt{F}}+q_{\mathtt{LF}}} \frac{(2^{rr})q_\rho}{2^n - (l+1)(q_{\mathtt{F}} + q_{\mathtt{LF}}) - (2^{rr})q_\rho} \\ &\leq \frac{2^{rr}(q_{\mathtt{F}} + q_{\mathtt{LF}})q_\rho}{2^{n-1}} \\ &= \frac{(q_{\mathtt{F}} + q_{\mathtt{LF}})q_\rho}{2^{n-rr-1}}\,. \end{aligned} \tag{17}$$

Then, combining inequalities (9), (10), (15), and (17) we obtain

$$\Pr[\mathcal{A}^{\mathsf{G}_2} \Rightarrow 1] - \Pr[\mathcal{A}^{\boxed{\mathsf{G}_2}} \Rightarrow 1] \leq \frac{q_T(q_T+2)}{2^{n-rr}} + \frac{q_\rho}{2^{k-\lambda 2^{rr}-1}} + \frac{l\,q_{\mathtt{F}}\,q_\rho}{2^{n-\lambda 2^{rr}-1}} + \frac{(q_{\mathtt{F}} + q_{\mathtt{LF}})q_\rho}{2^{n-rr-1}}\,, \tag{18}$$

where $q_T = (l+1)(q_{\mathtt{LF}} + q_{\mathtt{F}}) + q_\rho$.

$\boxed{\mathsf{G}_3}$ This game is described in Fig. 22. Here we remove the three flags Bad_2^1, Bad_2^2, and Bad_2^3, we introduce flag Bad_3, and change how the last round of $\mathtt{F}$ is evaluated. Notably, in tandem with $p^*[\,]$ we maintain an additional array $f[\,]$, where the latter is indexed by strings of size l instead of n. Then in every evaluation of $\mathtt{F}$ we sample a random string, store it in $f[X]$, and then copy this value to $p^*[Y_l]$. However if $p^*[Y_l]$ was already defined, then Bad_3 is set and the random string in $f[X]$ is replaced with $p^*[Y_l]$. This ensures that no entry in $p^*[\,]$ is ever overwritten. Thus the array $p^*[\,]$ is sampled in an equivalent manner as in G_2. Moreover, the functionality of G_2 is not dependent on the flags Bad_2^1, Bad_2^2, and Bad_2^3 and their removal is merely a cosmetic alteration. Hence $\boxed{\mathsf{G}_3}$ is identical to G_2 and it follows that

$$\Pr[\mathcal{A}^{\boxed{\mathsf{G}_3}} \Rightarrow 1] = \Pr[\mathcal{A}^{\mathsf{G}_2} \Rightarrow 1]\,. \tag{19}$$

G_3 This game is the game described in Fig. 22 without the boxed code. By removing the boxed code we now have that $\mathtt{F}(X) = f[X]$, and therefore $\mathtt{F}(X)$ behaves as a random function. Therefore

$$\Pr[\mathcal{A}^{\mathsf{LPRF}} \Rightarrow 1 \,|\, b = 0] = \Pr[\mathcal{A}^{\mathsf{G}_3} \Rightarrow 1]\,. \tag{20}$$

Furthermore, G_3 and $\boxed{\mathsf{G}_3}$ are identical until Bad_3 occurs. Now, if Bad_3 is set then $\mathcal{A}$ must have queried X and X' (where $X \neq X'$ since $\mathcal{A}$ is assumed to not repeat queries to $\mathtt{F}$) such that they result in the same value for Y_l. As before we can view this as a collision on $\mathcal{H}^{K\|IV}$ with l rounds. Thus, applying the fundamental lemma of game-playing and using once again inequality (49) from Appendix C.4, we have that

$$\begin{aligned}\Pr[\mathcal{A}^{\mathsf{G}_3} \Rightarrow 1] - \Pr[\mathcal{A}^{\boxed{\mathsf{G}_3}} \Rightarrow 1] &\leq \Pr[\mathsf{Bad}_3] \\ &\leq \frac{q'_T(q'_T+2)}{2^{n-rr}}\,. \end{aligned} \tag{21}$$

where $q'_T = l(q_{\mathtt{LF}} + q_{\mathtt{F}}) + q_\rho$.

Combining inequalities (7), (8), (18), (19), (21), and (20) yields the desired result. □

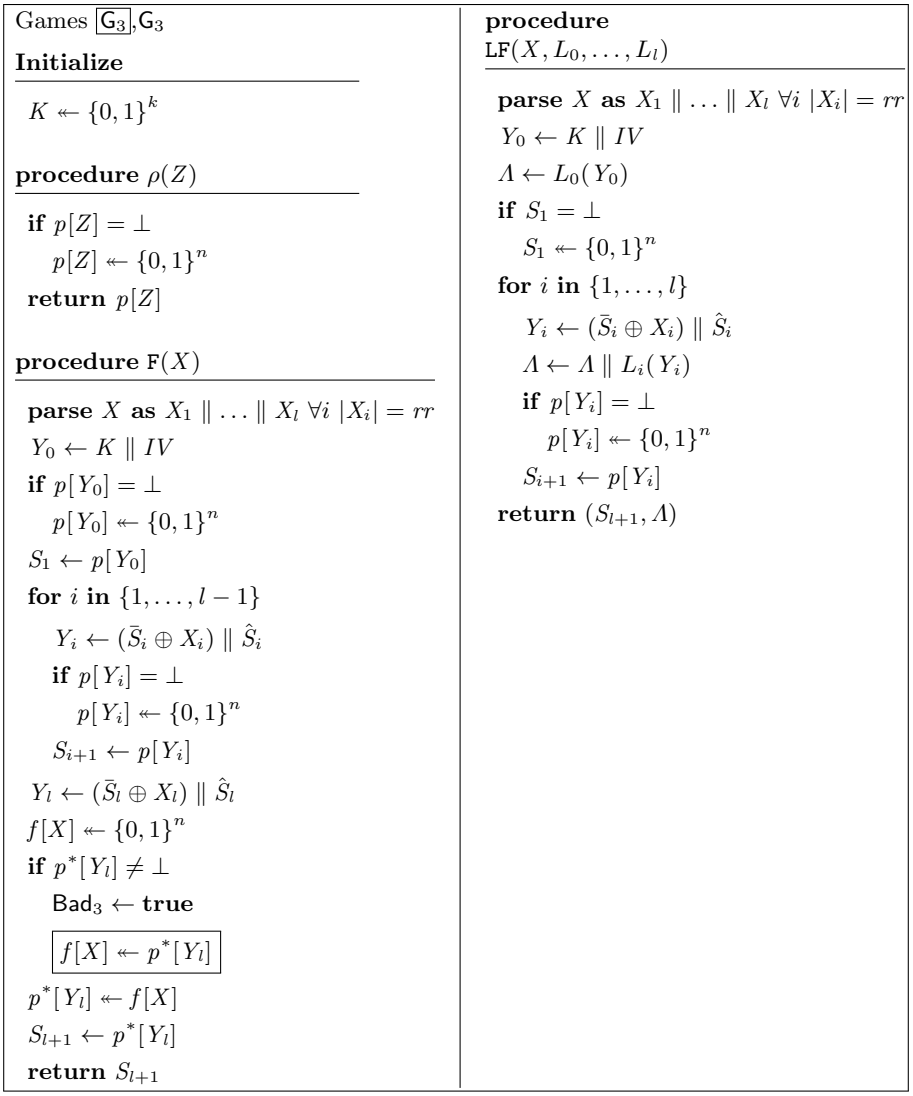

Fig. 22: Games $\boxed{\mathsf{G}_3}$ and G_3 used in the proof of Theorem 7. Game $\boxed{\mathsf{G}_3}$ includes the boxed code whereas G_3 does not.

C.2 Proof of Theorem 8

We assume that SLFUNC takes inputs of fixed length $(l \cdot rr)$ and returns outputs of length t. Now consider the game described in Fig. 23. It is the LUF game instantiated with SLFUNC where the adversary is also given oracle access to the random transformation ρ which SLFUNC depends on. This random transformation is sampled lazily across all oracles and the corresponding values are stored in a global array $p[\,]$. All entries of $p[\,]$ are initialised to $\perp$. Thus,

$$\Pr[\mathsf{LUF}^{\mathcal{A}} \Rightarrow \mathsf{true}] = \Pr[\mathsf{G}^{\mathcal{A}} \Rightarrow \mathsf{true}]\,. \tag{22}$$

At any point in time, let $\mathcal{V}_{\mathtt{Lkg}}$ be the set union of intermediate values $\{Y_1, \ldots, Y_l\}$ evaluated in Lkg over all queries made by $\mathcal{A}$ up to that point. Similarly, let $\mathcal{U}_{\mathtt{Lkg}}$ be the set of values S_{l+1} evaluated in Lkg over all queries made by $\mathcal{A}$. Further let W be the event that $\mathcal{A}$ wins and B be the event that $\mathcal{A}$ makes a query to ρ that is contained in $\mathcal{V}_{\mathtt{Lkg}} \cup \{Y_0\}$. Then for any event C it follows that

$$\begin{aligned}\Pr[\mathsf{G}^{\mathcal{A}} \Rightarrow \mathsf{true}] &\leq \Pr[W \cup B \cup C] \\ &\leq \Pr[C] + \Pr[B \,|\, \overline{C}] + \Pr[W \,|\, \overline{C}, \overline{B}]\,.\end{aligned} \tag{23}$$

Recall that SLFUNC(K, X) can be viewed as the sponge-based hash function from Section 5.4 evaluated on input $0 \,\|\, X$ with an initial state of $K \,\|\, IV$ (instead of 0^n) and capacity $n - rr$. Let $\mathcal{H}^{K\|IV}$ denote the variable-input-length hash function just described and C be the event that one of the following conditions is satisfied.

(i) $\mathcal{A}$ makes queries X and X' across F, Lkg, and Guess such that $\mathcal{H}^{K\|IV}(0 \,\|\, X) = \mathcal{H}^{K\|IV}(0 \,\|\, X')$ and $X \neq X'$.
(ii) $\mathcal{A}$ makes queries X and X' across F, Lkg, and Guess such that $\mathcal{H}^{K\|IV}(0 \,\|\, X) = \mathcal{H}^{K\|IV}(0 \,\|\, X'')$ for some X'' that is a prefix of X' and $|X''| < |X'|$.
(iii) $\mathcal{A}$ makes a query X to F, Lkg, or Guess such that $\mathcal{H}^{K\|IV}(0 \,\|\, X') = K \,\|\, IV$ for some X' that is a prefix of X.
(iv) $\mathcal{A}$ makes queries X and X' across F, Lkg, and Guess such that $\mathcal{H}^{K\|IV}(0 \,\|\, X'') = \mathcal{H}^{K\|IV}(0 \,\|\, X''')$, where X'' is a prefix of X, X''' is a prefix of X', $X'' \neq \varepsilon \neq X'''$, and $X'' \neq X'''$.

Now for any adversary $\mathcal{A}$ that causes C, we can construct an adversary $\mathcal{B}$ that finds a collision in $\mathcal{H}^{K\|IV}$. Specifically, $\mathcal{B}$ knows K and uses this together with its access to ρ to simulate game G_1 to $\mathcal{A}$ while constantly checking if event C has occurred. Note that any of the sub-cases in C will directly enable $\mathcal{B}$ to produce a collision on $\mathcal{H}^{K\|IV}$. We can therefore use the security bound derived in Section 5.4 for $\mathcal{H}$ since it applies to any initial state that $\mathcal{H}$ is assigned, as long as it is fixed at the start of the game. Thus, applying inequality (49) from Appendix C.4 and setting $w = n$, $c' = n - rr$, and $q_T = (l+1)(q_{\mathtt{F}} + q_{\mathtt{Lkg}} + q_{\mathtt{Guess}}) + q_\rho$, yields

$$\Pr[C] \leq \frac{q_T(q_T + 2)}{2^{n-rr}}\,. \tag{24}$$

We now bound $\Pr[B \,|\, \overline{C}]$. Let B_j be the event that B occurs on the j^{th} query that $\mathcal{A}$ makes to ρ. Then we have that

$$\begin{aligned}\Pr[B \,|\, \overline{C}] &\leq \Pr[B_1 \cup B_2 \cup \ldots \cup B_{q_\rho} \,|\, \overline{C}] \\ &= \Pr[B_1 \,|\, \overline{C}] + \Pr[B_2 \,|\, \overline{B}_1, \overline{C}] + \ldots \\ &\qquad + \Pr[B_{q_\rho} \,|\, \overline{B}_1, \ldots, \overline{B}_{q_\rho - 1}, \overline{C}]\,.\end{aligned} \tag{25}$$

Now as long as C does not occur no two queries will result in values Y_i and $Y_{i'}$ such that $Y_i = Y_{i'}$ and $i \neq i'$. Thus no state variable $p[Y_i]$ occurs in more than one position (across F, Lkg, and Guess), meaning that no state variable is subject to more than one leakage function and that no internal state variable is ever exposed as the output of F. Moreover, as long as B does not occur, the outputs of ρ will never expose any of the state variables contained in $\mathcal{V}_{\mathtt{Lkg}} \cup \mathcal{U}_{\mathtt{Lkg}} \cup \{Y_0\}$. Then, for $i \geq 1$, guessing a value $Y_i \in \mathcal{V}_{\mathtt{Lkg}}$ is tantamount to guessing the corresponding S_i, since $Y_i = S_i + (X_i \,\|\, 0^{n-rr})$ and X_i is known to the adversary. Then, by the union bound and the above observations, for all $j \in \{1, \ldots, q_\rho\}$, we have that

Game G

Initialize

$\mathsf{win} \leftarrow \mathsf{false}$; $K \twoheadleftarrow \mathcal{K}$
return

procedure $\rho(Z)$

if $p[Z] = \perp$
  $p[Z] \twoheadleftarrow \{0,1\}^n$
return $p[Z]$

procedure $\mathtt{Guess}(X, y')$

parse X **as** $X_1 \parallel \ldots \parallel X_l$ $\forall i$ $|X_i| = rr$
$Y_0 \leftarrow K \parallel IV$
if $p[Y_0] = \perp$
  $p[Y_0] \twoheadleftarrow \{0,1\}^n$
$S_1 \leftarrow p[Y_0]$
for i **in** $\{1, \ldots, l\}$
  $Y_i \leftarrow (\bar{S}_i \oplus X_i) \parallel \hat{S}_i$
  if $p[Y_i] = \perp$
    $p[Y_i] \twoheadleftarrow \{0,1\}^n$
  $S_{i+1} \leftarrow p[Y_i]$
$y \leftarrow \lfloor S_{l+1} \rfloor_t$
if $X \notin \mathcal{S} \wedge y = y'$
  $\mathsf{win} \leftarrow \mathsf{true}$
return $(y = y')$

procedure Finalize

return win

procedure $\mathtt{F}(X)$

$\mathcal{S} \leftarrow_{\cup} X$
parse X **as** $X_1 \parallel \ldots \parallel X_l$ $\forall i$ $|X_i| = rr$
$Y_0 \leftarrow K \parallel IV$
if $p[Y_0] = \perp$
  $p[Y_0] \twoheadleftarrow \{0,1\}^n$
$S_1 \leftarrow p[Y_0]$
for i **in** $\{1, \ldots, l\}$
  $Y_i \leftarrow (\bar{S}_i \oplus X_i) \parallel \hat{S}_i$
  if $p[Y_i] = \perp$
    $p[Y_i] \twoheadleftarrow \{0,1\}^n$
  $S_{i+1} \leftarrow p[Y_i]$
return $\lfloor S_{l+1} \rfloor_t$

procedure $\mathtt{Lkg}(X, L_0, \ldots, L_{l+1})$

parse X **as** $X_1 \parallel \ldots \parallel X_l$ $\forall i$ $|X_i| = rr$
$Y_0 \leftarrow K \parallel IV$
$\Lambda \leftarrow L_0(Y_0)$
if $p[Y_0] = \perp$
  $p[Y_0] \twoheadleftarrow \{0,1\}^n$
$S_1 \leftarrow p[Y_0]$
for i **in** $\{1, \ldots, l\}$
  $Y_i \leftarrow (\bar{S}_i \oplus X_i) \parallel \hat{S}_i$
  $\Lambda \leftarrow \Lambda \parallel L_i(Y_i)$
  if $p[Y_i] = \perp$
    $p[Y_i] \twoheadleftarrow \{0,1\}^n$
  $S_{i+1} \leftarrow p[Y_i]$
$\Lambda \leftarrow \Lambda \parallel L_{l+1}(S_{l+1})$
return (Λ)

Fig. 23: Game G used in the proof of Theorem 8.

$$\begin{aligned}
\Pr[B_j \,|\, \overline{B}_1, \ldots, \overline{B}_{j-1}, \overline{C}] \leq & \sum_{Y_i \in \mathcal{V}_{\mathtt{Lkg}}} 2^{-\mathrm{H}_\infty\left(Y_i \,|\, \Lambda, \overline{B}_1, \ldots, \overline{B}_{j-1}, \overline{C}\right)} \\
+ & \quad 2^{-\mathrm{H}_\infty\left(K \,|\, \Lambda, \overline{B}_1, \ldots, \overline{B}_{j-1}, \overline{C}\right)} \\
= & \sum_{Y_i \in \mathcal{V}_{\mathtt{Lkg}}} 2^{-\mathrm{H}_\infty\left(S_i \,|\, \Lambda, \overline{B}_1, \ldots, \overline{B}_{j-1}, \overline{C}\right)} \\
+ & \quad 2^{-\mathrm{H}_\infty\left(K \,|\, \Lambda, \overline{B}_1, \ldots, \overline{B}_{j-1}, \overline{C}\right)} . \qquad (26)
\end{aligned}$$

Now with respect to the key the adversary only learns $L_0(K \parallel IV)$ from the leakage. However, for a given S_i the adversary may obtain (at most) the leakage $L_i(S_i \oplus X_i \parallel 0^{n-rr})$ for each possible value of X_i. If we consider the aggregate leakage for a given S_i over all possible values of X_i, i.e. $L_i(S_i \oplus 0^{rr} \parallel 0^{n-rr}) \parallel \ldots \parallel L_i(S_i \oplus 1^{rr} \parallel 0^{n-rr})$, the support of this aggregate random variable is bounded by $2^{\lambda 2^{rr}}$. Then by applying Lemma 1 we have that

$$\mathrm{H}_\infty\left(K \,\middle|\, \Lambda_0, \overline{B}_1, \ldots, \overline{B}_{j-1}, \overline{C}\right) \geq \mathrm{H}_\infty\left(K \,\middle|\, \overline{B}_1, \ldots, \overline{B}_{j-1}, \overline{C}\right) - \lambda 2^{rr}$$

and

$$\mathrm{H}_\infty\left(S_i \,\middle|\, \Lambda_i, \overline{B}_1, \ldots, \overline{B}_{j-1}, \overline{C}\right) \geq \mathrm{H}_\infty\left(S_i \,\middle|\, \overline{B}_1, \ldots, \overline{B}_{j-1}, \overline{C}\right) - \lambda 2^{rr} . \qquad (27)$$

It now remains to bound the min-entropies of K and S_i conditioned on $B_1, \ldots, B_{j-1}$ and C not occurring. For each event B_j, conditioning on it not occurring excludes one value for the variable K and 2^{rr} possible values for the variable S_i. This is because for any query Z that the adversary makes to ρ, the values $(Z \oplus 0^{rr} \,\|\, 0^{n-rr}), \ldots, (Z \oplus 1^{rr} \,\|\, 0^{n-rr})$ may all be contained $\mathcal{V}_{\mathtt{Lkg}}$. On the other hand, conditioning on $\overline{C}$ excludes (at most) a further $(l+1)(q_{\mathtt{F}} + q_{\mathtt{Lkg}} + q_{\mathtt{Guess}})$ from the possible values that K and S_i may take. Thus if we constrain the adversary's queries such that $q_\rho + (l+1)(q_{\mathtt{F}} + q_{\mathtt{Lkg}} + q_{\mathtt{Guess}}) \leq 2^{k-1}$ and $(2^{rr})q_\rho + (l+1)(q_{\mathtt{F}} + q_{\mathtt{Lkg}} + q_{\mathtt{Guess}}) \leq 2^{n-1}$, we obtain, for all possible j, the following bounds

$$\mathrm{H}_\infty\left(K \,\middle|\, \overline{E}_1, \ldots, \overline{E}_{j-1}, \overline{C}\right) \geq k - 1$$

and

$$\mathrm{H}_\infty\left(S_i \,\middle|\, \overline{E}_1, \ldots, \overline{E}_{j-1}, \overline{C}\right) \geq n - 1\,. \tag{28}$$

Combining (26), (27), and (28) and applying the bound $|\mathcal{V}_{\mathtt{Lkg}}| \leq l q_{\mathtt{Lkg}}$ yields

$$\Pr[B_j \,|\, \overline{B}_1, \ldots, \overline{B}_{j-1}, \overline{C}] \leq \frac{1}{2^{k-\lambda 2^{rr}-1}} + \frac{l q_{\mathtt{Lkg}}}{2^{n-\lambda 2^{rr}-1}}\,.$$

We then substitute the above in (25) to obtain

$$\Pr[B \,|\, \overline{C}] \leq \frac{q_\rho}{2^{k-\lambda 2^{rr}-1}} + \frac{l q_{\mathtt{Lkg}} q_\rho}{2^{n-\lambda 2^{rr}-1}}. \tag{29}$$

We now conclude the proof by bounding $\Pr[W \,|\, \overline{C}, \overline{B}]$. Let W_j be the event that W occurs on the j^{th} query that $\mathcal{A}$ makes to $\mathtt{Guess}$. Then we have that

$$\begin{aligned}\Pr[W \,|\, \overline{C}, \overline{B}] &\leq \Pr[W_1 \cup W_2 \cup \ldots \cup W_{q_\rho} \,|\, \overline{C}, \overline{B}] \\ &= \Pr[W_1 \,|\, \overline{C}, \overline{B}] + \Pr[W_2 \,|\, \overline{W}_1, \overline{C}, \overline{B}] + \ldots \\ &\qquad + \Pr[W_{q_{\mathtt{Guess}}} \,|\, \overline{W}_1, \ldots, \overline{W}_{q_{\mathtt{Guess}}-1}, \overline{C}, \overline{B}]\,. \end{aligned} \tag{30}$$

Now conditioning on neither of events B and C occurring, it must be that if W occurs the value of S_{l+1} corresponding to that query must be either new (freshly sampled) or contained in $\mathcal{U}_{\mathtt{Lkg}}$. In either case, for all $j \in \{1, \ldots, q_{\mathtt{Guess}}\}$, we have that

$$\Pr[W_j \,|\, \overline{W}_1, \ldots, \overline{W}_{j-1}, \overline{C}, \overline{B}] \leq 2^{-\mathrm{H}_\infty\left(\lfloor S_{l+1} \rfloor_t \,\middle|\, \Lambda, \overline{W}_1, \ldots, \overline{W}_{j-1}, \overline{C}, \overline{B}\right)}\,, \tag{31}$$

where the case where S_{l+1} is freshly sampled corresponds to $\Lambda = \varepsilon$. Thus since Λ is at most $L_{l+1}(S_{l+1})$, by Lemma 1 we have that

$$\mathrm{H}_\infty\left(S_{l+1} \,\middle|\, \Lambda, \overline{W}_1, \ldots, \overline{W}_{j-1}, \overline{C}, \overline{B}\right) \geq \mathrm{H}_\infty\left(S_{l+1} \,\middle|\, \overline{W}_1, \ldots, \overline{W}_{j-1}, \overline{C}, \overline{B}\right) - \lambda\,. \tag{32}$$

Once again conditioning on $\overline{C}$ excludes (at most) $(l+1)(q_{\mathtt{F}} + q_{\mathtt{Lkg}} + q_{\mathtt{Guess}})$ from the possible values that S_{l+1} may take, whereas conditioning on $\overline{B}$ does not affect S_{l+1}. Moreover, for each event W_j, conditioning on it not occurring excludes one value from the pool of values that S_{l+1} may take. To cater for this we require that the adversary's queries satisfy $q_{\mathtt{Guess}} + (l+1)(q_{\mathtt{F}} + q_{\mathtt{Lkg}} + q_{\mathtt{Guess}}) \leq 2^{n-1}$. Then from the above and (32) we have that

$$\mathrm{H}_\infty\left(S_{l+1} \,\middle|\, \Lambda, \overline{W}_1, \ldots, \overline{W}_{j-1}, \overline{C}, \overline{B}\right) \geq n - 1 - \lambda\,. \tag{33}$$

To cater for the fact that S_{l+1} is truncated to the leftmost t bits, we apply Lemma 1 once more. The probability of guessing $\lfloor S_{l+1} \rfloor_t$ can only increase when given the rightmost $n - t$ bits of S_{l+1}. Thus

$$\mathrm{H}_\infty\left(\lfloor S_{l+1} \rfloor_t \,\middle|\, \Lambda, \overline{W}_1, \ldots, \overline{W}_{j-1}, \overline{C}, \overline{B}\right) \geq t - \lambda - 1\,. \tag{34}$$

Combining inequalities (30), (31), and (34) yields

$$\Pr[W \,|\, \overline{C}, \overline{B}] \leq \frac{q_{\mathtt{Guess}}}{2^{t-\lambda-1}}\,. \tag{35}$$

The theorem then follows by combining (23), (24), (29), and (35). □

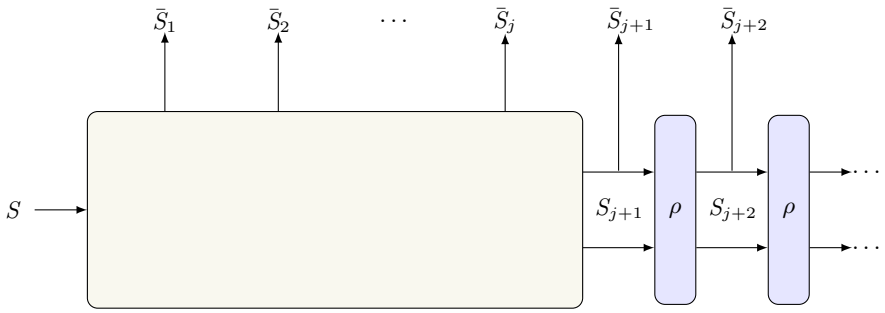

Fig. 24: A graphical illustration of the hybrid game H_j used to prove Theorem 9. The large box represents a random function with input S and output $(\bar{S}_2, \ldots \bar{S}_j, S_{j+1})$. The additional output $\bar{S}_1$ is simply copied from the input, i.e $\bar{S}_1 = \bar{S}$.

C.3 Proof of Theorem 9

The proof follows through a standard hybrid argument, where we gradually replace the SPRG construction with a random function that takes as its input the seed S. In each game hop we remove one call to the random transformation and extend the output of the random function. This is illustrated in Fig. 24.

Let L_{max} be an upper bound on the output bit-length specified by the adversary in its queries to $\mathtt{G}$. Then each evaluation will require at most $v-1$ call to the random transformation, where $v = \lceil \frac{L_{max}}{r} \rceil$. Accordingly we specify the hybrid games $\mathsf{H}_0, \ldots, \mathsf{H}_{v-1}$, as shown in Fig. 25. Following the hybrid argument, a further game hop to game G_0, displayed in Fig. 26, is required to complete the proof.

In game H_j (cf. Fig. 25), the $\mathtt{G}$ oracle samples the states $S_1, \ldots, S_{j+1}$ ideally and independent of the transformation ρ, while the remaining states $S_{j+2}, \ldots, S_v$ are sampled by querying ρ. The output of the $\mathtt{G}$ oracle is the concatenation of the outer states truncated to L bits. The boxed version, that is, $\boxed{\mathsf{H}_j}$, ensures that the state S_{j+1} is also evaluated through ρ, and thus $\mathsf{H}_j = \boxed{\mathsf{H}_{j+1}}$. Note that H_0 is equivalent to the PRG game instantiated with SPRG with the challenge bit b fixed to 1. Thus,

$$\Pr[\mathcal{A}^{\mathsf{PRG}} \Rightarrow 1 \,|\, b = 1] = \Pr[\mathcal{A}^{\mathsf{H}_0} \Rightarrow 1]\,. \tag{36}$$

Moreover, we have that

$$\Pr[\mathcal{A}^{\mathsf{H}_0} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathsf{H}_{v-1}} \Rightarrow 1] = \sum_{j=1}^{v-1} \Pr[\mathcal{A}^{\mathsf{H}_{j-1}} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathsf{H}_j} \Rightarrow 1]\,,$$

and since $\mathsf{H}_{j-1} = \boxed{\mathsf{H}_j}$, this reduces to

$$= \sum_{j=1}^{v-1} \Pr[\mathcal{A}^{\boxed{\mathsf{H}_j}} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathsf{H}_j} \Rightarrow 1]\,.$$

Let Bad_j^1, Bad_j^2, and Bad_j^3 each denote the event that the corresponding flag in game $\boxed{\mathsf{H}_j}$ is set. Further note that games $\boxed{\mathsf{H}_j}$ and H_j are identical until one of these bad events occurs. Then, applying the fundamental lemma of game playing and the union bound yields

$$\leq \sum_{j=1}^{v-1} \Pr[\mathsf{Bad}_j^1] + \Pr[\mathsf{Bad}_j^2] + \Pr[\mathsf{Bad}_j^3]\,. \tag{37}$$

The flag Bad_j^1 is set if the adversary makes a query S to ρ, such that $S = S_j[S^*]$ for some $S^* \in \mathcal{D}$. Now, for any random variable $S_j[S^*]$, only the outer part $\bar{S}_j[S^*]$ is known to the adversary (through the $\mathtt{G}$ oracle) and the inner c bits remain hidden. Moreover, at any point in time, there are only $|\mathcal{D}|$ such variables and $|\mathcal{D}| \leq q_{\mathtt{G}}$. Hence, by the union bound

$$\Pr[\mathsf{Bad}_j^1] \leq \sum_{i=0}^{q_\rho - 1} \frac{|\mathcal{D}|}{2^c - i} \leq \sum_{i=0}^{q_\rho - 1} \frac{q_{\mathtt{G}}}{2^c - i} \leq \frac{q_{\mathtt{G}}\, q_\rho}{2^c - q_\rho}\,. \tag{38}$$

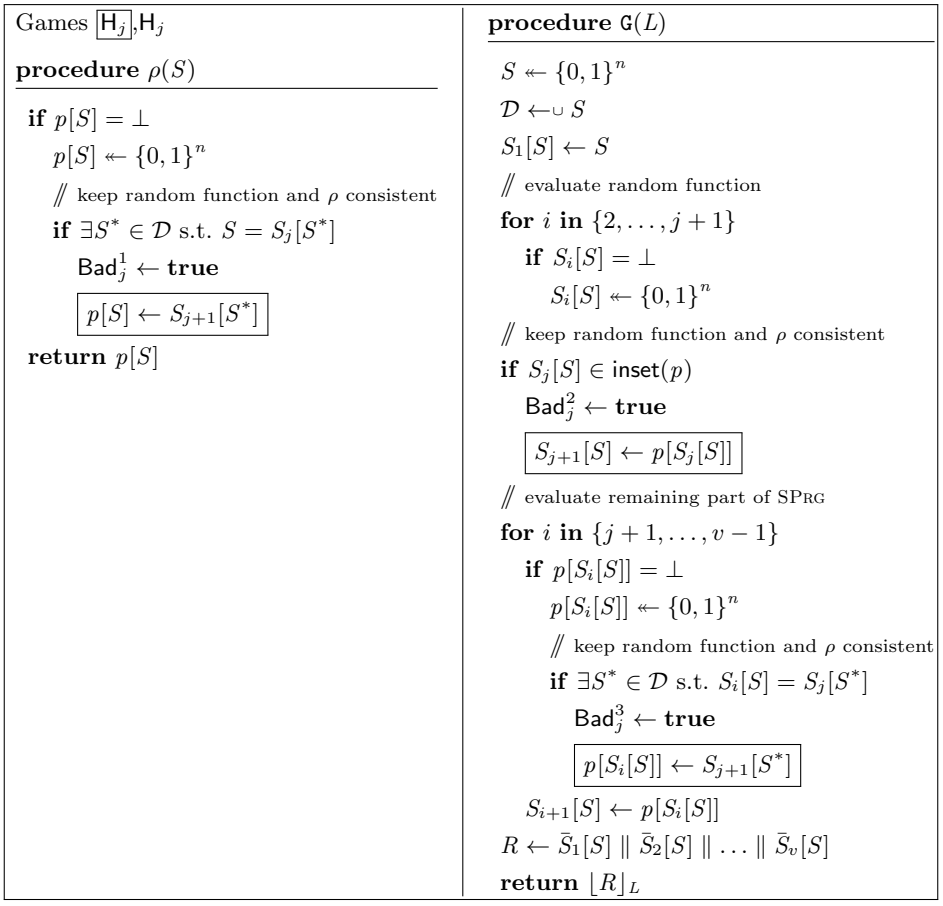

Fig. 25: The hybrid games used in the proof of Theorem 9. Game $\boxed{\mathsf{H}_j}$ contains the boxed code whereas H_j does not.

The flag Bad_j^2 is set whenever the $\mathtt{G}$ oracle samples a value S_j contained in $\mathsf{inset}(p)$. Since the $\mathtt{G}$ oracle samples the values S_j uniformly at random from $\{0,1\}^n$, it follows that

$$\Pr[\mathsf{Bad}_j^2] \leq \sum_{i=1}^{q_\mathtt{G}} \frac{|\mathsf{inset}(p)|}{2^n} .$$

At any point in time, $|\mathsf{inset}(p)| \leq q_\rho + q_\mathtt{G}(v-j-1)$, since each query to ρ and $\mathtt{G}$ adds at most 1 and $(v-j-1)$, new values to $\mathsf{inset}(p)$ respectively. This leads to

$$\leq \sum_{i=1}^{q_\mathtt{G}} \frac{q_\rho + q_\mathtt{G}(v-j-1)}{2^n} \leq \frac{q_\rho q_\mathtt{G} + q_\mathtt{G}^2(v-j-1)}{2^n} . \tag{39}$$

The flag Bad_j^3 is set if, for some $i \in \{j+1, \ldots, v-1\}$ and some $S^* \in \mathcal{D}$, the oracle $\mathtt{G}$ samples a value $S_i[S]$ equal to $S_i[S^*]$. The probability of this bad event depends solely on the number of queries that $\mathcal{A}$ makes to its oracle $\mathtt{G}$. In each query at most, $(v-j-1)$ states are sampled which can set this flag, hence

$$\Pr[\mathsf{Bad}_j^3] \leq \sum_{i=1}^{q_\mathtt{G}} \frac{(v-j-1)|\mathcal{D}|}{2^n} .$$

Since at any point in time $|\mathcal{D}| \leq q_\mathtt{G}$, this reduces to

$$\leq \sum_{i=1}^{q_\mathtt{G}} (v-j-1)\frac{q_\mathtt{G}}{2^n} \leq \frac{(v-j-1)\,q_\mathtt{G}^2}{2^n} . \tag{40}$$

Game G_0	**procedure** $\mathtt{G}(L)$
procedure $\rho(S)$	$S \twoheadleftarrow \{0,1\}^n$
if $p[S] = \bot$	$R \twoheadleftarrow \{0,1\}^{vr}$
$p[S] \twoheadleftarrow \{0,1\}^n$	**if** $S \in \mathcal{D}$
return $p[S]$	$\mathsf{Bad} \leftarrow \mathbf{true}$
	$\boxed{R \leftarrow \bar{S}_1[S] \parallel \ldots \parallel \bar{S}_v[S]}$
	$\mathcal{D} \xleftarrow{\cup} S$
	$(\bar{S}_1[S], \ldots, \bar{S}_v[S]) \leftarrow R$
	return $\lfloor R \rfloor_L$

Fig. 26: Games G_0 and $\boxed{\mathsf{G}_0}$ used in the proof of Theorem 9. Game $\boxed{\mathsf{G}_0}$ contains the boxed code, G_0 does not.

Combining (37), (38), (39), and (40) yields

$$\begin{aligned}\Pr[\mathcal{A}^{\mathsf{H}_0} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathsf{H}_{v-1}} \Rightarrow 1] &\leq \sum_{j=1}^{v-1} \Pr[\mathsf{Bad}_j^1] + \Pr[\mathsf{Bad}_j^2] + \Pr[\mathsf{Bad}_j^3] \\ &\leq \sum_{j=1}^{v-1} \frac{q_{\mathtt{G}} q_\rho}{2^c - q_\rho} + \frac{q_\rho q_{\mathtt{G}} + q_{\mathtt{G}}^2 (v-j-1)}{2^n} + \frac{(v-j-1) q_{\mathtt{G}}^2}{2^n} \\ &\leq \frac{v\, q_{\mathtt{G}} q_\rho}{2^c - q_\rho} + \frac{v\, q_\rho q_{\mathtt{G}} + v^2 q_{\mathtt{G}}^2}{2^n} \end{aligned} \tag{41}$$

To conclude the proof we require one more game hop. Consider the game $\boxed{\mathsf{G}_0}$ displayed in Fig. 26. In this game, the $\mathtt{G}$ oracle first samples a value S, followed by sampling a random output R, independent of ρ. The boxed code ensures that the output R is identical if a value for S is sampled twice. This game is equivalent to H_{v-1} which yields

$$\Pr[\mathcal{A}^{\mathsf{H}_{v-1}} \Rightarrow 1] = \Pr[\mathcal{A}^{\boxed{\mathsf{G}_0}} \Rightarrow 1]\,. \tag{42}$$

Now, games $\boxed{\mathsf{G}_0}$ and G_0 (also displayed in Fig. 26) are identical until Bad occurs. The flag is set to true if a value S is sampled that is already in the set $\mathcal{D}$. At any point in time $|\mathcal{D}| \leq q_{\mathtt{G}}$, and S is sampled by the $\mathtt{G}$ independently of $\mathcal{A}$, thus

$$\Pr[\mathcal{A}^{\mathsf{G}_0} \Rightarrow 1] - \Pr[\mathcal{A}^{\boxed{\mathsf{G}_0}} \Rightarrow 1] \leq \Pr[\mathsf{Bad}] \leq \sum_{i=1}^{q_{\mathtt{G}}} \frac{|\mathcal{D}|}{2^n} \leq \sum_{i=1}^{q_{\mathtt{G}}} \frac{q_{\mathtt{G}}}{2^n} \leq \frac{q_{\mathtt{G}}^2}{2^n}\,. \tag{43}$$

Moreover G_0 and the PRG game with its challenge bit fixed to 0 are functionally equivalent, and therefore

$$\Pr[\mathcal{A}^{\mathsf{G}_0} \Rightarrow 1] = \Pr[\mathcal{A}^{\mathsf{PRG}} \Rightarrow 1 \,|\, b = 0]\,. \tag{44}$$

By combining (36), (41), (42), (43), and (44) we obtain the desired result. □

C.4 Proof of Theorem 10

We prove the theorem in two stages. First, we reduce the collision resistance of the vector hash function to the collision resistance of the standard sponge-based hash function. Towards this end we introduce a padding scheme which we call *input separation padding* (isPad). This will allow us to view the evaluation of the vector hash over a triple of inputs as the evaluation of the standard sponge-based hash over an encoding of this triple of inputs into a single string – see Fig. 27. We show that isPad is injective, and consequently that any collision in the vector hash yields a collision in the standard sponge-based hash function. In the second part, we prove that the standard sponge-based hash function, instantiated with a random transformation, is collision-resistant. This part is based on the proof described by Boneh and Shoup in [10]. However, since their proof is in the random permutation model, it requires minor changes to adapt it to the random transformation model. We state it here for completeness.

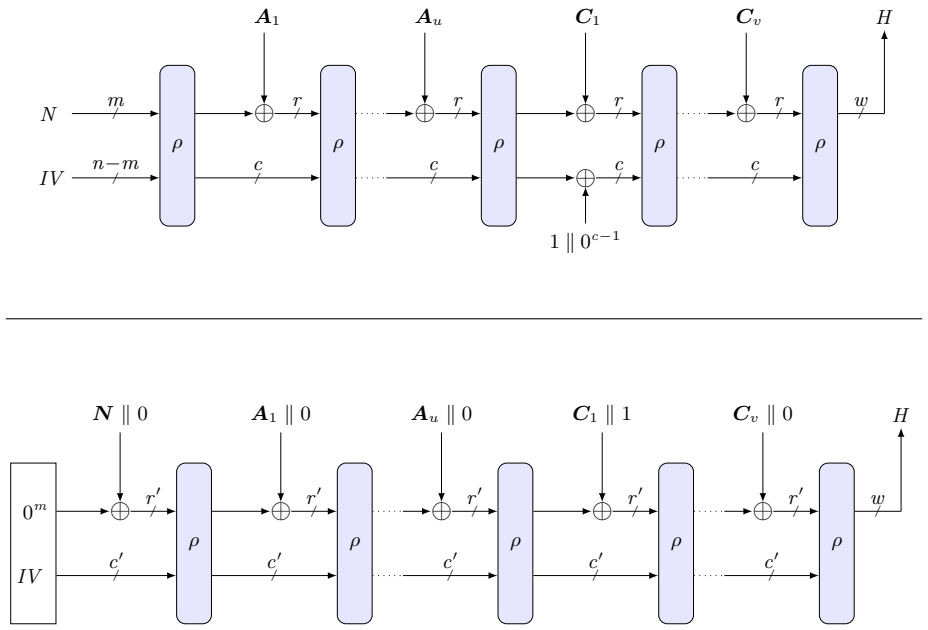

Fig. 27: **Above:** the sponge-based vector hash function SvHash (used in Slae). **Below:** SvHash viewed as the standard sponge-based hash function $\mathcal{H}'$ with its inputs encoded via isPad.

The padding isPad is described in Fig. 28, which in turn makes use of $\mathsf{lpad}^\star$. The latter takes the triple (N, A, C) and an integer r (the rate) as input and outputs $(\boldsymbol{N}, \boldsymbol{A}, \boldsymbol{C})$ such that the size of each is a multiple of r. The first input (N) is simply padded with 0's to a length r, whereas A and C are padded with $\mathsf{lpad}(A, r)$ and $\mathsf{lpad}(C, r)$. Here lpad simply applies the 10^* padding described in Fig. 9 and is already used in SvHash– see Fig. 10. Then $(\boldsymbol{N}, \boldsymbol{A}, \boldsymbol{C})$ is mapped to a string $Z = Z_1 \parallel Z_2 \parallel Z_3$ whose length is a multiple of $r + 1$, where Z_1 is $\boldsymbol{N}$ appended with a 0, Z_2 is $\boldsymbol{A}$ with a 0 inserted after each r bits, and Z_3 is $\boldsymbol{C}$ where a 1 is inserted after the first r bits and a 0 after all other r bits. Note that nonces are assumed to be of fixed size.

First note that lpad is itself an injective encoding. Assume now, towards a contradiction, that the isPad encodings of two distinct triples are equal, i.e. $(N, A, C) \neq (\overline{N}, \overline{A}, \overline{C})$ and $Z = \overline{Z}$. It then immediately follows that $N = \overline{N}$, since the nonces are of fixed size, and that $|Z| = |\overline{Z}|$. There are then only two possible cases: (1) $|Z_2| \neq |\overline{Z}_2|$ or (2) $|Z_2| = |\overline{Z}_2|$. For the first case, assume, without loss of generality, that $|Z_2| < |\overline{Z}_2|$, i.e. $u < \overline{u}$, where u and $\overline{u}$ are the number of r-bit blocks of $\boldsymbol{A}$ and $\overline{\boldsymbol{A}}$, respectively. Then when writing the strings Z and $\overline{Z}$ directly below each other, the block $\boldsymbol{C}_1$ is above $\overline{\boldsymbol{A}}_{u+1}$. However, isPad requires that $\boldsymbol{C}_1$ is followed by a 1 whereas $\overline{\boldsymbol{A}}_{u+1}$ is followed by a 0 which contradicts the assumption that $Z = \overline{Z}$. As for the second case, it follows that $Z_2 = \overline{Z}_2$ and $Z_3 = \overline{Z}_3$. In turn this means that $\boldsymbol{A} = \overline{\boldsymbol{A}}$ and $\boldsymbol{C} = \overline{\boldsymbol{C}}$, and by the injectivity of lpad it also follows that $A = \overline{A}$ and $C = \overline{C}$. Since we already established that $N = \overline{N}$, we have that $(N, A, C) = (\overline{N}, \overline{A}, \overline{C})$, which contradicts our assumption that the inputs are distinct.

$\mathsf{isPad}((N, A, C), r)$	$\mathsf{lpad}^\star((N, A, C), r)$
$(\boldsymbol{N}, \boldsymbol{A}, \boldsymbol{C}) \leftarrow \mathsf{lpad}^\star((N, A, C), r)$	$\boldsymbol{N} \leftarrow N \parallel 0^{r-\lvert N\rvert}$
$Z_1 \leftarrow \boldsymbol{N} \parallel 0$	$\boldsymbol{A} \leftarrow A \parallel \mathsf{lpad}(A, r)$
$Z_2 \leftarrow \boldsymbol{A}_1 \parallel 0 \parallel \boldsymbol{A}_2 \parallel 0 \parallel \ldots \parallel \boldsymbol{A}_u \parallel 0$ **st** $\forall i\ \lvert\boldsymbol{A}_i\rvert = r$	$\boldsymbol{C} \leftarrow C \parallel \mathsf{lpad}(C, r)$
$Z_3 \leftarrow \boldsymbol{C}_1 \parallel 1 \parallel \boldsymbol{C}_2 \parallel 0 \parallel \ldots \parallel \boldsymbol{C}_v \parallel 0$ **st** $\forall i\ \lvert\boldsymbol{C}_i\rvert = r$	**return** $(\boldsymbol{N}, \boldsymbol{A}, \boldsymbol{C})$
return $Z \leftarrow Z_1 \parallel Z_2 \parallel Z_3$	

Fig. 28: The padding schemes isPad and $\mathsf{lpad}^\star$.

Now we can use isPad to reduce the collision resistance of SvHash over triples to the collision resistance of the standard sponge-based hash $\mathcal{H}'$ over strings. Both hash functions are depicted in Fig. 27. Furthermore, note that SvHash can be expressed as $\text{SvHash}(N, A, C) = \mathcal{H}'(\mathsf{isPad}(N, A, C))$. This mapping is such that $\mathcal{H}'$ requires an extra call to the random transformation and has capacity $c' = c - 1$, where c is the capacity of $\mathcal{H}'$. Now, since isPad is injective, any collision that is found on SvHash must correspond to a collision on $\mathcal{H}'$. Moreover since isPad is efficiently computable, any collision on SvHash can easily be translated into a collision on $\mathcal{H}'$. This leads to a straightforward reduction which for any adversary $\mathcal{A}$ against SvHash yields an adversary with similar resources $\overline{\mathcal{A}}$ against $\mathcal{H}'$ such that

$$\mathbf{Adv}^{cr}_{\text{SvHash}}(\mathcal{A}) \leq \mathbf{Adv}^{cr}_{\mathcal{H}'}\left(\overline{\mathcal{A}}\right) . \tag{45}$$

This concludes the first part of the proof and we now go on to prove the collision-resistance of the standard sponge-based hash function $\mathcal{H}'$ with capacity c'.

We assume, without loss of generality, that the adversary makes queries to the random transformation which correspond to its final output. That is, it queries the random transformation on all intermediate states that occur while computing the output. Furthermore, we assume that no redundant queries are made by the adversary, i.e. whenever the adversary makes a ρ query on Y yielding $S = \rho(Y)$, the adversary will never make another ρ query on Y. This assumption is justified by the fact that an adversary gets no additional information from redundant queries and any adversary that makes redundant queries can easily be turned into an adversary which makes no redundant queries.

Similar to the proof in [10], we use a directed graph G to visualize the attack. At the start, the graph contains all bit strings of length n as nodes and no edges. Let Y and S be two nodes. An edge from Y to S is added to the graph, if the adversary makes a query to ρ on Y resulting in S. Since there are no redundant queries, an edge is added to the graph exactly once. In contrast to the proof in [10], we have to deal with the fact that the transformation ρ is not injective and that a node may have more than one incoming edge. For $k \geq 1$, a path of length k can be described by a sequence of $2k$ nodes

$$Y_1, S_2, Y_2, S_3, Y_3, \ldots, S_k, Y_k, S_{k+1} .$$

Moreover it holds that $\hat{Y}_1 = IV$, $\hat{S}_i = \hat{Y}_i$ for $i = 2, \ldots, k$, and the graph G contains edges $Y_i \to S_{i+1}$ for $i = 1, \ldots, k$. The message of this path is defined as a tuple $(M_1, \ldots, M_k)$ of bit strings of length r, where $M_1 = \bar{Y}_1$ and $M_i = \bar{S}_i \oplus \bar{Y}_i$ for $i = 2, \ldots, k$. The result of this path is $M_{k+1} = \bar{S}_{k+1}$. Such a path corresponds to the computation of $\mathcal{H}'(M_1 \parallel \ldots \parallel M_k)$ resulting in output M_{k+1}. We write such paths as

$$M_1|\, Y_1 \to S_2|M_2|\, Y_2 \to \cdots \to S_k|M_k|\, Y_k \to S_{k+1}|M_{k+1} .$$

We can then define a collision in the hash function in terms of the graph G. A collision corresponds to finding a pair of *colliding paths*, which are paths on different messages $(M_1, \ldots, M_k) \neq (M'_1, \ldots, M'_l)$ resulting in messages M_{k+1} and M'_{l+1} which agree in their first w bits, i.e. $\lfloor M_{k+1} \rfloor_w = \lfloor M'_{l+1} \rfloor_w$.

As in [10], we use the notion of *problematic paths*. Consider the following two paths on messages $(M_1, \ldots, M_k) \neq (M'_1, \ldots, M'_l)$ of length k and l, respectively.

$$\begin{aligned} M_1|\, Y_1 \to S_2|M_2|\, Y_2 \to \cdots \to S_{k-1}|M_{k-1}|\, Y_{k-1} \to S_k|M_k|\, Y_k \to S_{k+1}|M_{k+1} \\ M'_1|\, Y'_1 \to S'_2|M'_2|\, Y'_2 \to \cdots \to S'_{l-1}|M'_{l-1}|\, Y'_{l-1} \to S'_l|M'_l|\, Y'_l \to S'_{l+1}|M'_{l+1} \end{aligned}$$

Intuitively, we call a pair of paths *problematic* if their states are equal before the last random transformation is applied. More formally, using the above representation of a path, if $Y_k = Y'_l$ then the paths are called problematic.

Let us denote by CP the event that the adversary finds a pair of colliding paths and by PP the event that the adversary finds a pair of problematic paths. Then the probability of finding a pair of colliding paths, i.e. a collision in the hash function, is bounded as follows

$$\Pr[CP] \leq \Pr[CP \wedge \neg PP] + \Pr[PP] . \tag{46}$$

We start by proving an upper bound for $\Pr[CP \wedge \neg PP]$. This argument closely follows the argument given in [10], however, we obtain a different bound due to the fact that the proof is in the random transformation model. That a pair of paths are both colliding and not problematic means that the final edges correspond to ρ queries on different inputs resulting in outputs where the first w bits are equal. That is, the adversary makes queries to ρ on $Y \neq Y'$ which result in S and S', respectively, such that $\lfloor \bar{S} \rfloor_w = H = \lfloor \bar{S}' \rfloor_w$. For $i \leq j$, let X_{ij} denote the event that the i-th query of the adversary is some value

Y resulting in $S = \rho(Y)$, while the j-th query is on some value $Y' \neq Y$, yielding output $S' = \rho(Y')$ with $\lfloor \bar{S} \rfloor_w = \lfloor \bar{S}' \rfloor_w$. Fix i, j, the random coins of the adversary, and the outputs of all queries made before the j-th query. Then Y, S, and Y' are fixed while S' is distributed uniformly at random over a set of size 2^n. Since there are 2^{n-w} nodes W for which it holds that $\lfloor W \rfloor_w = \lfloor \bar{S} \rfloor_w$, S' must be equal to one of these, thus

$$\Pr[X_{ij}] \leq \frac{2^{n-w}}{2^n} \leq \frac{1}{2^w}\,,$$

which leads to

$$\Pr[CP \wedge \neg PP] \leq \sum_{j=1}^{q} \sum_{i=1}^{j-1} \Pr[X_{ij}] \leq \sum_{j=1}^{q} \sum_{i=1}^{j-1} \frac{1}{2^w} \leq \frac{q(q-1)}{2^{w+1}}\,.$$

Next, we prove an upper bound for $\Pr[PP]$. The probability of finding a pair of problematic paths is closely related to two basic attacks against random sponges introduced by Bertoni et al. [9], which allows us to upper bound the probability. The first one, called *path to an inner state*, asks, for a given bit string $x \in \{0,1\}^{c'}$, to find a path to a node S with inner part equal to x, i.e. $\hat{S} = x$. To match this to our case, we define *E1* to be the event that an adversary finds a path to the inner state IV. The second one, called *inner collision*, asks to find two different paths to nodes S and S' with equal inner states, i.e. $\hat{S} = \hat{S}'$, hence we define *E2* to be the event that an adversary finds an inner collision. In the following, we show that

$$\Pr[PP] \leq \Pr[E1] + \Pr[E2]\,, \tag{47}$$

i.e. every adversary that finds a pair of problematic paths either finds a path to the inner state IV (*E1*) or an inner collision (*E2*). Consider an adversary that finds a pair of problematic paths which are of length k and l, respectively. Write these paths as

$$\begin{aligned} M_1|\,Y_1 \rightarrow S_2|M_2|\,Y_2 \rightarrow \cdots \rightarrow S_{k-1}|M_{k-1}|\,Y_{k-1} \rightarrow S_k|M_k|\,Y_k \rightarrow S_{k+1}|M_{k+1} \\ M'_1|\,Y'_1 \rightarrow S'_2|M'_2|\,Y'_2 \rightarrow \cdots \rightarrow S'_{l-1}|M'_{l-1}|\,Y'_{l-1} \rightarrow S'_l|M'_l|\,Y'_l \rightarrow S'_{l+1}|M'_{l+1} \end{aligned}$$

Due to the fact that the paths are problematic, it holds that $Y_k = Y'_l$ and $(M_1, \ldots, M_k) \neq (M'_1, \ldots, M'_l)$. We further assume that the pair is the shortest among all pairs of problematic paths. This means that $k + l$ is minimal, and without loss of generality, that $k \leq l$. There are three cases to consider, depending on the length of the paths:

1. ($k = 1$ and $l = 1$) In this case, the paths are simply $M_1|\,Y_1 \rightarrow S_2|M_2$ and $M'_1|\,Y'_1 \rightarrow S'_2|M'_2$. The messages of these paths are M_1 and M'_1, respectively. However, it is impossible that both $Y_1 = Y'_1$ and $M_1 \neq M'_1$. By construction it holds that

   $$\bar{Y}_1 = M_1 \neq M'_1 = \bar{Y}'_1$$

   which leads to a contradiction.

2. ($k = 1$ and $l \geq 2$) In this case, we have that $Y_1 = Y'_l$. The adversary has made a ρ query on Y'_{l-1} which resulted in $S'_l = \rho(Y'_{l-1})$. Therefore, it holds that $\hat{S}'_l = \hat{Y}'_l = \hat{Y}_1 = IV$. The first equality follows from the construction, where the inner state between two evaluations of ρ does not change. The second one trivially follows from $Y_1 = Y'_l$, while the third follows again from the construction. By taking the path up to node S'_l, the adversary has found a path to the inner state IV, i.e. event *E1* occurs.

3. ($k \geq 2$ and $l \geq 2$) Here it holds that $Y_k = Y'_l$. Consider now the last but one edges which are

   $$Y_{k-1} \rightarrow S_k|M_k|\,Y_k$$

   $$Y'_{l-1} \rightarrow S'_l|M'_l|\,Y'_l$$

   There are two cases depending on the absorbed messages M_k and M'_l:

   (a) The absorbed messages are equal, i.e. $M_k = M'_l$. Hence, it holds that $S_k = S'_l$. Due to the fact that ρ is a random transformation we have to further distinguish between the following sub-cases:

i. ($Y_{k-1} = Y'_{l-1}$) This contradicts the minimality of $k + l$, as we can find a shorter pair of problematic paths by throwing away the last edge in each path, while the truncated messages $(M_1, \ldots, M_{k-1})$ and $(M'_1, \ldots, M'_{l-1})$ still differ.

ii. ($Y_{k-1} \neq Y'_{l-1}$) In this case, both Y_{k-1} and Y'_{l-1} map to $S_k = S'_l$, therefore the adversary has found an inner collision on the nodes S_k and S'_l, i.e. event *E2* occurs.

(b) The absorbed messages are different, i.e. $M_k \neq M'_l$. This yields $\bar{S}_k \neq \bar{S}'_l$ which further implies $S_k \neq S'_l$ and $Y_{k-1} \neq Y'_{l-1}$. On the other hand, the absorbed messages do not affect the inner state of S_k and S'_l which ensures $\hat{S}_k = \hat{S}'_l$. Combining these, it holds that the adversary has found an inner collision on the nodes S_k and S'_l by taking the paths up to S_k and S'_l, i.e. event *E2* occurs. We emphasise that $Y_{k-1} \neq Y'_{l-1}$ implies that the paths are indeed distinct.

This proves inequality (47). Thus, we can upper bound $\Pr[PP]$ using the following bounds on $\Pr[E1]$ and $\Pr[E2]$ by Bertoni et al. [9], where q denotes the number of queries the adversary makes to ρ.

$$\Pr[E1] \leq \frac{q}{2^{c'}} \qquad \Pr[E2] \leq \frac{q(q+1)}{2^{c'+1}}$$

Substituting everything into (46) yields

$$\Pr[CP] \leq \frac{q(q-1)}{2^{w+1}} + \frac{q}{2^{c'}} + \frac{q(q+1)}{2^{c'+1}} \leq \frac{q(q-1)}{2^{w+1}} + \frac{q(q+2)}{2^{c'}}\,. \tag{48}$$

By combining (45) and (48) we get

$$\mathbf{Adv}^{cr}_{\text{SvHash}}(\mathcal{A}) \leq \mathbf{Adv}^{cr}_{\mathcal{H}'}\left(\overline{\mathcal{A}}\right) \leq \Pr[CP] \leq \frac{q(q-1)}{2^{w+1}} + \frac{q(q+2)}{2^{c'}}\,, \tag{49}$$

and then replacing $c' = c - 1$ we obtain the desired result:

$$\mathbf{Adv}^{cr}_{\text{SvHash}}(\mathcal{A}) \leq \frac{q(q-1)}{2^{w+1}} + \frac{q(q+2)}{2^{c-1}}\,.$$

□