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@crisp-da.de
[2] Cryptoplexity, Technische Universität Darmstadt, Germany
christian.janson@cryptoplexity.de
[3] CDC, Technische Universität Darmstadt, Germany
pstruck@cdc.informatik.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) [10]. 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 [13], 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, 12, 14, 22, 24] have settled for and argued that non-adaptive leakage security often suffices in practice. Moreover, as discussed in [24], 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 [10] 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 [19] 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 [10] instantiated with a random transformation (T-sponge) instead of a random permutation (P-sponge). In [10] this was proposed without proof, instead its security was argued based on its apparent similarity to the GGM construction [15] and the corresponding results in [14,22] 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 [14,15,22] – 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. Nevertheless, in both cases we are able to show the intuitive claim made in [10] that λ bits of leakage can be compensated for by increasing the capacity by λ bits.

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 [10] 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 [7,8,16,20] 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 [18,23]. 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 [11] 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). Furthermore, while the duplex is a more general construction than SlFunc, we prove better security bounds that allow for a more efficient realisation for the same level of security. Essentially for λ bits of leakage and absorption rate rr, their security bound degrades by $\lambda(rr+1)$ whereas ours degrades by $\lambda + rr$. In addition, we leverage the leakage resilience of SlFunc to construct the leakage-resilient AEAD scheme Slae.

Other independent and concurrent work by Guo et al. [17] 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. We conclude with Section 5 where we cover the security of the sponge-based primitives used to instantiate FGHF′ and thereby obtain Slae. The full details of the proofs can be found in the full version of this paper. We conclude in Section 6 with some remarks on implementing Slae.

2 Preliminaries

We start by reviewing the basic tools and definitions that we require for our results. We begin by establishing some notation.

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 will 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* $\text{AEAD} = (\mathcal{E}, \mathcal{D})$ 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* $\text{MAC} = (\mathcal{T}, \mathcal{V})$ 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 $\mathcal{V} : \mathcal{K} \times \mathcal{X} \times \{0,1\}^t \to \{\top, \bot\}$ takes as input a key K, a value X, and a tag T to return either $\top$ indicating a valid input or $\bot$ 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 [19] 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 $\bot$ 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. Output is produced from the sponge during the squeezing phase in one or more iterations, r bits at a time.

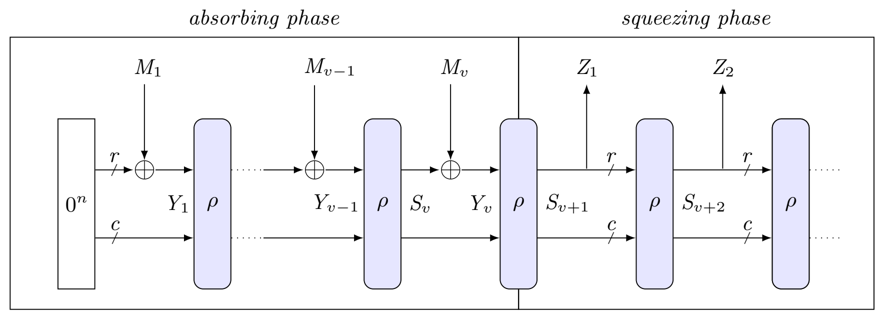

Fig. 1: Illustration of the plain sponge construction.

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 [13]. 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 [14]. 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 [22]. 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 [22], 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 [22], 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 [13]. Furthermore, they prove composition theorems analogous to [19] 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.

Games INDaCPLA

procedure Initialize

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

procedure `Enc`(N, M)

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

procedure `LEnc`(N, M, L)

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

procedure `LDec`(N, C, L)

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

procedure Finalize (b')

return $(b' = b)$

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

[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.

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* $\mathsf{INDaCPLA}$ *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:*

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

Game LPRF

procedure Initialize

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

procedure $\mathtt{LF}(X, L)$

$y \leftarrow \mathcal{F}(K, X)$
$\Lambda \leftarrow L(K, X)$
return (y, Λ)

procedure $\mathtt{F}(X)$

if $b = 0$
  if $f[X] = \perp$
    $f[X] \twoheadleftarrow \{0,1\}^t$
  return $f[X]$
else
  return $\mathcal{F}(K, X)$

procedure Finalize (b')

return $(b' = b)$

Fig. 3: Game used to define LPRF 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.

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:*

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

Game LUF

procedure Initialize

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

procedure $\mathtt{F}(X)$

$\mathcal{S} \leftarrow_{\cup} X$
$y \leftarrow \mathcal{F}(K, X)$
return y

procedure $\mathtt{Lkg}(X, L)$

$\Lambda \leftarrow L(K, X)$
return Λ

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

$y \leftarrow \mathcal{F}(K, X)$
if $X \notin \mathcal{S} \wedge y = y'$
  $\mathsf{win} \leftarrow \mathsf{true}$
return $(y = y')$

procedure Finalize

return (win)

Fig. 4: Game used to define LUF security.

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:*

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

Unforgeability in the Presence of Leakage. For message authentication we will require the analogue of strong unforgeability in the leakage setting

Game SUFCMLA

procedure Initialize

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

procedure $\texttt{Vfy}(X, T)$

if $b = 0$
  return $\perp$
else
  $v \leftarrow \mathcal{V}(K, X, T)$
  return v

procedure $\texttt{LTag}(X, L)$

$\Lambda \leftarrow L(K, X)$
$T \leftarrow \mathcal{T}(K, X)$
return (T, Λ)

procedure $\texttt{LVfy}(X, T, L)$

$\Lambda \leftarrow L(K, X, T)$
$v \leftarrow \mathcal{V}(K, X, T)$
return (v, Λ)

procedure Finalize (b')

return $(b' = b)$

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

(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.

Definition 4 (SUF-CMLA Security). *Let* $\text{MAC} = (\mathcal{T}, \mathcal{V})$ *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:*

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

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 [21] to the leakage setting. This is defined formally below.

Definition 5 (LAE Security). *Let* $\text{AEAD} = (\mathcal{E}, \mathcal{D})$ *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 leakage functions in the respective sets* $\mathcal{L}_{AE}$ *and* $\mathcal{L}_{VD}$*, its corresponding* LAE *advantage is given by:*

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

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.

Generic Composition in the Leakage Setting. The N2 construction was introduced in [19] 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.

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:*

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

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

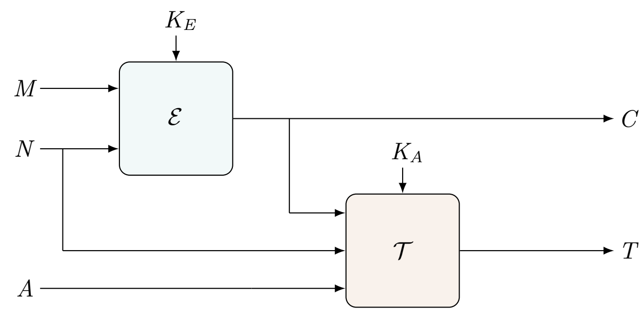

Fig. 7: Graphical representation of the N2 construction.

AEAD scheme called Isap [10]. 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 [10] 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.

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

Slae-$\mathcal{E}(K, N, A, M)$

parse K **as** $K_E \parallel K_A$
$C \leftarrow \text{SlEnc-}\mathcal{E}(K_E, N, M)$
$T \leftarrow \text{SlMac-}\mathcal{T}(K_A, (N, A, C))$
$\overline{C} \leftarrow C \parallel T$
return $\overline{C}$

Slae-$\mathcal{D}(K, N, A, \overline{C})$

parse K **as** $K_E \parallel K_A$
parse $\overline{C}$ **as** $C \parallel T$
$v \leftarrow \text{SlMac-}\mathcal{V}(K_A, (N, A, C), T)$
if $v = \top$
  $M \leftarrow \text{SlEnc-}\mathcal{D}(K_E, N, C)$
  return M
else
  return $\bot$

Fig. 8: High-level description of Slae in terms of SlMac and SlEnc.

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

$T' \leftarrow \text{SlMac-}\mathcal{T}(K_A, (N, A, C))$
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.

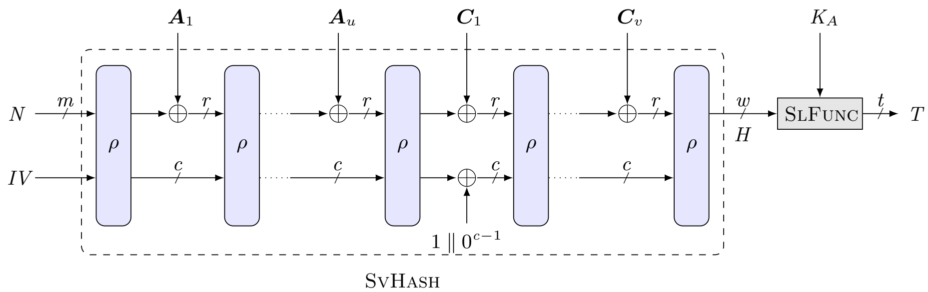

Fig. 10: Graphical illustration of SlMac-$\mathcal{T}$.

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 $\rho(N \,\|\, IV)$, 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 $\rho(K_A \,\|\, IV)$ 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.

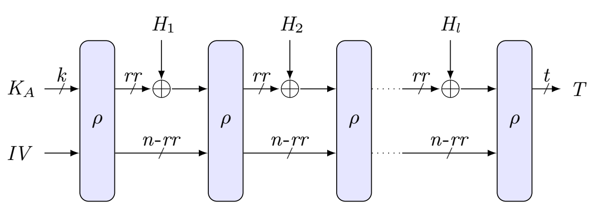

Fig. 11: Graphical illustration of SlFunc.

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-$\mathcal{E}(K_E, N, M)$

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

SlEnc-$\mathcal{D}(K_E, N, C)$

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

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

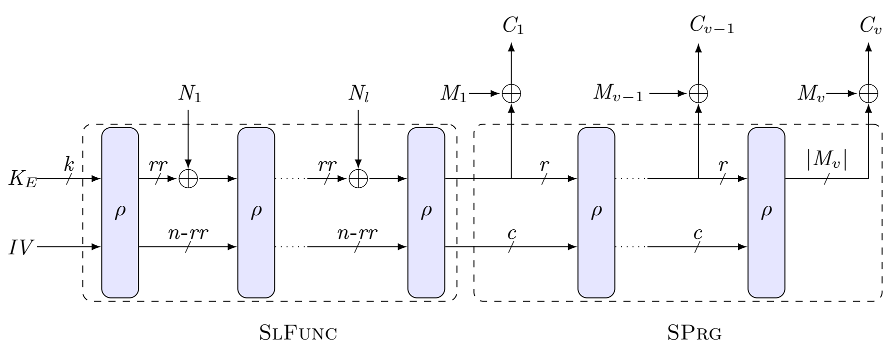

Fig. 13: Graphical illustration of SlEnc-$\mathcal{E}$.

```
SPrg(seed, L)
-------------------------------
v ← ⌈L/r⌉
S_1 ← seed
for i in {1, ..., v − 1}
   S_{i+1} ← ρ(S_i)
R ← S̄_1 ‖ ... ‖ S̄_v
return ⌊R⌋_L
```

Fig. 14: Pseudocode description of SPrg.

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 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.

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. 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 thereby showing that the last round in ISAPMAC is essentially redundant.

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 [19] 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 $\text{SE}[\mathcal{F}, \mathcal{G}]$, and the vector MAC component, denoted by $\text{MAC}[\mathcal{H}, \mathcal{F}']$, 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

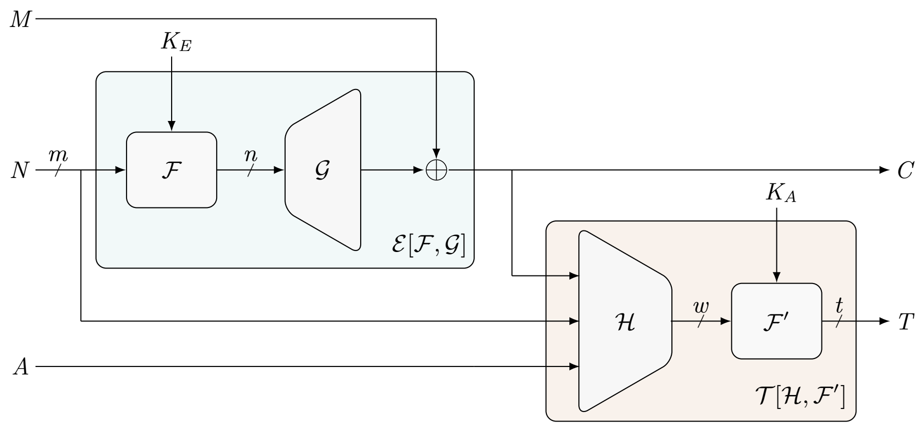

Fig. 15: Graphical representation of the FGHF′ construction. It corresponds to the N2 composition of $\textsc{Se}[\mathcal{F}, \mathcal{G}] = (\mathcal{E}, \mathcal{D})$ and $\textsc{Mac}[\mathcal{H}, \mathcal{F}'] = (\mathcal{T}, \mathcal{V})$ which are in turn composed of a fixed-input-length LPRF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

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 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 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 the full version of this paper.

Theorem 2. *Let* 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* 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 the full version of this paper.

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* $\text{MAC}[\hat{\mathcal{F}}]$ *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* $\text{MAC}[\hat{\mathcal{F}}]$*, there exists an adversary $\mathcal{A}_{luf}$ against $\hat{\mathcal{F}}$ such that:*

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

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:*

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

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 $\text{MAC}[\mathcal{H},\mathcal{F}']$ to that of its building blocks $\mathcal{H}$ and $\mathcal{F}'$.

Corollary 1. *Let* $\text{MAC}[\mathcal{H},\mathcal{F}']$ *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* $\text{MAC}[\mathcal{H},\mathcal{F}']$ *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:*

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

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 $\textsc{Mac}[\boldsymbol{\mathcal{H}}, \boldsymbol{\mathcal{F}'}]$ is LPRF Secure

The final piece needed to apply Theorem 1 is to show that $\textsc{Mac}[\mathcal{H}, \mathcal{F}']$, 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 the full version of this paper.

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:*

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

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:*

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

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 λ bits of leakage can be compensated for by increasing the capacity, the key, and the output (in the case of LUF security) by λ bits. While this may seem intuitive, and indeed this was already conjectured informally in [10], the actual proofs are fairly involved. 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 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 the full version of this paper.

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* $[L_1, \ldots, L_l]$ *where each component maps* n *bits to* λ *bits such that* $\mathcal{L}_\lambda = \{[L_1, \ldots, L_l]\}$*, it holds that:*

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

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*,* F*, and* 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 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* $[L_1, \ldots, L_{l+1}]$ *where each component maps* n *bits to* λ *bits such that* $\mathcal{L}_\lambda = \{[L_1, \ldots, L_{l+1}]\}$*, it holds that:*

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

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*,* F*,* Lkg*, and* 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* $q_{\mathtt{Guess}} + (l+1)(q_{\mathtt{F}} + q_{\mathtt{Lkg}} + q_{\mathtt{Guess}}) \leq 2^{n-1}$*.*

5.2 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 the full version of this paper.

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

$$\mathbf{Adv}^{\mathrm{prg}}_{\mathrm{SPRG}}(\mathcal{A}) \leq \sum_{i=1}^{v_{max}-1} \left(\frac{q_{\mathtt{G}} q_\rho}{2^c - q_\rho} + \frac{q_\rho q_{\mathtt{G}} + 2q_{\mathtt{G}}^2 (v_{max} - i)}{2^n} \right) + \frac{q_{\mathtt{G}}^2}{2^n} .$$

In the above, $\mathcal{A}$ can make q_ρ queries to the random transformation ρ and $q_{\mathtt{G}}$ queries to the challenge oracle $\mathtt{G}$ *of size at most L_{max}, and $v_{max} = \lceil \frac{L_{max}}{r} \rceil$.*

5.3 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 the full version of this paper.

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:*

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

5.4 Concrete Security of Slae

A bound for the security of Slae is directly obtained by combining Theorem 6 with Theorems 7 –10. It 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

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 λ bits of leakage per transformation call can be compensated for by increasing the sponge capacity by λ bits. 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 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 was co-funded by the DFG as part of project P2 within the CRC 1119 CROSSING. Struck was funded by the DFG as part of project P1 within the CRC 1119 CROSSING.

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. 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.
11. C. Dobraunig and B. Mennink. Leakage resilience of the duplex construction. Cryptology ePrint Archive, Report 2019/225, 2019. https://eprint.iacr.org/2019/225.
12. 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.
13. S. Dziembowski and K. Pietrzak. Leakage-resilient cryptography. In *49th FOCS*, pages 293–302. IEEE Computer Society Press, Oct. 2008.
14. 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.
15. O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, Oct. 1986.
16. 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.

17. 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.
18. 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.
19. 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.
20. 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.
21. P. Rogaway. Authenticated-encryption with associated-data. In V. Atluri, editor, *ACM CCS 2002*, pages 98–107. ACM Press, Nov. 2002.
22. 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.
23. 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.
24. 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.