Randomness Generation for Secure Hardware Masking – Unrolled Trivium to the Rescue

Gaëtan Cassiers[1], Loïc Masure[2], Charles Momin[2], Thorben Moos[2], Amir Moradi[3] and François-Xavier Standaert[2]

[1] Graz University of Technology, Graz, Austria
firstname.lastname@iaik.tugraz.at

[2] Crypto Group, ICTEAM Institute, UCLouvain, Louvain-la-Neuve, Belgium
firstname.lastname@uclouvain.be

[3] Ruhr University Bochum, Horst Görtz Institute for IT Security, Bochum, Germany
firstname.lastname@rub.de

Abstract. Masking is a prominent strategy to protect cryptographic implementations against side-channel analysis. Its popularity arises from the exponential security gains that can be achieved for (approximately) quadratic resource utilization. Many variants of the countermeasure tailored for different optimization goals have been proposed over the past decades. The common denominator among all of them is the implicit demand for robust and high entropy randomness. Simply assuming that uniformly distributed random bits are available, without taking the cost of their generation into account, leads to a poor understanding of the efficiency and performance of secure implementations. This is especially relevant in case of hardware masking schemes which are known to consume large amounts of random bits per cycle due to parallelism.

Currently, there seems to be no consensus on how to most efficiently derive many pseudo-random bits per clock cycle from an initial seed and with properties suitable for masked hardware implementations. In this work, we evaluate a number of building blocks for this purpose and find that hardware-oriented stream ciphers like Trivium and its reduced-security variant Bivium B outperform all competitors when implemented in an *unrolled* fashion. Unrolled implementations of these primitives enable the flexible generation of many bits per cycle while maintaining high performance, which is crucial for satisfying the large randomness demands of state-of-the-art masking schemes. According to our analysis, only Linear Feedback Shift Registers (LFSRs), when also unrolled, are capable of producing long non-repetitive sequences of random-looking bits at a high rate per cycle even more efficiently than Trivium and Bivium B. Yet, these instances do not provide black-box security as they generate only linear outputs. We experimentally demonstrate that using multiple output bits from an LFSR in the same masked implementation can violate probing security and even lead to harmful randomness cancellations. Circumventing these problems, and enabling an independent analysis of randomness generation and masking scheme, requires the use of cryptographically stronger primitives like stream ciphers. As a result of our studies, we provide an evidence-based estimate for the cost of securely generating n fresh random bits per cycle. Depending on the desired level of black-box security and operating frequency, this cost can be as low as $20n$ to $30n$ ASIC gate equivalents (GE) or $3n$ to $4n$ FPGA look-up tables (LUTs), where n is the number of random bits required. Our results demonstrate that the cost per bit is (sometimes significantly) lower than estimated in previous works, incentivizing parallelism whenever exploitable and potentially moving low randomness usage in hardware masking research from a primary to secondary design goal.

Keywords: Hardware Masking · Randomness · Side-Channel Analysis · Trivium

1 Introduction

Side-channel analysis is known to be a significant threat to implementations of cryptographic algorithms and protocols that must operate under adversarial exposure. If untrusted individuals gain physical access to a cryptographic device, measurable quantities such as power consumption or electromagnetic emanations during the processing of secret material can be monitored to extract sensitive information. This type of attack was first demonstrated by Kocher et al. in 1999 [KJJ99] and has since inspired a great deal of research on the theory and practice of implementation security. Masking (also known as secret sharing) is a well-known countermeasure to protect cryptographic implementations from side-channel analysis adversaries. It was first proposed by Chari et al. in 1999 [CJRR99], and is nowadays widely considered to be the most potent protection mechanism against passive physical adversaries. Its core principle is based on splitting each potentially sensitive intermediate variable into a discrete number of shares, in such a way that only the combination of all shares reveals information about the secrets. Using this technique, adversaries can be forced (implicitly or explicitly) to collect information on all individual shares before combining them to reconstruct the sensitive intermediates. Yet, learning information about the secrets from partial information on their shares is a hard problem. To be precise, if the leakage of the individual shares is sufficiently noisy and independent, masking is capable of providing exponential security in the number of shares against adversaries trying to extract sensitive information from side-channel observations – see for example [PR13, DDF14, DFS15] for a formalization. The implementation cost of this countermeasure is typically estimated to be quadratic in the number of shares due to the known complexity of masked multiplications [ISW03].

Yet, and especially when masking is applied to hardware implementations of cryptographic algorithms, the independence assumption is often invalidated by physical defaults such as glitches [MPG05], transitions [CGP+12] and couplings [CBG+17]. It has taken the research community several years to develop generalizable strategies to avoid these issues at the conceptual level. The first solid approaches toward preventing glitches from recombining shares in hardware masking schemes came in the form of threshold implementations [NRR06, NRS08]. A few years later, other masking schemes with a lower number of shares to achieve a given protection order (compared to threshold implementations) were proposed [RBN+15, GMK16, GMK17, GM17]. These schemes also targeted higher-order security for the first time, after it was found that higher-order threshold implementations suffer from conceptual flaws [BGN+14a, Rep15, RBN+15]. Simultaneously, independent researchers started investigating the requirements needed to securely compose masked building blocks into full cipher implementations, resulting in the security notion of Strong Non-Interference (SNI) [BBD+16]. Following these advances, the robust probing model was introduced to allow formal analysis of composability and robustness against physical defaults jointly [FGP+18]. Subsequently, it was shown at CHES 2019 that many previously proposed hardware masking schemes suffer from composability flaws under the robust probing model at higher orders, giving substantial evidence that a formal analysis is beneficial to properly generalize schemes to arbitrary orders [MMSS19]. Finally, in 2020, a new composability notion called Probe Isolating Non-Interference (PINI) was introduced to allow trivial composition of masked implementations of linear and non-linear functions [CS20, CGLS21]. Based on this notion, new masked gadgets have been introduced [CGLS21, KM22b, KM22a] along with tools that allow formal verification of their properties [KSM20, CGLS21] and automated generation of full cipher masked hardware circuits based on PINI gadgets [KMMS22].

1.1 Motivation

Clearly, the past few years have advanced our collective understanding of how to best capture physical defaults such as transitions, couplings, and glitches through proven design principles. This, in turn, enables fast and efficient masked hardware implementations that can leverage parallelism to simultaneously operate on the individual shares of a secret intermediate without sacrificing security against physical adversaries. In order to achieve this symbiosis between performance and security, hardware masking schemes often consume a notoriously large number of random bits per cycle. Indeed, in the past several years, a strong trend is observable in the community towards constructing masked hardware implementations entirely from circuit gadgets that are provably robust probing secure and composable (e.g., [CS21, KSM22, MCS22, KMMS22]). This is done in order to automatically derive guarantees that the resulting full hardware implementations are provably secure themselves. The gadgets used for this purpose typically consume a certain amount of fresh random bits per clock cycle to satisfy the required properties, e.g., $d(d-1)/2$ bits for one 2-input AND gate with d shares in domain-oriented masking [GMK16] or HPC2 [CGLS21]. Naturally, full ciphers composed of many such gadgets also need many bits of fresh randomness per cycle, especially if those implementations leverage parallelism, are optimized for low latency and if higher-order protection is required. Hence, it is not uncommon for parallel masked hardware implementations of full block ciphers to require hundreds or even thousands of independent, uniformly distributed, and unpredictable random bits per cycle. Recent works presented at CHES 2022 [KMMS22] and CCS 2022 [KM22b] list exemplary cost and performance figures for masked round-based cipher implementations that demand multiple thousands of freshly random bits in each cycle. Even for serialized implementations or single S-boxes it is not uncommon to see requirements in the range of hundreds of bits per cycle. Despite this huge demand, most works on the topic have considered the efficient generation and distribution of these bits to be beyond their scope. In fact, the majority of publications in the masking literature simply assumes the existence of robust and high-entropy randomness sources. We argue that the lack of focus on this topic can have negative consequences, since concurrent randomness generation is a crucial part of masked implementations, especially in hardware. Failing to include this component in the evaluation of implementations clearly leads to a poor understanding of the efficiency and performance of secure circuits.

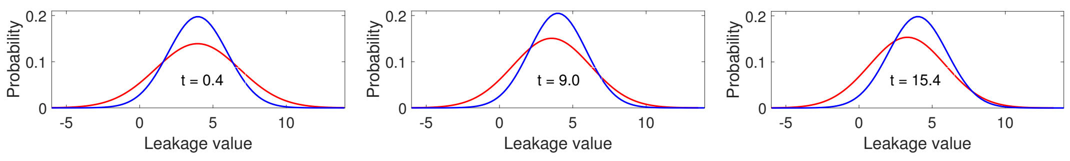
1.2 Research Question

Our work aspires to answer the question whether such huge demands for randomness can be satisfied in hardware and at what cost, as this aspect has been neglected in most previous publications. While reducing randomness requirements is an often researched topic (initiated in [BBP+16], with many follow-up works), studies of the actual cost of randomness for masking are surprisingly missing in the literature. For example, the authors of the recent [KM22a] consider quite different approaches in order to provide meaningful comparisons between masking schemes including randomness generation, such as an individual 32/64-bit LFSR for each bit of randomness required per cycle or a Keccak-based PRNG, which both turn out to be rather expensive. Hence, it is our goal to find more efficient solutions while also clarifying the relevant security properties that must be satisfied in the masking context.

Once a reliable estimate of the cost of producing a certain number of random bits per cycle is established, it becomes much easier to decide on crucial trade-offs in masked hardware implementations. Additionally, it will help to answer the question whether schemes that minimize randomness requirements are more worthwhile, or whether it is better to optimize other parameters such as latency or area at the cost of higher randomness usage.

1.3 TRNG vs. PRNG

Whenever randomness is required in a design, at least some initial entropy must come from a true randomness/noise source, usually extracted by a True Random Number Generator (TRNG). Yet, as we will confirm in the paper, TRNGs tend to be either fairly slow or resource-hungry, making the cost of generating each truly random bit significant. Thus, it is a common strategy to use Pseudo Random Number Generators (PRNGs), which are generally considered to be much more efficient than TRNGs, to stretch the initial seed (obtained once at power-up) into many pseudo-random bits whenever needed during runtime.



(a) $H = 4.0$ (b) $H = 3.8$ (c) $H = 3.6$

Figure 1: Fitted distributions obtained for 5000 fixed-vs-fixed samplings of the leakage function $l(x_m) + l(m) = HW(x_m) + HW(m) + \mathcal{N}(\mu, \delta^2)$ with $x_m = x \oplus m$, $x \in \{0,1\}^4$, $m \in \{0,1\}^4$, $\mu = 0$, $\delta^2 = 2$ and $HW(\cdot)$ being the Hamming weight function. Red curves are obtained for $x = 0$, blue curves for $x = 15$. First-order Welch's t-statistics result given in the middle of the figures; Shannon entropy H of the mask m given below the figures.

1.4 Requirements for Masking Randomness

Many different PRNG constructions have been proposed in the literature for a variety of applications. Of course, depending on the concrete use case, different properties are required from the random numbers. Cryptographically strong PRNGs, as required in many cryptographic protocols for generating keys, nonces or salts, should produce output that is indistinguishable from genuine randomness for computationally bounded adversaries. However, the requirements are not always that strong. In the concrete case of masking, adversaries typically cannot directly obtain the output of the PRNGs. In fact, there is no output that depends on the generated random bits at all, since these values are only used for internal randomization of intermediates computations, and the final results of the masked operations are unmasked internally before being released to outside observers. Hence, adversaries can only obtain a noisy version of the generated random bits from their side-channel observations (especially noisy in hardware if generated in parallel to the masked cipher implementation), leaving the possibility to perform direct state recovery attacks on the PRNG quite theoretical (see [JD06, BMV07, CMM14, MCB$^+$22]). As a result, many previous works have opted for random number generators without cryptographic strength for mask generation, such as Linear Feedback Shift Registers (LFSRs).

LFSRs are arguably the most simple primitive for generating long non-repetitive sequences of random-looking, uniformly distributed bits from an initial seed.[1] However, as mentioned above, LFSRs cannot provide black-box security, and their linear output can be distinguished from true randomness using statistical test suites such as the one proposed by the National Institute of Standards and Technology (NIST) [BRS$^+$10]. This raises doubts regarding their suitability to fill the two main requirements for (pseudo-)random numbers in masking contexts, namely *uniformity* and *unpredictability* [GSF13]. Without uniformity, masking schemes cannot keep their security promises. The example in Figure 1 illustrates that even a small bias in the sampling of random masks can lead to a reduction of the protection order, even if noise and independence assumptions are fulfilled. Despite the fact that x_m and m leak independently and in the presence of noise, the two rightmost figures show distributions that can be confidently distinguished by their means using statistical hypothesis tests corresponding to first-order leakage, i.e., the null hypothesis is rejected. This is caused by a biased distribution of m, with a reduced Shannon entropy H (recall that the entropy is maximized for a uniform distribution). The reason why the unpredictability of random numbers is important in masking contexts is even more obvious. If an adversary can predict the random bits and has knowledge of the data being processed (known-plaintext scenario), she can compute all intermediate values that are actually processed inside the cryptographic implementation and perform attacks in the same trivial way as on unprotected circuits. Since all future outputs (at least until the next re-seeding) can be calculated once the internal state of a deterministic PRNG is discovered, state recovery attacks are the most relevant threat to

[1] An LFSR of degree m has a maximum period of $2^m - 1$ and repeats the output sequence every 2^m steps.

the unpredictability requirement. With respect to LFSRs, once an adversary has obtained a sufficient number of consecutive output bits, state recovery is trivial. More precisely, if the feedback polynomial is known – with m being the LFSR's degree – an attacker only needs to observe m consecutive output bits to know its internal state and predict all further outputs. When the feedback polynomial is unknown, the attacker typically requires $2m$ consecutive output bits [PP10]. However, as mentioned before, the attacker model in the context of masking does not allow direct access to the generated random bits, making the application of such attacks difficult for noisy leakages. Uniformity and unpredictability also implicitly require that the adversary can neither bias nor control the generation of randomness.

1.5 Masking Randomness in Previous Works

In the state-of-the-art hardware masking literature, different authors have used various techniques to generate the random bits required for their experimental analyses. An AES-128 in counter (CTR) mode has been used in [BGN+14b] to provide 44 random bits per clock cycle. Assuming that a round-based AES-128 implementation requires at least 12 500 gate equivalents (GE) of area (the smallest value listed in the comparison of [UHM+20]) and produces 128 output bits every 10 clock cycles, the area cost of generating one random bit per cycle can be estimated to be 977 GE. Other works like [CRB+16] instantiated reduced-round variants of the low-latency cipher PRINCE [BCG+12] in Output FeedBack (OFB) mode to rapidly generate a high number of random bits per clock cycle. It is not stated how many of the PRINCE rounds were removed, but calculating based on a full PRINCE as unrolled implementation which requires approximately 8 000 GE [BCG+12] of area and generates 64 output bits per cycle, the cost of generating one random bit per cycle is about 125 GE. For a round-reduced variant, this cost might be halved at the expense of a reduced security level. In [SBHM20], a sponge-based PRNG [BDPA10] using a variant of Keccak [BDPA13] is used to generate 976 random bits every clock cycle. The design details provided are insufficient to estimate the cost per bit of the concrete construction used. However, a round-based Keccak-f[200] permutation requires about 5 000 GE of area and runs for 18 clock cycles [KY10]. Using the construction proposed in [BDPA10], either 64 or 96 bits are obtained per call to Keccak-f[200], resulting in a minimum cost of 938 GE to produce one random bit per cycle. As mentioned earlier, there are also a number of works that use LFSRs to generate random values for masking. For instance, 31-bit LFSRs are employed in [MMW18, Moo19, SM21, KMMS22, KSM22] in such a way that each required random bit is generated by a dedicated LFSR that is randomly seeded on power-up. Following the same principle, the authors of [KM22a] have considered such a 31-bit LFSR, a 64-bit LFSR, and different variants of Keccak-Sponge-based PRNGs, and have reported the overhead of a hardware masking scheme including the area required for the necessary PRNGs. The cost of generating one random bit per cycle is estimated as 286 GE and 565 GE for the 31-bit and 64-bit LFSRs respectively [KM22a]. Further, it has been explored in [PYR+16] whether evolutionary computation can be beneficial in the design and optimization of lightweight PRNGs for masking applications. The authors have proposed multiple PRNG variants that have passed all tests of the NIST statistical test suite [BRS+10]. The most efficient one, based on Cartesian genetic programming, is said to have a throughput-area ratio of 68.14 Mbps/GE based on the NanGate 45 nm library. While this is significantly more efficient than all other approaches mentioned above, it is entirely unclear whether this primitive provides any resistance against modeling or state-recovery attacks. The other variants proposed by the authors, which are said to be prediction resistant, are significantly more expensive. To summarize, the current state of the art lacks dedicated, efficient, and well-studied solutions for generating masking randomness and has mostly relied on ad hoc approaches so far.

We note that between all these publications, there is one work about multiplicative masking of the AES which applies one of the solutions we recommend in this work, namely an unrolled Trivium instance to generate multiple masking randomness bits in parallel [MRB18]. It is easy to overlook, as the use of Trivium is only mentioned at the very end of the work's

Appendix, without any further reasoning or cost analysis. But it appears that the authors had a promising idea which, unfortunately, does not seem to have caught on in the community (as evident by all the publications which since reverted to less suitable solutions).

1.6 Our Contributions

In this work, we focus on the problem of efficiently and securely generating randomness in hardware with properties suitable for use in masked implementations. As a first step, we briefly investigate the efficiency of state-of-the-art on-chip TRNGs, focusing mainly on a high-throughput, low-area TRNG proposed at CHES 2018 [YRG+18], and discard the possibility of using only TRNGs for the entire randomness generation of masking due to their sub-optimal cost-performance trade-off. In consequence, we conclude that PRNGs are indeed a better choice for this purpose. While LFSRs are generally considered a poor choice for random number generation in cryptographic contexts due to the linear dependency between their output bits, we include them in our comparison of potential primitives for mask generation and evaluate their security in masking contexts later. Alongside LFSRs, we analyze a number of efficient cryptographic building blocks with potential for secure mask generation based on their throughput-area ratio. We chose these building blocks in part based on prior reports comparing the 32 primitives that survived to Round 2 of NIST's Lightweight Cryptography (LWC) standardization process [oSN17] and the 8 stream ciphers in Profile 2 (hardware) that reached the final phase of eSTREAM, the ECRYPT stream cipher project [oEiCE04]. We have included the following ones in our comparisons:

- The lightweight primitive *Subterranean 2.0* [DMMR20], which offers by far the best throughput-area ratio among NIST LWC Round 2 candidates according to [AZ21].
- The cross-platform permutation *Gimli* [BKL+17], which is among the highest throughput primitives of NIST LWC Round 2 candidates according to [AZ21].
- The ultra low-latency block cipher *SPEEDY* [LMMR21], whose variant `SPEEDY-5-192` is claimed to provide the best throughput-area ratio among low-latency ciphers.
- The stream ciphers *Trivium* [Can06, CP08], *Grain v1* (both 80- and 128-bit variant) [HJM07, HJMM06] and *MICKEY 2.0* (both 80- and 128-bit variant) [BD08] which offer the most promising performances in throughput-area ratio among the eSTREAM competition's Phase-3 candidates, as reflected by different comparative efforts [GB08, GLB+06, BKSQ07, GSB07, Rog07, HCK+08, KSPS13, LLL20].

Where possible, we also consider reduced-security variants of these primitives, motivated by the assumption that full cryptographic strength may not be required in our target setting, i.e., randomness used in masking. For Gimli and SPEEDY this means that we consider reduced-round versions. For Trivium, we consider its reduced-security variant Bivium B, which has been introduced to study the cryptanalytic properties of Trivium [Rad06]. Thanks to its unrolled implementation style available in hardware, our throughput-area comparison leaves no doubt that Trivium is the most efficient primitive, outperforming the other candidates by an impressive margin. By the term *unrolling* in the context of stream ciphers we denote the generation of multiple output bits in a single cycle using a single hardware module (see [GB08, GLB+06, GSB07, Rog07, HCK+08, LLL20] and explanations in Section 3). Since the degree or level of unrolling can be chosen arbitrarily for stream ciphers, these primitives offer a high flexibility to hardware designers. Considering primarily the throughput-area ratio, our comparison shows that only LFSRs, when also unrolled to produce multiple bits simultaneously, are able to outperform Trivium and Bivium B. Hence, as a next step we evaluate whether unrolled LFSRs are suitable candidates for mask generation.

In this respect it is noteworthy that, to the best of our knowledge, no previous work has explicitly used unrolled LFSRs to generate randomness for masking. The authors of [KM22a] for example discussed the need to use an independent LFSR for each random bit required per cycle in a masked circuit, and all previous works have apparently followed the same strategy. However, according to our comparison, this is not a cost-effective strategy. Hence, we investigate the security of unrolled LFSRs in two case studies, and find that without great care, this strategy does indeed lead to problems when used in masked implementations, and may render the entire side-channel security null and void. In fact the demonstrated problems extend, although in limited form, to the single-LFSR-per-bit scenario which has more commonly been considered in practice. To summarize, we do not prove that unrolled LFSRs can never be used securely for mask generation – as long as state recovery attacks are not a problem which, as previously mentioned, should hold for high noise levels and could even be formalized with arguments similar to the ones given in [DFH$^+$16].[2] But, our case studies provide clear evidence that when using such linear primitives which lack black-box security guarantees, the randomness generation and the masking scheme *must always* be analyzed jointly to ensure that each random bit is used only in positions where it cannot cause problems. Moving to primitives that offer black-box security, such as Trivium, solves this issue and allows the independent analysis of the masked implementation and the randomness generator, which is much more convenient from the designer's perspective.

In summary, we warn against the use of (unrolled) LFSRs and recommend unrolled Trivium for high security levels and unrolled Bivium B for medium security levels for the efficient generation of randomness for masked hardware implementations. We detail how to use these primitives, discuss their security against side-channel attacks, and finally estimate the resulting cost of generating n random bits per cycle. For Trivium, the asymptotic cost per random bit updated per clock cycle is about 30 GE on ASIC or 4 LUTs on FPGA. Using Bivium B, the cost can even be reduced to 20 GE or 3 LUTs per random bit. These results show that randomness generation is significantly cheaper than estimated by most works in the past, which incentivizes highly parallel (low latency) masked implementations and should motivate researchers to focus on alternative optimization goals than reducing randomness usage in masking schemes. We believe that our conclusions are of positive nature for the physical security community, as implementations using many random bits per cycle are also known to provide superior security levels against more sophisticated attacks (e.g., so-called horizontal attacks [BCPZ16]) compared to low-randomness approaches.

2 Background

In this section, we introduce the primitives that could be considered to generate randomness for masking. We start with TRNGs that we will rule out for performance reasons in Section 3.2. We follow with LFSRs that we will rule out for security reasons in Section 4. We then discuss the different (cryptographic) PRNGs that we will investigate, paying a special attention to Trivium and Bivium that we suggest as the candidates of choice in this paper. We conclude by describing how stream cipher implementations can be unrolled.

2.1 True Random Number Generators (TRNGs)

Whenever randomness is needed in a digital design, at least some initial entropy has to come from an analog noise source, as deterministic digital computation methods are unable to generate true randomness. Thus, TRNGs exploit noise sources based on physical phenomena with unpredictable behavior. An optimal source of entropy would be radioactive decay, since the timing of events at the atomic level is impossible to predict, even with unbounded memory and computational resources. Yet, it is clearly not realistic to sample radioactive decay in integrated circuits to generate random numbers for cryptographic applications. Instead, noise sources inherent to modern integrated circuits are commonly leveraged. These include clock jitter, metastability, thermal noise in resistors, oscillatory metastability, write collisions in dual-port random access memories and random initialization of bi-stable circuits [FD02, FL14].

[2] For sufficiently many and/or large LFSRs so that correlation attacks do not trivially apply [Can11b].

A large number of TRNG designs based on these physical phenomena is discussed and compared in [PMB+16, YRG+18]. The raw random numbers extracted from entropy sources are typically subject to statistical defects and need to be tested and post-processed before being used in applications. Obtaining independent random values with high entropy is therefore a laborious process that comes at a significant cost (e.g., in latency or area).

2.2 Linear-Feedback Shift Registers (LFSRs)

LFSRs are structures that hold an array of bits shifted one position per step in a certain direction. The bit that gets shifted out of the array is typically the output, and the new bit shifted into the array is determined by a feedback function computing a linear combination of a number of state bits. LFSRs consist of clocked storage elements like flip-flops, and the feedback function is typically described by a polynomial. The number of storage elements is the degree of the LFSR. The maximum period, or sequence length, of an LFSR of degree m is $2^m - 1$. LFSRs with maximum period exist for any degree m. Since the LFSR output is determined by a linear combination of the initial state bits only, state recovery attacks are trivial once a sufficient number of consecutive output bits are observed ($2m$ if the feedback polynomial is unknown; m otherwise [PP10]). Due to this lack of black box security, LFSRs are rarely used as standalone primitives in cryptographic applications but more commonly as useful ingredients (e.g., for stream ciphers – see next). Yet, and somewhat surprisingly, they have been employed for the generation of randomness for masking.

2.3 Pseudo-Random Number Generators (PRNGs)

Due to the high cost of generating true random values in integrated circuits, it is common practice to use PRNGs to stretch short sequences of true random bits (called seeds) into long sequences of pseudo-random bits. PRNGs are deterministic polynomial time algorithms constructed from an iterated function [BM82]. They generally use a pair of functions f and g, where $f : \{0,1\}^n \rightarrow \{0,1\}^n$ iteratively updates an n-bit state $s_i = f^i(s_0)$, and $g : \{0,1\}^n \rightarrow \{0,1\}^m$ generates the output bitstream $g(s_0)\|g(s_1)\|\ldots$. The initial state s_0 is derived from the seed, which can be obtained from a TRNG at device power-up. Therefore, all the entropy in the output descends from the initial random seed, and is further limited by the state size of a PRNG. Cryptographically strong PRNGs, required for key or nonce generation in many cryptographic protocols – when properly seeded – should produce output indistinguishable from genuine randomness for computationally bounded adversaries.

PRNGs are natural candidates to generate randomness for masked implementations since they generally have good properties for leakage due to the continuous update of their secret state [YSPY10]. Furthermore their initialization, that may lead to stronger side-channel attack vectors if it had to be synchronized with another communication party [SPY+10], is not needed in this context and can be done with a truly random seed generated on-chip. As most cryptographic primitives, they can be obtained generically from well-investigated building blocks like (tweakable) block ciphers (as in the previous reference) or permutations [BDPA10] – both possibly coming with similar guarantees in terms leakage-resistance [BBC+20]. They can also be obtained from dedicated constructions, usually introduced as stream ciphers. Such dedicated constructions generally correspond to a slightly more aggressive security (margins) vs. efficiency tradeoff compared to generic constructions. The latter appears appealing for our purposes since expensive randomness generation makes the application of higher-order masking prohibitive and, as already mentioned, the adversarial scenario of this generation is different from the stream cipher one (i.e., the adversary sees only the leakage of the PRNG). Concretely, we will primarily investigate Trivium [oEiCE04, Can06] and its Bivium B variant [Rad06] that we detail next. We will also report performance figures for Subterranean 2.0 [DMMR20], Gimli [BKL+17] and SPEEDY [LMMR21]. The former two were among the most promising candidates from the NIST lightweight cryptography standardization process [oSN17], while the latter is a performance-driven block cipher with few rounds. For completeness, we finally provide results for other stream ciphers of the eSTREAM competition [oEiCE04], namely Grain v1 [HJM07] and MICKEY 2.0 [BD08], despite previous works already reporting their lower performances compared to Trivium, and for Kreyvium which is a variant of Trivium with 128-bit security [CCF+16].[3]

2.4 Trivium & Bivium

Trivium is a stream cipher submitted by De Cannière and Preneel to the eSTREAM competition, a multi-year effort to collect compact stream ciphers suitable for widespread adoption [oEiCE04, Can06]. It was selected to be part of the final portfolio [CP08] and has later been standardized as part of the lightweight stream cipher standard ISO/IEC 29192-3. Trivium is based on a combination of three Non-Linear Feedback Shift Registers (NLFSRs) of degree 93, 84 and 111 (288 bits in total) – see Figure 2 for an illustration. It has two input parameters, an 80-bit key and an 80-bit initialization vector (IV). As is common in cryptographic applications, the IV is public, but should take a new value for each encryption session. During the initialization phase, the IV is loaded into the 80 leftmost positions of the upper register, while the key is loaded into the 80 leftmost positions of the middle register. All other bits are set to zero, with the exception of the three rightmost bits of the bottom register, which are set to one. The cipher is then clocked for 1152 steps without producing any keystream, which corresponds to 4 rotations of the state ($4 \cdot 288 = 1152$), that randomizes the content of the registers. After the initialization phase (also called warm-up) is completed, the online phase begins and the keystream is generated. According to the performance comparisons of phase-3 candidates of the eSTREAM competition presented in [GB08, GLB+06, BKSQ07, GSB07, Rog07, HCK+08], unrolled Trivium offers by far the best throughput-area ratio for hardware implementations.

In an attempt to better understand the security of Trivium, Raddum introduced two reduced variants, called Bivium A and Bivium B [Rad06]. Both of them consist of only two of Trivium's NLFSRs, namely the 93-bit and the 84-bit ones. Bivium B is depicted in Figure 3. In Bivium A, the keystream is generated as the sum of 2 state bits, both from the same register. In Bivium B, the keystream is generated as the sum of 4 state bits, 2 from each NLFSR. While no key recovery attack on Trivium with a complexity below 2^{80} is known, there have been effective attacks on both Bivium variants. In 2006, Haddum presented an attack to break Bivium A in about a day [Rad06] by building and solving a system of equations using the output keystream with the initial state bits as the unknowns. He estimated the same attack to require about 2^{56} seconds (about 2^{31} years) on Bivium B. Later in 2007, Maximov and Biryukov presented an attack on Bivium B with complexity $c \cdot 2^{37}$, where the constant c denotes the time required to solve a system of equations (estimated to be $c \approx 2^{14}$) [MB07]. In 2011, it was reported in [HL11] that by guessing 35 variables, Bivium B can be solved in $2^{32.81}$ seconds (about 7.9 years) by MFCS, an algorithm for solving Boolean polynomial equations. In 2019, a key-recovery attack on Bivium B based on Boolean equation solving has been reported [SSD19]. The concrete complexity of the attack is unclear due to ambiguous claims in the paper, but the authors state that about 4 terabytes of memory and a parallel search over 2^{39} threads is required to recover the initial state and the key. We conclude that Bivium A is weaker and that Bivium B can be broken in practical time complexity on large computation clusters. For the rest of the paper we denote Bivium B as Bivium and use it as an aggressive design that may be secure in the leakage-only setting we consider, to gauge the performance gains that such an optimization offers.

2.5 Cautionary Notes

The move from LFSRs to cryptographically strong designs is admittedly gradual and different tweaks can be used in order to add non-linearity to stream ciphers based on shift

[3] We do not provide the details of these additional algorithms due to place constraints.

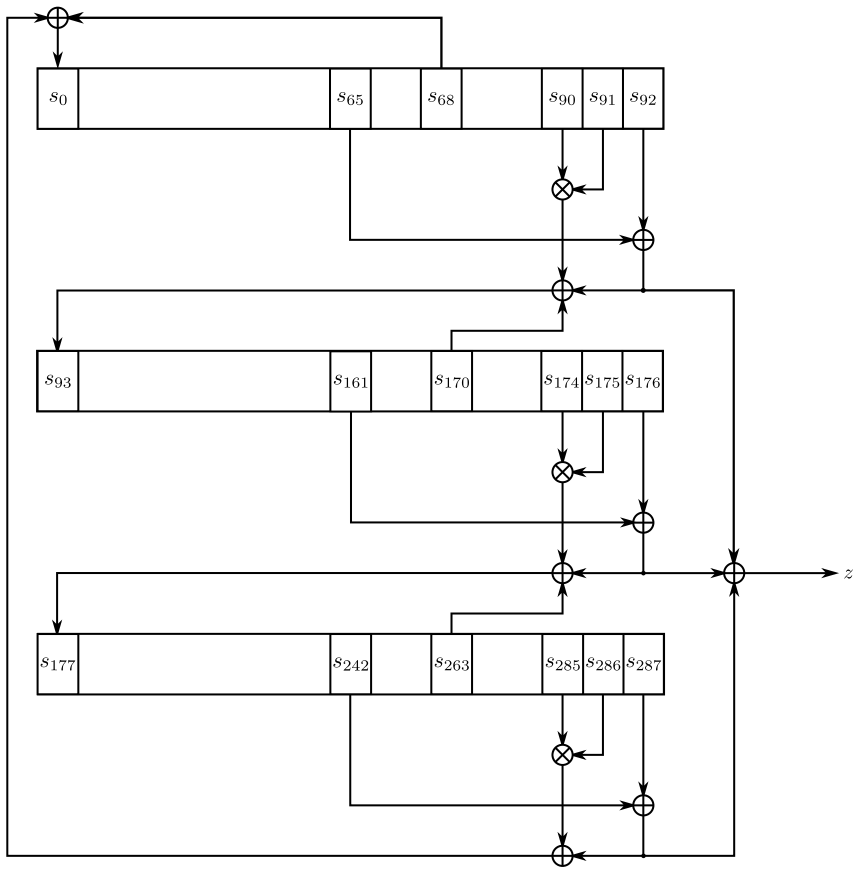


Figure 2: Schematic of the Trivium stream cipher consisting of 3 NLFSRs.

registers. As just described, Trivium uses NLFSRs, but other approaches have been considered in the literature, including filtering one or several LFSRs with a non-linear Boolean function [Can11c, Can11a] or using irregularly clocked generators [Fon11a, Fon11b]. Grain v1 [HJM07, HJMM06] is an example where both an LFSR with a non-linear filter function and an NLFSR are combined (and where the filtered LFSR part eventually became the weak point [TIM$^+$18]). Such approaches have been intensively analyzed, culminating with the eSTREAM competition [oEiCE04]. To the best of our knowledge no existing stream cipher design gets close to Trivium in terms of security vs. performance tradeoff in hardware, explaining our focus on this cipher. Note that a minimum requirement for all these designs is that the register is large enough to avoid trivial correlation attacks where the adversary can just guess the full register [Can11b]. Somewhat surprisingly, even this minimum requirement is not always met in practice. See for example the issue recently reported for the masked AES core of the OpenTitan project which uses multiple independent 32-bit filtered LFSRs (1 for 4 S-boxes each): `https://github.com/lowRISC/opentitan/issues/19091`. Hence, guessing the seed of each LFSR and testing the hypothesis is feasible.

Whether we could use sufficiently long LFSRs to avoid state guessing combined with a filtering function that is just strong enough to be secure in the leakage-only setting and to prevent the issues we put forward in Section 4 while not being a secure stream cipher is an interesting open question. Yet, it might not be worthwhile in the long run due to the limited potential for performance gains over stream ciphers combined with the increased security risk (too little non-linearity enables the attacks in Section 4; too small size and too little unrolling enable the attacks in Section 5). Additionally, it would annihilate the convenient possibility to consider the security of the PRNG used to generate randomness and the masking scheme itself independently that we also promote in this paper.

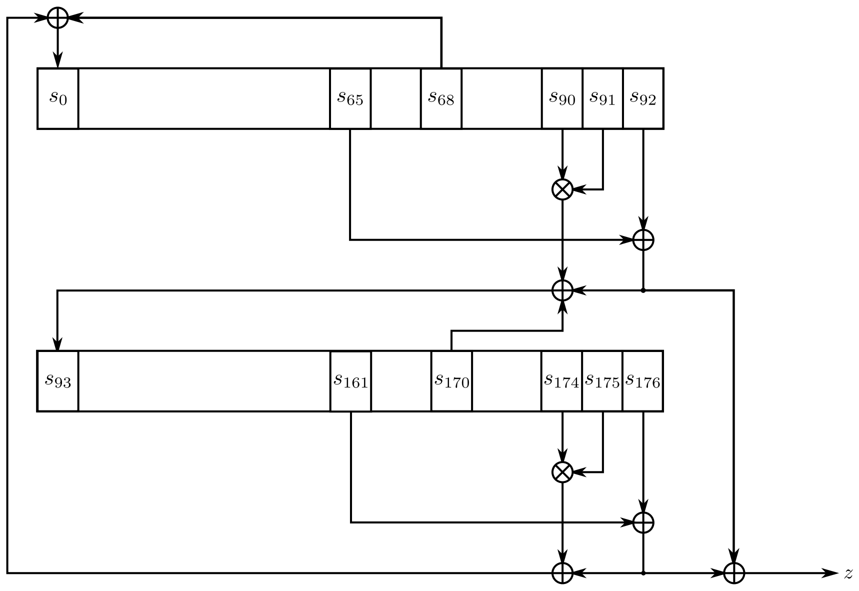


Figure 3: Schematic of Bivium B, a reduced security variant of the stream cipher Trivium consisting of 2 NLFSRs.

2.6 Unrolling Stream Ciphers

While our terminology of *unrolling* a stream cipher may appear somewhat novel, the general concept is not. It has merely been described using different terms in the past. Stream ciphers like Trivium and Grain have been developed with this implementation trick in mind in the early 2000s already. Similar to the iterated round functions of a block cipher or permutation, the (normally consecutively executed) state update (and output) functions of a stream cipher can be unrolled, which means multiple instances are realized in hardware without any memory elements in between so that the combinatorial logic of multiple steps is evaluated in a single cycle. See Figure 4(a) for an exemplary illustration of unrolling an update function. Here, each instance of the update function is producing 1 output bit or word per cycle (different from the block cipher analogy). While this is obviously a convenient instrument to adjust the tradeoff between latency (in cycles) and critical path (in seconds), there is another big advantage in the case of stream ciphers. Much unlike typical round functions of block ciphers and permutations, the state update functions of stream ciphers (typically) produce only 1 bit/word per cycle and can additionally be chosen to be sparse. Sparsity of the function means here that it receives only a few selected bits/words as inputs instead of the entire state. Thus, when choosing the function in a clever way, namely that the next update function does not receive the current feedback bit as input, it becomes possible to unroll a certain number of consecutive state updates (i.e., compute them in parallel in one cycle) without directly increasing the logic depth or critical path of the circuit. See Figure 4(b) for an example of a 2-bit unrolled Bivium implementation where the gate depth is not changed compared to the regular design in Figure 3. It turns out stream ciphers like Trivium and Grain [HJM07, HJMM06] indeed employ well-chosen functions to allow fast implementations generating multiple output bits and performing multiple updates per clock cycle. Trivium's update function ensures that any state bit which has just been modified is not used for at least the 64 following update steps [CP08]. For Grain v1 with 80-bit key, modified state bits are not used in the next 16 updates [HJM07] while for the 128-bit key variant that number is even increased to 32 [HJMM06]. Consequently, the critical path is only increased in steps of 64, 16 and 32 bits of unrolling, respectively. The area, however, is increased for each additional copy of the function. Yet, as only the combinatorial logic needs to be replicated and as the area of stream ciphers like Trivium and Grain is highly dominated by the state register(s), unrolling these primitives corresponds to a net gain in throughput-area ratio up to a certain level. Even beyond that level, the bits/cycle metric continues to grow much faster than the critical path of the implementation.

We acknowledge that our definition of *unrolling* has been mostly been referred to as *parallelization* in previous works on stream cipher implementations [GB08, GSB07, Rog07, LLL20]. We have decided to avoid this term in our work, as it conflicts with the description of multiple unrolled stream ciphers in parallel, which is a concept we need later in Section 5. Less commonly, unrolled stream cipher circuits have also been referred to as *higher-radix* implementations (e.g., radix-64 for 64-bit unrolled Trivium) [GLB+06, HCK+08].

3 Initial Cost Efficiency Comparison

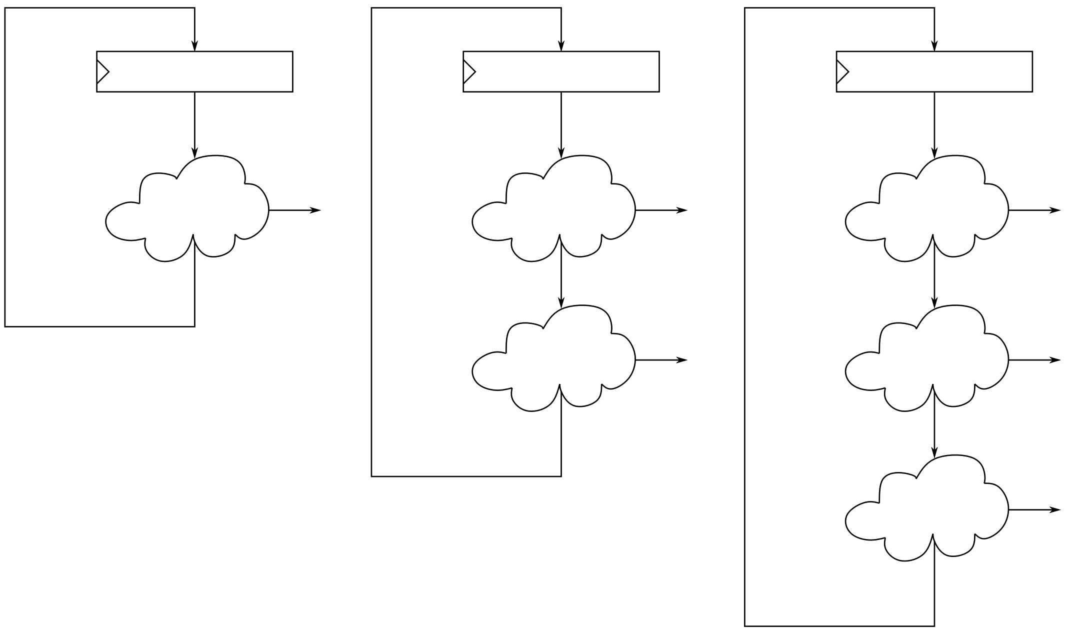
In the following we explore the suitability of multiple building blocks to be used as randomness generators for masked hardware implementations from the performance viewpoint.

3.1 Throughput-Area Ratio

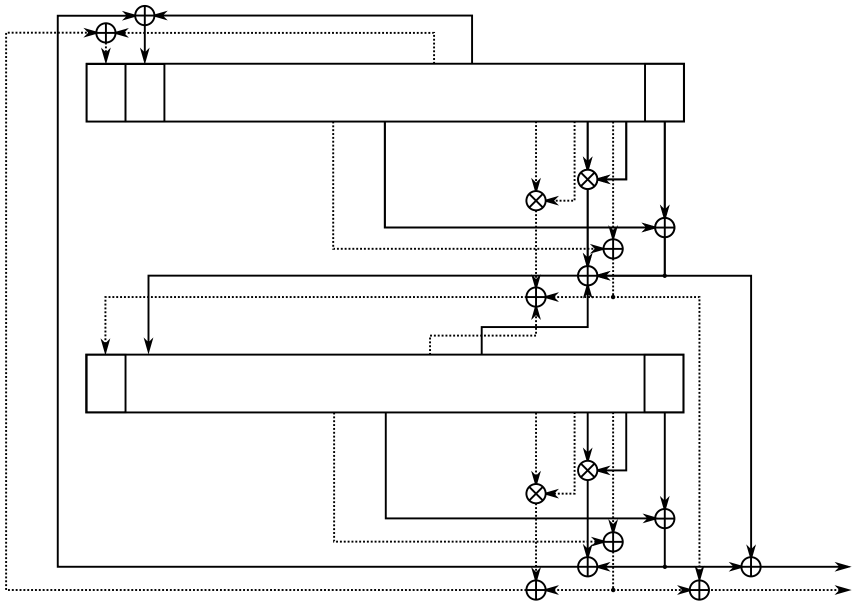
Comparing primitives for cost-efficient randomness generation begs the question which performance metric is the most relevant to evaluate. Since we aim to satisfy the randomness demands of even large, parallel, higher-order masked hardware implementations of cryptographic algorithms, we argue that the throughput-area ratio (TPA) is the optimal choice for our scenario (as opposed to the throughput, area or power consumption individually or any other combination of them). Assuming a masked implementation requires up to 1000 fresh random bits per cycle (not unusual for gadget-based parallel higher-order implementations, c.f. [KMMS22]) at 100 MHz in 65 nm ASIC technology, the TPA is arguably the most suitable metric to find the cheapest solution for generating the required amount of randomness per cycle at the desired clock frequency. For concrete settings, the throughput-area ratio measured in (bits/cycle)/GE is probably ideal for decision making. Yet, it is not best suited for general comparisons of multiple candidates due to its strong dependency on the chosen frequency and implementation technology. Indeed, to obtain both fair and optimal results in the (bits/cycle)/GE metric for any given candidate, a separate implementation needs to be constructed to best leverage the available critical path budget at each considered frequency and for each considered implementation technology. When measuring the throughput-area ratio in (bits/s)/GE instead, the critical path delay is part of the formula and it becomes easier to compare multiple candidates without limiting the analysis to one predefined setting. In fact, the (bits/s)/GE value can be seen as an upper bound on the TPA in the sense that the optimal value is obtained for a concrete circuit under the assumption that the operating frequency of the device is determined by this implementation, instead of the implementation being tailored to one predefined frequency. As not every candidate will fit into each setting perfectly, flexibility of an implementation is another important factor. In the following we use the throughput-area ratio measured in (bits/s)/GE to compare the selected candidates for concurrent randomness generation. We aim to avoid remaining technology biases by evaluating the metric in multiple standard cell libraries for the PRNG building blocks.

3.2 High-Throughput Low-Area TRNG

Since an efficient TRNG implementation is always needed when randomness has to be generated inside an integrated circuit, the first important question to answer is whether this primitive can potentially be used for all randomness generation, which would save the cost of implementing a PRNG in addition. To answer this question, we have selected a suitable representative from the TRNG literature and evaluate its efficiency in the following. At CHES 2018, Yang et al. proposed a high-throughput, low-area TRNG suitable for both ASIC and FPGA implementations [YRG+18]. The design is called ES-TRNG, where ES stands for Edge Sampling. The chosen noise source is timing jitter, and the design relies on two



(a) 1-bit/1-word (left), 2-bit/2-word (middle) or 3-bit/3-word unrolled update functions.



(b) 2-bit unrolled Bivium, producing keystream at twice the rate as regular Bivium.

Figure 4: Unrolling a stream cipher like Trivium/Bivium is achieved by implementing multiple consecutive update functions in the same hardware circuit, each producing one output bit/word per cycle. If one update function's inputs are independent of the feedback bits of multiple previous update functions, consecutive steps can be unrolled without increasing the circuit depth (e.g., up to 64 steps for Bivium/Trivium/Kreyvium).

techniques that the authors call variable-precision phase encoding and repetitive sampling to increase throughput and reduce area. We implemented this design using a 65 nm low-power CMOS standard cell library and obtained the results listed in Table 1. The throughput is estimated in part based on timing jitter measurements on a test chip manufactured under the same 65 nm technology and in part based on a prototype FPGA implementation whose output has been evaluated positively by statistical test suites. For a 100 MHz system clock the resulting throughput when synthesized in (and parametrized for) 65 nm ASIC technology is 2.2 Mbit/s and thus higher than the 1.15 Mbit/s given in [YRG+18]. This is expected when moving from FPGA to ASIC implementation. In fact, higher performances are likely still possible on ASIC platforms. Our concrete implementation is able to produce 7.4 kbit/s of high-entropy random numbers for each gate equivalent (GE) of area (i.e., a TPA of 7.4 (kbit/s)/GE). Please note that a recent improvement of the original ES-TRNG design has been published in [LBS22] under the name Tight-ES-TRNG. The authors performed low-cost optimizations to ensure that the signal edges populate a larger portion of the full distribution of phase jitter to increase the achievable entropy level. In that work, the throughput is increased to 5.6 Mbit/s. Even higher throughput may potentially be achieved with alternative TRNG designs according to the comparisons presented in [PMB+16, YRG+18], at the cost of a significantly larger area and less freedom in the implementation. In particular, the only TRNG design listed which provides a larger throughput-area ratio compared to the ES-TRNG is a Self-Timed Ring (STR) based TRNG [CFAF13, CFFA13], with an approximately 6 times better efficiency [PMB+16, YRG+18]. However, this design occupies a 20 to 30 times larger area (high default cost) and requires both manual placement and manual routing. Even when considering such a design, the throughput-area ratio would still fall in the range of tens, maybe hundreds, of kbit/s/GE for a 65 nm ASIC implementation, without yet considering the cost of monitoring the entropy source or continuous internal testing. We will demonstrate in this section that it is possible to implement cryptographically strong PRNGs with a much higher throughput-area ratio than that, providing strong support for the idea that randomness-hungry masked implementations are better served by PRNGs than TRNGs.

Table 1: ES-TRNG in a 65 nm low-power CMOS technology @ 100 MHz system clock.

	Min. Area	Min. Latency	Throughput @ 100 MHz
`ES-TRNG`	297 GE	0.135803 ns	2.2 Mbit/s

3.3 Permutations vs. Block Ciphers vs. Stream Ciphers

Having concluded negatively on the suitability of TRNG instances to generate random values for (parallel, higher-order) hardware masking, we now compare a number of potentially cost-efficient cryptographic building blocks based on their throughput-area ratio to be used in PRNG constructions. We rely on prior efficiency comparisons to pre-select such primitives. In more detail, the authors of [AZ21] compared the Round 2 candidates of the NIST lightweight cryptography standardization process [oSN17] based on their ASIC implementation figures. Several different metrics are evaluated, including throughput and throughput-area ratio. The lightweight primitive *Subterranean 2.0*, introduced in [DMMR20], offers by far the best throughput-area ratio, while the cross-platform permutation *Gimli*, proposed in [BKL+17], offers one of the best throughput figures. Both of these primitives are included in our further investigation. According to the comparison of low-latency ciphers given in [LMMR21], the 5-round version of the ultra low-latency cipher SPEEDY, called `SPEEDY-5-192` requires a smaller area per output bit and can be clocked at a higher frequency than any other low-latency primitive when implemented fully-unrolled in hardware. Finally, the comparison of all 8 stream ciphers in the hardware profile of the eSTREAM competition that reached the third phase are compared for their cost and efficiency in [GB08]. The stream cipher *Trivium* [Can06, CP08] offers by far the best throughput-area ratio among all its competitors, hence, we include it in our preliminary investigation as well. Other related works reached similar conclusions [GLB+06, BKSQ07, GSB07, Rog07, HCK+08, LLL20]

3.3.1 Full-Security Versions

To summarize, in our initial comparison we have selected two of the most cost-efficient building blocks from the NIST lightweight cryptography competition, the supposedly most cost-efficient low-latency cipher and the supposedly most cost-efficient hardware stream cipher from the eSTREAM competition. The corresponding synthesis results are given in Table 2 for 4 different ASIC standard cell libraries, 2 commercial ones and 2 open-source ones. The synthesis tool used is *Synopsys Design Compiler Version O-2018.06-SP4*. Our Subterranean 2.0, Gimli and SPEEDY hardware implementation results are based on publicly available source code that can be found here:

- Subterranean 2.0: `https://github.com/pmassolino/hw-subterranean`
- Gimli: `https://gimli.cr.yp.to/impl.html`
- SPEEDY: `https://github.com/Chair-for-Security-Engineering/SPEEDY`

The complete set of individual delay, area, power consumption and throughput figures, in addition to combined metrics such as energy consumption per bit and power-area-time product, are listed for each candidate in Appendix A. It is important to note that we compare the raw primitives in this initial comparison and do not consider any framework that is needed to turn them into usable PRNGs (typically required for the block-oriented primitives, but not for stream ciphers). For the block ciphers and permutations, two different versions are considered: (i) a fully-unrolled single-cycle implementation from combinatorial logic only and (ii) a fully-unrolled round-pipelined implementation (✓). For Gimli and SPEEDY, 24 and 5 cipher rounds are unrolled respectively. Both versions (i) and (ii) produce one block of output per clock cycle. Yet, the pipelined versions obviously require a number of cycles equal to the number of rounds before the first usable output is produced. We also acknowledge that efficiently initializing such large pipelines, for example $24 \cdot 384 = 9216$ bits in case of Gimli, while keeping the initial seed small, might become difficult. For Trivium the situation is different, as common stream ciphers are not based on iterative round functions. Instead, they are typically constructed from state update functions that produce a single output bit or word per step while updating the state register. As discussed in Section 2, these update functions can be unrolled in a similar manner as the round functions of a block cipher or permutation, with the additional advantage that state update functions can be sparse and chosen in such a way that a certain degree of unrolling leads to no increase of the gate depth or critical path. In fact, the structure of Trivium allows it to be implemented in a way that neither the depth nor the delay of the hardware circuit are increased for 64 bits of unrolling or below, making `Trivium_X64` the most cost efficient primitive in Table 2. `Trivium_X48` and `Trivium_X64` outperform `Trivium_X32`, `Trivium_72` and any larger (> 72) or lower ($<$ 32) unrolling level in fact. We evaluated Trivium for many degrees of unrolling (arbitrary degrees are possible, see also Section 5), but none of them offered a better throughput-area ratio than `Trivium_X64` in (bits/s)/GE. Of course, when integrated into a larger chip design where other components demand a lower operating frequency, larger unrolling factors are still more attractive in order to fully exhaust the critical path budget and generate as many bits per cycle as possible. Since the degree of unrolling can be chosen arbitrarily, the number of output bits produced per cycle is adjustable with single-bit granularity, which also provides a conveniently high flexibility compared to permutations and block ciphers.

Subterranean 2.0 is a hybrid primitive that is hard to put in a single category. It uses elements from permutation-based cryptography and resembles a sponge-like construction that behaves like a stream cipher when squeezed, producing 32-bit words per step. Its round/update function can also be unrolled to provide even more bits per cycle, but unlike Trivium, unrolling will not improve its TPA in (bits/s)/GE. Instead, the standard variant producing 32 bits per cycle is the most cost-efficient one in this metric and makes it the second most cost-efficient primitive in two of the four considered standard cell libraries in Table 2. Yet, its throughput-area ratio is 4 times lower compared to `Trivium_X64`, mostly

Table 2: Comparison of the throughput-area ratio of unrolled building blocks, including the stream cipher Trivium, the lightweight primitive Subterranean 2.0, the cross-platform permutation Gimli and the ultra low-latency cipher SPEEDY.

			Throughput/Area [(Mbit/s)/GE]			
			Commercial Foundry		NanGate OCL	
Primitive	pip.	bits/cycle	90 nm LP	65 nm LP	45 nm	15 nm
`Subterranean2`		32	14.272	17.848	15.610	156.644
`Subterranean2_X2`		64	9.430	11.418	10.461	92.776
`Subterranean2_X4`		128	5.422	6.786	6.987	60.899
`Subterranean2_X8`		256	2.910	3.309	3.859	32.578
`Gimli (24 rounds)`		384	1.254	1.540	1.739	16.351
	✓	384	13.518	15.875	16.561	162.556
`SPEEDY-5-192`		192	1.354	1.637	2.159	18.442
	✓	192	7.446	10.146	8.857	75.990
`Trivium`		1	2.356	4.072	2.189	22.232
`Trivium_X32`		32	51.292	46.261	46.827	412.244
`Trivium_X48`		48	48.623	63.186	57.639	607.553
`Trivium_X64`		64	58.899	73.785	77.514	752.875
`Trivium_X72`		72	42.484	59.694	61.534	530.410

because it occupies a larger area while only producing half the number of bits per cycle (c.f. Table 7). Its power consumption, however, is lower than unrolled Trivium's and almost as low as unrolled Bivium's (c.f. Table 8). Round-pipelined Gimli performs similarly well and places second in the other two cell libraries. Yet, it comes at a much higher default cost. Round-pipelined SPEEDY is roughly half as cost-efficient as Gimli and Subterranean 2.0 here.

Note that the throughput-area ratio of `Trivium_X64` is 77.51 (Mbit/s)/GE in NanGate 45 nm library, more than 13% larger than the 68.14 (Mbit/s)/GE achieved by the PRNG based on evolutionary programming presented in [PYR+16] using the same library. We believe it is a strong result that Trivium, a stream cipher with proven cryptographic strength that has been analyzed for the past 18 years, leads to more cost-efficient implementations in hardware than dedicated PRNGs for mask generation that only achieve sufficient statistical properties to pass common test suites without cryptographic guarantees.

3.3.2 Reduced-Security Versions

Yet, as explained in detail in Section 1, full 80-bit or 128-bit security may not always be needed for PRNGs in masking contexts, as adversaries may at most obtain a noisy version of the PRNG's output through side-channel observations. Performing cryptanalytic attacks or solving complex systems of equations based on partially erroneous data is known to be a hard problem. The security of modern lattice-based post-quantum cryptography for instance depends on the hardness of computational problems such as Learning With Errors (LWE), which requires solving a system of noisy linear equations. Hence, lower security levels in the black-box model can lead to much higher security levels against adversaries with access to noisy data only [DFH+16]. For this reason our concrete setting might tolerate lower security levels if the cost-efficiency can be improved significantly. In this context, the advantage of block ciphers and permutations over stream ciphers is that the number of rounds can be adapted in order to flexibly adjust the security-vs-performance tradeoff. In fact, it is possible for most modern block ciphers and permutations to remove multiple rounds and still maintain a security level beyond enumeration power. For stream ciphers, no such well-understood mechanism exists (shortening the initialization phase reduces security but brings no gains in hardware performance during keystream generation). Luckily, in case of Trivium, the reduced security variant Bivium has been introduced as a study object for cryptanalytic analyses. Hence, we also compare Bivium to round-reduced versions of the previously analyzed building blocks Gimli and SPEEDY in Table 3. The smallest round-reduced versions (8-round Gimli and 2-round SPEEDY) chosen here can both be practically broken in the black-box setting (distinguishers with complexities below 2^{40} exist), but attacks are still expected to require a computational effort that becomes prohibitive when only partial information on inputs and outputs is available. `Bivium_X64` clearly outperforms the reduced security primitives in TPA. Here, round-pipelined 8-round Gimli performs best among the remaining primitives, with approximately half the throughput-area ratio compared to `Bivium_X64`, while also consuming a 9 times larger area and 10 times more power on average. Reduced-round SPEEDY achieves less than a quarter of the cost efficiency of `Bivium_X64`. The results in Table 2 and Table 3 show that Trivium and Bivium are the most suitable among the tested primitives.

Table 3: Comparison of the throughput-area ratio of the reduced-security versions of the primitives from Table 2.

			Throughput/Area [(Mbit/s)/GE]			
			Commercial Foundry		NanGate OCL	
Primitive	pip.	bits/cycle	90 nm LP	65 nm LP	45 nm	15 nm
`Gimli (8 rounds)`		384	8.619	10.544	12.839	108.180
	✓	384	42.754	44.256	49.251	480.270
`Gimli (16 rounds)`		384	2.484	2.996	4.273	36.684
	✓	384	20.603	25.763	23.827	243.939
`SPEEDY-2-192`		192	10.429	12.539	15.224	134.877
	✓	192	19.734	25.811	24.427	242.620
`SPEEDY-3-192`		192	4.155	4.918	6.359	55.208
	✓	192	12.513	16.790	15.249	134.630
`SPEEDY-4-192`		192	2.183	2.751	3.425	29.738
	✓	192	9.297	12.971	11.013	96.851
`Bivium`		1	3.624	5.364	3.544	34.050
`Bivium_X32`		32	70.893	77.847	71.559	690.677
`Bivium_X48`		48	75.572	96.861	107.462	969.275
`Bivium_X64`		64	89.620	107.543	109.730	1212.312
`Bivium_X72`		72	66.909	90.785	81.547	809.863

3.4 Other Stream Ciphers

The observation that stream ciphers can outperform block-based encryption algorithms in throughput-area ratio is not new. Stream ciphers are known to require a smaller area footprint in hardware (on average) compared to block ciphers with similar security levels and have therefore been of primary interest for resource-constrained devices such as smart cards, sensor networks or Radio-Frequency Identification (RFID) tags [BKSQ07]. Since Trivium delivered very promising results in the preliminary analysis, we now investigate whether other stream ciphers can provide similarly impressive performance for randomness generation.

The eSTREAM competition was held from 2004 to 2008 with the goal to identify and collect secure and compact stream cipher proposals suitable for widespread adoption; separately for a hardware and software profile [oEiCE04]. During this time, researchers have compared many of the proposed stream ciphers to each other and also to older standards such as A5/1 [BGW98], RC4 [Sch96] or E0 [E02], resulting in a number of publications containing performance rankings of multiple candidates [GB08, GLB+06, BKSQ07, GSB07, Rog07, HCK+08]. They all have in common that Trivium is identified as the number one candidate with respect to throughput-area ratio, although among different sets of primitives, most oftenly followed by Grain v1 (either the 80- or 128-bit key variant). Also for the maximum throughput, Trivium places first in all works that have implemented its 64-bit unrolled variant [GB08, GLB+06, GSB07, Rog07, HCK+08]. The top rank for minimum area is mostly split between Trivium, Grain v1 [HJM07, HJMM06], MICKEY 2.0 [BD08] when considering only eSTREAM candidates - with the insecure A5/1 algorithm (broken in practical time since the year 2000 [BD00]) being consistently smaller. Two further comparisons have been published several years after the eSTREAM competition ended, namely [KSPS13] in 2013 and [LLL20] in 2020, which also include more recent stream cipher proposals. While the work by Kitsos et al. favors MICKEY 2.0 over Trivium or Grain v1 for maximizing the throughput-area ratio, this is mostly due to the fact that only the basic non-unrolled versions of the latter have been implemented [KSPS13]. This is also criticized by Li et al. [LLL20], who provide implementation figures including unrolled Trivium and Grain v1 and report an advantage of Trivium over MICKEY 2.0 in TPA by more than an order of magnitude. In total, all listed publications collectively include the following set of stream ciphers (or variants of them) implemented in hardware: A5/1 [BGW98], RC4 [Sch96], E0 [E02], SNOW3G [SNO], Phelix [WSLM05], Lex [Bir08], Achterbahn [GGK05], MOSQUITO [DK05], SFINKS [BLM+05], VEST [OGL05], ZK-Crypt [GGV05], Trivium [CP08], Grain v1 [HJM07, HJMM06], MICKEY 2.0 [BD08], DECIM [BBC+08], Edon80 [GMK08], F-FCSR [ABL08], Moustique [DK08], Pomaranch [JHK08], Salsa20 [Ber08], ZUC [ZUC], Plantlet [MAM16] and Lizard [HKM17]. Among all these primitives, the candidates consistently performing best in maximum throughput, minimum area and maximum throughput-area ratio (without being broken in practical time complexity yet) are Trivium, Grain v1 (80- and 128-bit variant) and MICKEY 2.0 (80- and 128-bit variant). These are, not surprisingly, also the three candidates that have been selected for the hardware portfolio of the eSTREAM competition. Hence, we decided to select these three primitives for a closer look at stream cipher performances for mask generation. While, to the best of our knowledge, no cryptanalytic attacks against full Trivium or full MICKEY 2.0 exist, key recovery attacks against both Grain v1 variants (80- and 128-bit key) with complexities below exhaustive key search are known [TIM+18, BCM23]. However, the complexities remain significant enough to not disregard these primitives for our purposes, as the performance of the attacks is still prohibitive for most computationally bounded adversaries even in the noise-free setting (approximately 2^{75} and 2^{112} respectively). Since we take a look at 80- and 128-bit variants of both Grain and MICKEY, we also consider a 128-bit secure version of Trivium for this exercise, which is called Kreyvium and has been first proposed at FSE 2016 for efficient homomorphic encryption [CCF+16]. Unlike the 128-bit versions of Grain and MICKEY, Kreyvium is not a real re-design of its successor based on larger parameters, but mostly relies on keeping original Trivium intact and extending it by two additional registers holding the 128-bit key and 128-bit IV, which are never updated beyond a rotation of the bits. This change, while promising for the purposes intended by the authors, is not exactly optimal to improve the throughput-area ratio.

The full comparison is shown in Table 4; individual delay, area, power consumption and throughput figures in addition to combined metrics (energy consumption per bit, power-area-time product) can be found in Appendix A. As already pointed out in previous stream cipher comparisons, the optimal level of unrolling to maximize the throughput-area ratio differs between the candidates. For Grain v1 with 80-bit key and 128-bit key it is 16 and 32 respectively, for Trivium and Kreyvium it is 64 (because the core design is identical) and for MICKEY 2.0 with 80-bit key and 128-bit key it is actually 1. The MICKEY design is not based on the principle that recently updated state bits are not used in the next X update steps. Hence, unrolling this stream cipher will not lead to any improved results in the (bits/s)/GE metric (doubling the output bits per cycle will also increase both delay and area of the combinational logic approximately by factor 2, resulting in a lower TPA overall). To still have a common denominator between all designs for comparison purposes, we made sure to evaluate a 32-bit unrolled version of each of them. While the small size and power consumption of Grain v1 is attractive (c.f., Table 7; similar to Bivium), its update function has a larger gate depth than Trivium's and the number of consecutive update functions based on independent bits is too small to reach the throughput-area of Trivium. In fact, Subterranean 2.0 and round-pipelined Gimli achieve better TPA than both Grain v1 variants (c.f., Table 2). MICKEY 2.0 is only competitive when no unrolling is considered, making it a poor choice for our purposes. Additionally, it has received the least amount of cryptanalysis among the three eSTREAM candidates considered here and it has been pointed out that its data-dependent irregular clocking may lead to simple, timing-based side-channel attacks [GBC+08], which is highly undesirable for our setting. Finally, we extrapolated the critical path values given in Appendix A Table 6 to estimate how many bits per *cycle* can at most be generated by each of the stream ciphers (i.e., the maximum level of unrolling) in 65 nm ASIC technology at 100MHz operating frequency. The results show that Bivium, Trivium and Kreyvium may generate over 3000 bits per cycle, while the Grain v1 variants achieve 400 (for the 80-bit key) and 1000 bits (for the 128-bit key) respectively. Both MICKEY 2.0 versions may generate at most 50 bits per cycle. Once again, we conclude that Trivium and its variants are clearly the most promising cryptographic algorithms for cost efficient and secure randomness generation in hardware.

Table 4: Comparison of the throughput-area ratio of unrolled stream ciphers.

			Throughput/Area [(Mbit/s)/GE]			
			Commercial Foundry		NanGate OCL	
Primitive	key length	bits/cycle	90 nm LP	65 nm LP	45 nm	15 nm
`GrainV1_80`	80	1	1.464	2.020	1.580	14.825
`GrainV1_80_X8`		8	6.391	7.849	7.287	69.252
`GrainV1_80_X16`*		16	8.274	10.861	9.031	86.915
`GrainV1_80_X32`		32	5.817	7.023	7.158	61.417
`GrainV1_128`	128	1	1.086	1.484	1.093	9.841
`GrainV1_128_X16`		16	8.522	11.465	9.751	91.195
`GrainV1_128_X32`*		32	12.410	18.556	13.996	130.803
`GrainV1_128_X48`		48	8.907	11.978	11.195	94.795
`MICKEY2_80`*	80	1	1.224	1.807	1.391	11.291
`MICKEY2_80_X2`		2	0.967	1.440	1.108	9.191
`MICKEY2_80_X4`		4	0.621	0.835	0.763	6.627
`MICKEY2_80_X32`		32	0.056	0.077	0.068	0.642
`MICKEY2_128`*	128	1	0.778	0.967	0.885	6.247
`MICKEY2_128_X2`		2	0.575	0.817	0.652	5.602
`MICKEY2_128_X4`		4	0.360	0.496	0.443	3.603
`MICKEY2_128_X32`		32	0.033	0.046	0.039	0.359
`Trivium`	80	1	2.356	4.072	2.189	22.232
`Trivium_X32`		32	51.292	46.261	46.827	412.244
`Trivium_X48`		48	48.623	63.186	57.639	607.553
`Trivium_X64`*		64	58.899	73.785	77.514	752.875
`Trivium_X72`		72	42.484	59.694	61.534	530.410
`Kreyvium`	128	1	0.982	1.240	1.027	9.218
`Kreyvium_X32`		32	21.554	27.449	23.696	233.767
`Kreyvium_X48`		48	29.554	36.597	33.600	325.753
`Kreyvium_X64`*		64	37.142	45.264	37.719	388.195
`Kreyvium_X72`		72	27.773	38.244	32.134	291.893

* Optimal level of unrolling to maximize Throughput/Area ratio.

3.5 LFSRs

There is one well-known primitive for generating long sequences of random-looking bits that has not been discussed yet and which is preferable over Trivium and Bivium from a pure performance standpoint, namely an unrolled LFSR. Indeed, the update function of an LFSR can be unrolled in the same manner as for stream ciphers like Trivium or Grain. Additionally, sparse feedback polynomials which guarantee maximum period are known for (almost) arbitrary register sizes. Of course, a sufficiently large state is required (e.g., > 64 bits) to (i) guarantee a long enough non-repetitive output sequence and (ii) keep adversaries from simply guessing all state bits. While this requirement seems rather obvious, it has apparently been neglected in the design of OpenTitan, the open source silicon root of trust developed by lowRISC, where multiple 32-bit LFSRs are instantiated to generate masking randomness.[4] To attack this design, one may simply enumerate all possible states of the LFSRs to find the correct seed for each of them separately (since there is no diffusion between the LFSRs) which in turns allows one to generate all previous and future masks.

However, even when respecting the minimum size of 64 bits, an unrolled LFSR can easily be 3-4 times more cost-efficient than Bivium, which is demonstrated for different levels of unrolling in Table 5. The obvious caveat is that LFSRs provide no black-box security whatsoever, as all output bits depend linearly on the initial state. As discussed in Section 2 already, this allows trivial state recovery attacks when sufficiently many consecutive output bits are observed. Another problem may occur when multiple bits with linear dependencies between each other are used as pseudo-randomness in the same masked implementation. Hence, a closer look at the severity of such risks is needed, which we study next.

Table 5: Throughput-area ratio of an exemplary 64-bit LFSR for different degrees of unrolling.

		Throughput/Area [(Mbit/s)/GE]			
		Commercial Foundry		NanGate OCL	
Primitive	bits/cycle	90 nm LP	65 nm LP	45 nm	15 nm
`LFSR64`	1	10.805	12.984	11.741	106.251
`LFSR64_X32`	32	230.191	265.875	218.516	2384.159
`LFSR64_X64`	64	222.286	298.575	308.312	3058.591
`LFSR64_X96`	96	294.813	399.138	387.573	3310.856
`LFSR64_X128`	128	287.444	373.509	332.251	2821.053

4 Security Analysis: Unrolled LFSRs

In this section, we tackle the question of the (in)security of using LFSRs as the source of masking randomness. At first sight, although LFSRs do not have as strong properties as other cryptographic primitives (i.e., their state can be retrieved by observing the stream it generates), we would like to point out again that their use as a source of randomness in the context of masking may make sense. Indeed, masking schemes usually only require that their randomness is uniformly distributed and unpredictable. Since the adversary does not have access to the output of the LFSR, but only a noisy version of it, state recovery might not be easy, and we will optimistically assume that the unpredictability holds true. Regarding the uniformity, a properly seeded LFSR has uniform output bits, but they are not independent. Are these dependencies an issue in practice? We answer this question positively with two simple examples where the masking security is broken due to the linear dependencies in the LFSR output stream, both in theory and in real-world experiments.

At its core, the problem is simple. Let a set of values $x_1, \ldots, x_n$ represented by the first-order Boolean sharings $(x_1^0, x_1^1), \ldots, (x_n^0, x_n^1)$ that are XORed together: $x^i = x_1^i \oplus \cdots \oplus x_n^i$ for $i = 0, 1$.[5] If the sharings are freshly generated from randomness r_j out of the LFSR (i.e.,

[4] See `https://github.com/lowRISC/opentitan/blob/master/hw/ip/aes/rtl/aes_prng_masking.sv#L7` and the already mentioned issue `https://github.com/lowRISC/opentitan/issues/19091`

[5] This problem also exists for higher-order masking. We focus on the simplest (first-order) example.

$x_j^1 = r_j$ and $x_j^0 = x_j \oplus r_j$), and moreover if these bits are linearly dependent ($r_1 \oplus \cdots \oplus r_n = 0$), then $(x^0, x^1) = (x, 0)$: the masking is completely removed.

Let us now relax the fresh sharing generation hypothesis, which might not be realistic. Instead, let us consider a case where the sharings (x_j^0, x_j^1) are the outputs of AND gadgets, such as the well-known ISW multiplication [ISW03]. In this case, $x_j^i = y_j^i z_j^0 \oplus y_j^i z_j^1 \oplus r_j$ for $i = 0, 1$, assuming that the multiplication inputs are sharings of y_j and z_j. Since this value can be re-written as $x_j^i = y_j^i z_j \oplus r_j$, we can clearly see that, assuming again a linear dependency of the r_js, the result of XORing the x_js is an insecure sharing:

$$(x^0, x^1) = \left(\bigoplus_{j=1}^{n} x_j^0 z_j, \bigoplus_{j=1}^{n} x_j^1 z_j \right).$$

For example, if the z_js are all 0, then $x^0 = 0$, while if they are all 1, it is uniformly distributed.

Let us now show how this problem happens in concrete cases. Both case studies are based on a circuit masked at the first order using the HPC2 masking scheme [CGLS21] (i.e., we have HPC2 AND gadgets and sharewise XORs), and we ensure that the circuits are secure against glitches and transitions [CS21] when provided with fresh randomness. The first case-study is a low-latency implementation of Ascon [DEMS20] (the recent winner of NIST's Lightweight Cryptography standardization process [oSN17]) showing that the above problem can appear in real-world circuits and not only in artificial examples. Next, the second case study shows how glitches and transitions interact with the LFSR, providing further insights on how (not) to use an LFSR for generating masking randomness.

4.1 Case Study 1

We implemented the masked Ascon permutation with a round-based architecture and with minimal latency (2 clock cycles per round, due to the latency of the HPC2 AND gadgets). The permutation operates on a 320-bit state that can be represented as binary matrix of dimension 5x64. Next, we use $b^{u,v} = b_{0 \leq 5v+u < 320}$ to depict the bit of the state located at the u-th row and the v-th column.[6] The permutation is composed of the layers shown in Figure 5: the substitution layer consists in 64 parallel S-boxes (each operating on a single column) and the linear diffusion layer operates on the rows. As depicted in Figure 6 and Figure 7, the non-linear part of the S-box (which is the Keccak S-box) is the only sequential logic, and the other layers (S-box output, Ascon diffusion layer, and S-box input) are all linear and implemented with combinational logic. The non-linear S-box layer is implemented as a two-stage pipeline that is fed with the fresh masks $r_{i,0 \leq i < 320}$. It outputs the sharing of the state bits $s_{i,0 \leq i < 320}$ (which are simply a forwarding of its input) and the results of the AND gates denoted $a_{i,0 \leq i < 320}$. The others layers combined together form a purely combinational XOR network producing the sharing for the bits $s'_{i,0 \leq i < 320}$that are then forwarded back to the input of the substitution layer. The values of the fresh masks r_i are obtained by applying 320 times the update function of the instantiated n-bit LFSR and can thus directly be expressed as a linear combination of the previous LFSR state bits denoted $l_{r,0 \leq r < n}$.

By tracking back the operations from the output of the combined linear layer to its input (i.e., the output of the substitution layer), the values of the bits s'_i can be expressed as a linear combination of some specific s_i and a_i only. Furthermore, the sharings of the bits a_i are linearly dependent on the values of the fresh masks r_i, which means that every s'_i is the result of a combination involving multiple r_i. When using an n-bit unrolled LFSR with $n < 320$ to generate the r_is, some of these bits become linearly dependent, which we can make evident by writing the r_is as a linear combination of the initial LFSR state l_r.

It follows that, if for a wire in the circuit, the combination of the involved r_i results in the canceling of the randomness (i.e., the involved l_r bits cancel out to 0), the value

[6] We order bits by column in the state, from the left to the right and from the top to the bottom position inside a column. That is, b_0 (resp., b_{319}) is the top left (resp., bottom right) bit in the state matrix.

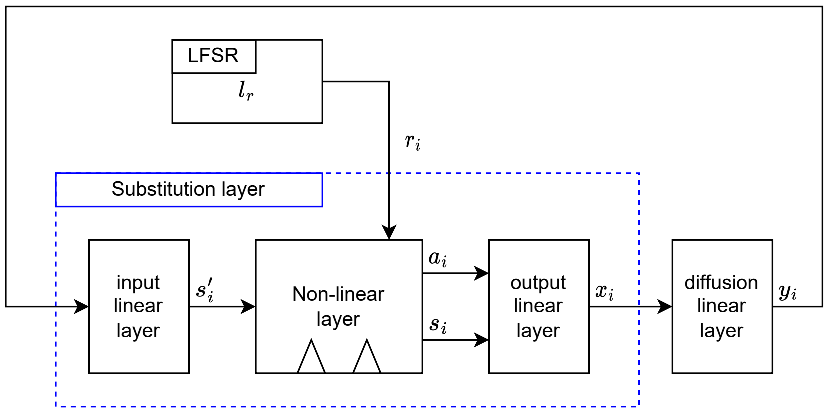


Figure 5: Top-level representation of the Ascon permutation.

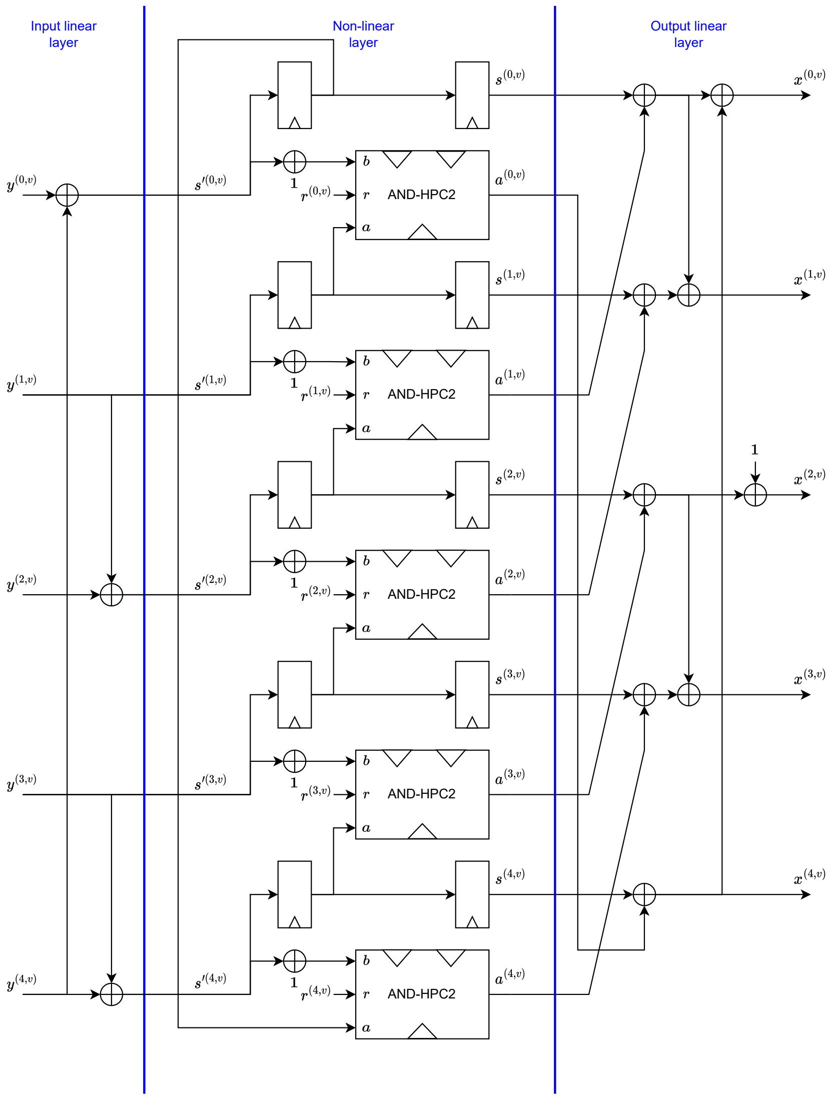


Figure 6: Masked Ascon S-box architecture.

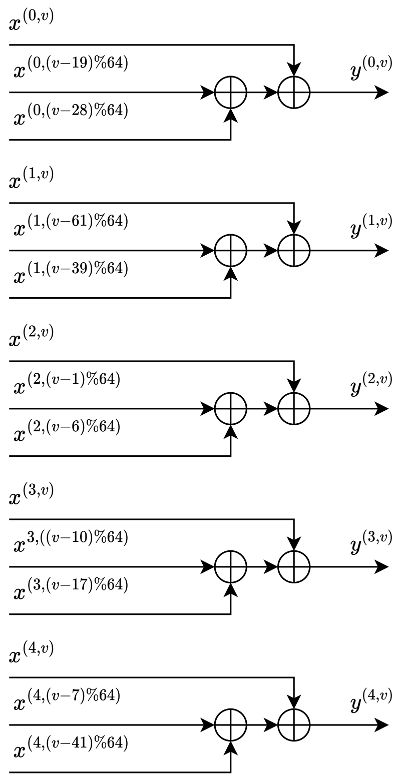


Figure 7: Masked Ascon diffusion layer architecture.

on the wire may become dependent on the secret values. Such a canceling may occur explicitly (i.e., a physical wire in the implementation is designed to hold this value) or may be induced by physical default such as glitches (i.e., this value appears only ephemerally due to propagation delays). Both cases may occur in practice. Whether they occur or not is out of the designer's control when using a standard tool flow where automated algorithms determine the concrete gate-level implementation and adjust wire/path delays as needed to satisfy timing constraints.[7] For the sake of simplicity, we discuss the explicit randomness cancellation in the following discussion. More precisely, we analyzed the maximum-length LFSRs proposed in [Alf96]. By symbolically simulating the execution of our Ascon implementation when using these LFSRs as PRNG, we identified that 13 designs (out of 166 possible ones) had randomness cancellations. We next detail the results we obtained for the 63-bit LFSR. For simplicity, we only analyze the results for an implementation with 2 shares, but our methodology could be applied identically when more shares are considered.

We first illustrate how randomness cancellation may occur during the computation using the 63-bit LFSR. In particular, we identified 39 randomness cancellations in the linear tree and describe next a concrete occurrence for the bit $s'_{32} = s'^{(2,6)}$ as an example. From Figure 5 and Figure 7, $s'^{(2,6)}$ can be expressed as follows:

[7] While the designer may add synthesis constraints in order to prevent the first case, avoiding glitches requires the addition of registers, which in turn increases the latency of the implementation.

$$
\begin{aligned}
s'^{(2,6)} &= y^{(2,6)} \oplus y^{(1,6)} && (1)\\
&= x^{(2,6)} \oplus x^{(2,5)} \oplus x^{(2,0)} \oplus x^{(1,6)} \oplus x^{(1,9)} \oplus x^{(1,31)} && (2)\\
&= 1 \oplus s^{(2,6)} \oplus a^{(3,6)} \oplus 1 \oplus s^{(2,5)} \oplus a^{(3,5)} \oplus 1 \oplus s^{(2,0)} \oplus a^{(3,0)} \\
&\oplus s^{(1,6)} \oplus s^{(0,6)} \oplus a^{(2,6)} \oplus a^{(1,6)} \oplus s^{(1,9)} \oplus s^{(0,9)} \oplus a^{(2,9)} \oplus a^{(1,9)} \\
&\oplus s^{(1,31)} \oplus s^{(0,31)} \oplus a^{(2,31)} \oplus a^{(1,31)} && (3)\\
s'_{32} &= s_{32} \oplus s_{27} \oplus s_2 \oplus s_{31} \oplus s_{30} \oplus s_{46} \oplus s_{45} \oplus s_{156} \oplus s_{155} \\
&\oplus a_{33} \oplus a_{28} \oplus a_3 \oplus a_{32} \oplus a_{31} \oplus a_{47} \oplus a_{46} \oplus a_{157} \oplus a_{156} && (4)
\end{aligned}
$$

where Equation (1) is the expansion of the S-box input linear layer, Equation (2) is the expansion of the diffusion linear layer, Equation (3) is the expansion of the S-box output linear layer and Equation (4) removes the constants and applies the mapping to state indices instead of matrix locations. The constants are removed since we are only interested in expressing the linear dependencies on state variables. To ease the expressions, we next define the set $\mathcal{I} = [32, 27, 2, 31, 30, 46, 45, 156, 155]$ as the set of state bit indices involved in Equation (4). By expanding the results of the AND operations, the following expression holds for $s'^{0 \leq i < 2}_{32}$, the i-th share of the value s'_{32}:

$$
\begin{aligned}
s'^{i}_{32} &= \bigoplus_{j \in \mathcal{I}} (s^i_j \oplus a^i_{j+1}) && (5)\\
&= \bigoplus_{j \in \mathcal{I}} (s^i_j \oplus (s^i_{j+2} \odot \overline{s_{j+1}}) \oplus r_{j+1}) && (6)
\end{aligned}
$$

where Equation (5) is a variant of Equation (4) to express the shares instead of the unmasked value and Equation (6) is the expansion of the multiplication gadgets. The latter depicts an expression very similar to the problematic case explained at the beginning of the section: a randomness cancellation may lead to the sharing (s'^0_{32}, s'^1_{32}) being insecure. Such a cancellation can be searched by relying on symbolic executions of the LFSR update function in order to express the r_i as a linear combination of the LFSR state bits. Considering the previously defined set $\mathcal{I}$, it turns out that $r^j_j \in [31, 33, 157]$ are linearly dependent.

To check the validity of our analysis, we ran simulations of our implementation, computing the share distribution of the 39 bits for which randomness cancellations are expected to occur. We used a netlist in which the additions are ordered such that the computations of the randomness cancellations are explicit (while keeping the exact same functionality). In order to easily highlight the bias, we used a fixed input state sharing and a uniformly generated PRNG seed for each simulation case. As depicted in Figure 8, the conditional distribution of the sharing is significantly biased for the identified failing path (depicted in blue, for the biased shares b^0 and b^1) while it shows a uniform distribution for the cases where no failing paths were detected (depicted in red for the unbiased shares u^0 and u^1).

We conclude our analysis by performing an experimental verification of the discovered issue on a SAKURA-G board using a Spartan-6 FPGA as target. To do so, we performed a power analysis by collecting measurements at 500MS/s using a Picoscope 5224D digital oscilloscope and by measuring the voltage drop across a 1 Ohm shunt resistor, with a vertical resolution of 12bits. The FPGA was fed with an external clock of 2.5MHz synchronized with the clock of the oscilloscope to ensure a proper temporal alignment of the traces. With this configuration, we performed a fixed-vs-fixed statistical t-test using 10M traces to evaluate the statistical security order achieved by our securely first-order masked Ascon implementation. Figure 9 shows that first-order leakage is present and confidently detectable with less than 10M traces. In order to verify that the source of this leakage is indeed the use of the unrolled 63-bit LFSR for fresh mask generation we have repeated the same experiment with

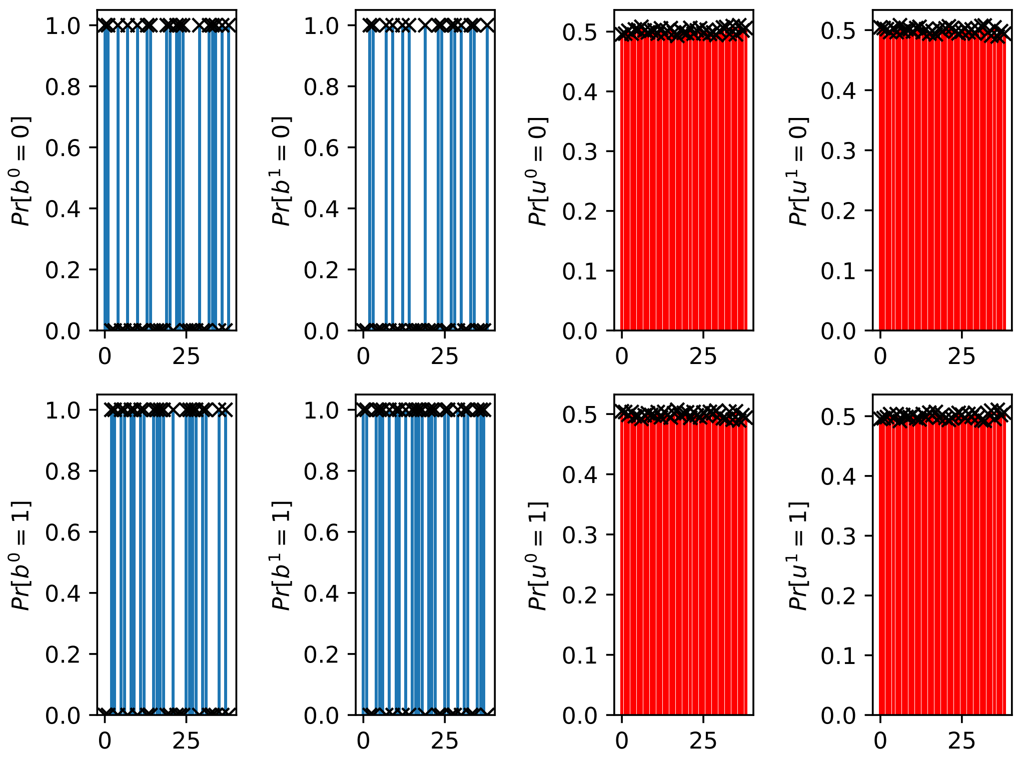
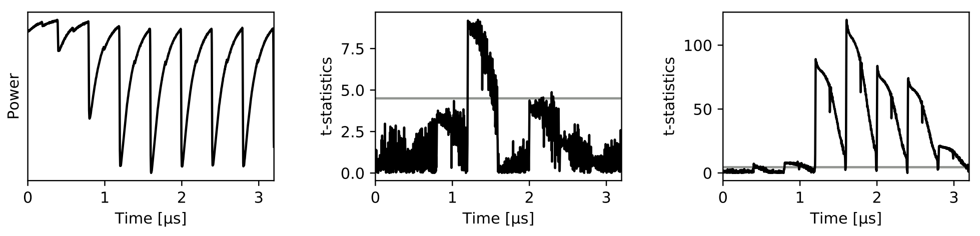
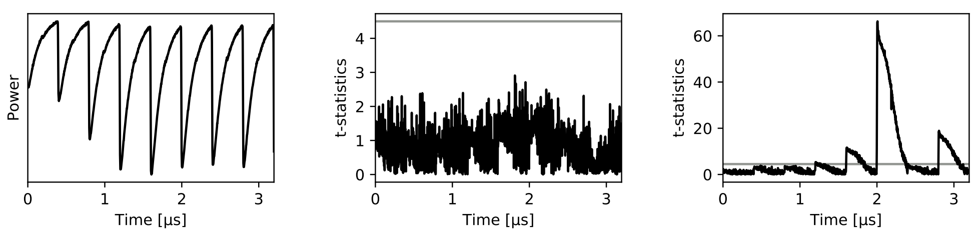


Figure 8: Conditional distribution of the sharing.



(a) Sample trace (b) first order t-test (c) second order t-test

Figure 9: Case study 1: fixed-vs-fixed t-test results for the first-order masked Ascon experiment (10M traces) using the unrolled 63-bit LFSR as source of fresh randomness.



(a) Sample trace (b) first order t-test (c) second order t-test

Figure 10: Case study 1: fixed-vs-fixed t-test results for the first-order masked Ascon experiment (10M traces) using unrolled Trivium as source of fresh randomness.

an unrolled Trivium instead. The result is depicted in Figure 10, showing a clear absence of first-order leakage (only the expected second-order leakage is present and has a lower amplitude). It confirms that using multiple bits from the same LFSR as random values for a masked implementation can lead to a security degradation in realistic settings.

4.2 Case Study 2

As a second case study, we consider the case of a network of AND and XOR operations. More precisely, we suppose that the underlying circuit receives two n-bit operands $x_1, \dots, x_n$ and $y_1, \dots, y_n$ and generates the output as $z = \bigoplus_{i=1}^{n} x_i y_i$. Considering an n-bit LFSR, we have seen in the first case study what happens when unrolling more than n rounds. In this case study, we demonstrate that even unrolling less than that can cause issues, due to transitions. More precisely, the observation is that such an n-bit LFSR should not be unrolled more than $n/2$ rounds.[8] Otherwise, the security of the circuit under robust probing model [FGP+18] cannot be guaranteed. This can be justified by the fact that transition-extended probes would allow the attacker to observe two consecutive values stored in a register. Without considering transitions, it should be possible to securely use an n-bit LFSR with n unrolled rounds (again considering 1 execution cycle). In order to practically show this, we set two examples with $n = 4$ and $n = 5$ (identified as `ANDXOR4` and `ANDXOR5`), while using an unrolled 8-bit LFSR generating 4 (resp., 5) random bits per clock cycle (next referred to as `LFSR8_X4` and `LFSR8_X5`).

Verification Tool. To examine the security of these circuits, we conducted two distinct experiments. We first synthesized the circuits for an ASIC platform by Synopsys Design Compiler with NanGate 45 nm digital library and used `PROLEAD` [MM22] to examine their first-order probing security under glitch- and transition-extended probing model. As a side

[8] If we unroll less than $n/2$, we do not expect to see problem arising when considering only one execution cycle, however there might still be issues when executing the implementation over multiple cycles.

note, `PROLEAD` is a simulation-based leakage assessment software tool developed for evaluation of masked hardware designs following the robust probing security model. We analyzed both circuits by `PROLEAD` using 100M simulations and allowed first-order probes to be placed on every location of the circuit (including the LFSR) and extended based on glitches as well as transitions. We set `PROLEAD` to seed the LFSR with a randomly generated 8-bit value for every simulation, and performed a fixed-vs-random statistical test with fixed input $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ being all zero. While the tool did not report any leakage for $n = 4$, it revealed a single probe placed on an output share (e.g., z^0) being able to see significantly different distributions for $n = 5$ using less than 1M simulations. Note that we have similarly examined smaller circuits with $n < 4$ which led to the same result as with $n = 4$. Note also that transition-extended probes play an important role here. Without covering transitional leakages, `PROLEAD` did not report any first-order leaking probes up to $n = 8$.

FPGA-Based Experiments. As in the previous section, we also have carried out FPGA-based experimental analysis on both aforementioned circuits. To this end, we made use of a SAKURA-G board with Spartan-6 FPGAs, and measured dynamic power consumption of the underlying circuit being operated at a low clock frequency of 3 MHz. The power traces have been collected by a digital oscilloscope at a sampling rate of 500 MS/s; corresponding sample power traces can be seen in Figure 11(a) and Figure 12(a). We followed the procedure suggested in [SM15] for collecting the traces suitable for a fixed-vs-random t-test. Similar to the simulation-based analysis explained above, the LFSR was randomly seeded before every measurement. We further made sure that the circuit is enabled once the input is given and the corresponding clock cycles are covered by the collected power traces. For each circuit, we collected 100M traces and performed a first- and second-order t-test, whose results are depicted in Figure 11(b), 11(c) and Figure 12(b), 12(c), respectively. The experimental results are in line with the simulation-based analyzes employing `PROLEAD`, i.e., detectable first-order leakage is seen for $n = 5$ while it is not the case for $n = 4$.

We would like to highlight that these experiments cannot be seen as a proof that unrolling an n-bit LFSRs for more than $n/2$ steps will always cause leakage. However, we presented an example (for small n) where this is indeed the case and consider the rule of thumb plausible based on the joint occurrence of glitches and transitions in practice.

More generally, we conclude from our two case studies that even in simple settings (i.e., analyzing one clock cycle), LFSRs as PRNGs can cause serious security issues. While it might be possible to work around these issues with thorough analysis, exhaustive verification of combined circuits becomes quickly prohibitive and some countermeasures to prevent issues from occurring (e.g., not unrolling too much, paying close attention to synthesis details) will add undesirable overheads. We therefore conclude that a simpler solution will be often desirable, such as the use of a proper cryptographically-secure PRNG.

5 Unrolled Trivium/Bivium for Masking Randomness

We finally provide guidelines for the secure use of Trivium and Bivium as PRNGs for masking randomness generation and analyze the efficiency and security of the resulting circuits.

5.1 Initialization

Trivium (resp., Bivium) normally expect two input parameters, a key and an initialization vector. One simple and efficient option to turn them into PRNGs is to use a randomly chosen fixed 80-bit IV for the device lifetime and at every device power-up obtain 80 true random bits from a TRNG as a seed to supply via the key input. If multiple instances in parallel are required (to generate many bits without increasing the critical path too much), each of them needs a different 80-bit IV, but can be fed with the same seed. Before randomness is produced, at least the initial phase of 1152 (resp., 708) steps needs to be executed. Please note that the amount of cycles for this initialization depends directly on the level of unrolling.

Using `Trivium_X64` this phase requires only $\lceil 1152/64 \rceil = 18$ cycles. Using `Trivium_X256` for example only $\lceil 1152/64 \rceil = 5$ cycles are needed. We note that these numbers of initialization cycles are in the same range as required for the pipelined block-based primitives discussed in Section 3 (5 for full SPEEDY, 24 for full Gimli). Alternatively, in order to skip the initialization phase altogether, the state of each Trivium (resp., Bivium) instance can be completely filled with 288 bits (resp., 177 bits) of TRNG output. In this case, the respective instance's output can be used immediately in the masked implementation without any further delay. Yet, due to the limited efficiency of TRNGs, this approach is often less convenient, especially when multiple Trivium (resp., Bivium) instances in parallel are needed.

5.2 Cost and Performance Analysis

Since the unrolling factor may be chosen arbitrarily in both Trivium and Bivium hardware implementations, it is crucial to investigate how the relevant cost and performance metrics scale when increasing the number of bits produced per cycle. Figure 13 illustrates the relationship between the area cost per output bit produced, which predictably decreases with larger levels of unrolling (blue curves), and the critical path delays defining the maximum possible operating frequency of the resulting circuits, which expectedly increase when more bits per cycle are generated (red curves). These results *exclude* the cost of the register stage that is needed before the generated bits can be used in a masked implementation in order to avoid timing dependencies between generated output bits. The curves are based on synthesis results using a commercial 65nm CMOS ASIC library and have been obtained without placing any timing constraints, i.e., the critical path delays are larger than reported in Appendix A and considered in Section 3, while the area figures are smaller (this choice has been made to reduce the tool runtime for the large unrolling factors). Critical path delays are given in nanoseconds and area is measured in gate equivalents (circuit area divided by 2-input NAND area). The x-axis covers the range from $2^0 = 1$ to $2^{13} = 8192$ bits per cycle. Please note that there are two separate y-axes, where the left y-axis (red) is in linear scale and the right y-axis (blue) is in logarithmic scale. We have chosen this kind of data representation here to highlight that there is a *sweet spot* (i.e., an effective zone) between $2^6 = 64$ and $2^{10} = 1024$ that combines a high maximum operating frequency (> 100 MHz) with a low average area cost per bit (< 60 GE for Trivium, < 40 GE for Bivium). The concrete values for $2^8 = 256$ output bits per cycle, which is the center of the effective area, are 35.48 GE per bit at a maximum 419 MHz for Trivium, and 22.9 GE per bit at a maximum 436 MHz for Bivium. The asymptotic values for the area per bit decrease even below 31 GE for Trivium and 20 GE for Bivium, but beyond a certain point the critical path delay becomes prohibitive.

In Figure 14 we have repeated the same analysis on an FPGA. Since our experimental results are obtained on Spartan-6 FPGAs we have chosen the same device model for the cost and performance analysis here. In contrast to the ASIC results presented above, the area cost is now measured by the number of six-input look-up tables (LUTs) that are instantiated by the synthesis and implementation tool (ISE Design Suite 14.7) per output bit per cycle. Again, these results *exclude* the cost of the registers that the finally generated bits need to pass through before being be used in a masked implementation. In general, the FPGA results are similar to the ASIC analysis. The *sweet spot* is again between $2^6 = 64$ and $2^{10} = 1024$. The concrete values for $2^8 = 256$ output bits per cycle are 4.82 LUTs per bit at a maximum 140 MHz for Trivium, and 3.65 GE per bit at a maximum 139 MHz for Bivium.

In summary, the cost and performance of unrolled Trivium and Bivium hardware implementations reach their optimal trade-off when a few hundreds, but less than thousands of bits per cycle are required by a masked implementation. In consequence, when more than a thousand bits per cycle are required by a masked implementation and if the unrolled PRNG becomes part of the critical path of the entire design, one may consider instantiating multiple PRNG instances in parallel while reducing the degree of unrolling.

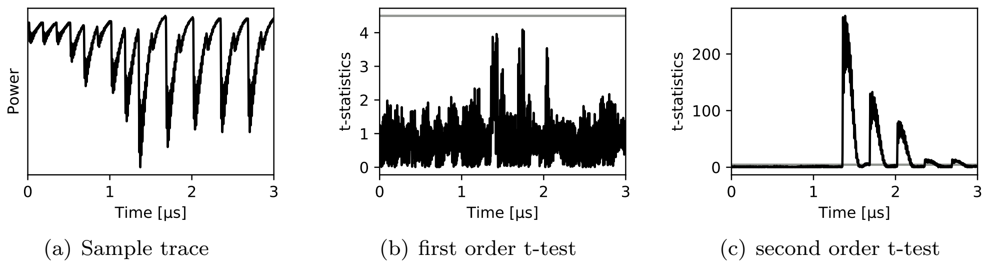


Figure 11: Case study 2: fixed-vs-random t-test results for the `ANDXOR4` experiment (100M traces) with `LFSR8_X4` as source of fresh randomness.

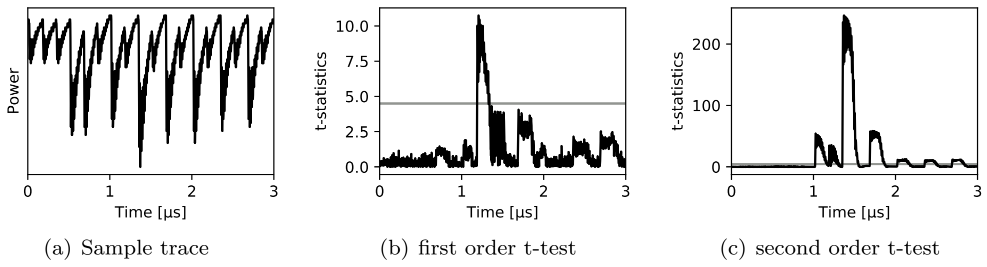


Figure 12: Case study 2: fixed-vs-random t-test results for the `ANDXOR5` experiment (100M traces) with `LFSR8_X5` as source of fresh randomness.

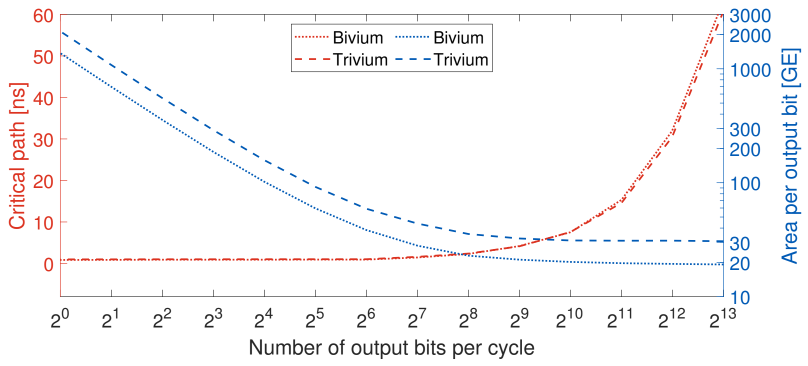


Figure 13: Area cost per bit and respective critical path delays for different degrees of unrolling (i.e., number of output bits produced per cycle).

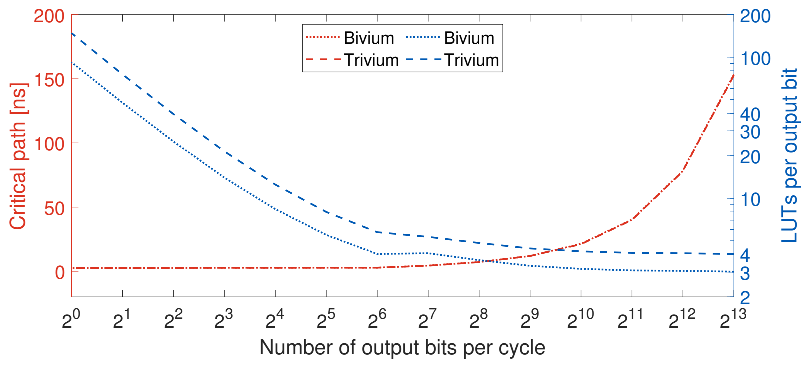


Figure 14: Number of LUTs per bit instantiated on a Spartan-6 FPGA for different degrees of unrolling (i.e., number of output bits produced per cycle).

5.3 Security against SCA Attacks

In this subsection we discuss the security of the pseudo-random phase of Bivium and Trivium against Side-Channel Analysis (SCA) attacks. Our analysis relies mostly on the state-of-the-art investigations reported by Kumar et al. at CHES 2022 [KDB+22]. To the best of our knowledge, this is the only successful SCA on the pseudo-random phase of Trivium, and no follow-up of this work has been published yet. More precisely, the authors successfully performed a simulated SCA on a software implementation of Trivium embedded on a 32-bit microcontroller. To this end, they assumed that each register over which the state is encoded leaked its Hamming weight. This leakage model has nice algebraic properties that allows one to instantiate a so-called Algebraic SCA [RSV09], by solving equations whose variables are the state bits, and whose constraints are given not only by the output stream bits, but also by the Hamming weights of the state registers. Concretely, Kumar et al. instantiated this approach using the Z3 Satisfiability Modulo Theory (SMT) solver.[9]

In a nutshell, knowing the Hamming weights of each register at each state update enables the adversary to straightforwardly guess some information about the input/output bits of these registers, provided that their size is less than the size of the three NLFSRs in Trivium. As an example, consider one particular register whose bit-size is small enough to not store the bits s_0, s_{92} or s_{177} — those are the only bits that may not be just shifted from one clock cycle to another. If the adversary observes a sequence $(h, h+1)$ (resp., $(h+1, h)$) of Hamming weights for some value h, this means that the bit leaving the register is 0 (resp., 1) while the bit entering the register is 1 (resp., 0). Likewise, if the adversary observes a sequence of two equal Hamming weights, this means that the register intput and output bits are equal. Overall, this leaks exactly $\frac{3}{2}$ bits per clock-cycle, and per such register — assuming pseudo-randomness of the input/output bits. Then, letting the state bit shift over a few updates is sufficient to easily recover the whole state. We could even extend Kumar et al.'s attack with the Hamming weight leakage model up to 128-bit registers. There, the bit-size becomes larger than the size of each NLFSR, but the last 31 bits of the third NLFSR need to be stored in a third register, whose bits are just shifted from one update to another.

However, for attacks with larger registers, and in particular for a whole state fitting into one single 288-bit register, the picture becomes different. Hereupon, we did not succeed in extending Kumar et al.'s attack, with 288, 300 or 3000 state updates, within 2 days of computation, even with this noise-free idealized leakage model. Likewise, the results are the same when turning to attacks on a 288-bit register assuming a Hamming distance leakage model. These results were already reported by Kumar et al. [KDB+22, Sec. 5.2.2]. Hereupon, although the authors succeed in recovering the state during the initialization phase using a few hundreds of rounds and a few hundreds of seconds, and assuming to know in advance a dozen of key bits, they could not succeed any attack on the pseudo-random phase whithin 2 days, unless potentially combined with an enumeration of 140 bits, which is more than the complexity of directly enumerating the 80-bit secret key of Trivium.

To summarize, when assuming a Hamming weight or a Hamming distance leakage model, operating on the whole Trivium state in parallel, as it is done for hardware implementations, seems to be the key ingredient to make side-channel attacks on Trivium prohibitive. Moreover, thanks to their open-source code, we replicated Kumar et al.'s experiment with the Hamming distance leakage model to verify whether considering reduced variants of Trivium, i.e., Bivium A and Bivium B could result in different outcomes, but we were not able to report any success for those variants within 2 days as well. Hence, unless a leakage model is observed in practice which proves to be significantly more informative than noise-free HW/HD on the whole state, attacks on hardware Trivium and Bivium have to be considered hard in our setting. Thus, given the current knowledge, state recovery through SCA attacks is not a primary concern for hardware implementations of stream ciphers with state sizes like Trivium's, meaning that the unpredictability requirement for masking randomness is expected to be fulfilled.

[9] The authors used an SMT solver rather than a SAT solver, in order to deal with potential noise into the SCA measurements, that could prevent from a fully accurate guessing of the Hamming weights.

On the effect of unrolling. Based on these evaluations, we predict that unrolling a hardware implementation can only be beneficial for the SCA security, as the sequential nature of the leakage should be much less exploitable. To ground this claim, let us consider side-channel analysis against an LFSR. Without considering unrolling, we have been able to successfully apply Kumar et al.'s approach to 64-bit LFSRs within a few seconds and a few rounds, thanks to the sequential nature of the leakages. However, at the extreme case where the adversary is not able to exploit the sequential nature of the Hamming weight/distance leakages due to unrolling — i.e. the adversary assumes that the leakages are independent, whereas they are not — the problem can be seen as an instance of the so-called *hidden multiplier problem*, introduced by Belaïd et al. [BCF+15]. There only exists two known attacks against this problem. The first one leverages the parity of the least significant bit of the Hamming weight of the leakage, and is therefore highly sensitive to the presence of noise [BFG14]. The second one leverages the most significant bit of the Hamming weight of the leakages, but the attack uses an instance of the LWE problem, for which there is no polynomially efficient algorithm [BCF+15]. Overall this contrasts with attacks against non-unrolled LFSRs, which should be straightforward to break using Kumar et al.'s approach if the noise level is not prohibitive. The same also applies to ($\leq$ 128-bit) LFSRs with non-linear filter functions when the state is only updated by one or a few bits per cycle. Hence, these are similarly insecure as regular LFSRs in the low-noise setting because the state can be recovered through SCA observations, which violates the unpredictability requirement for masking randomness.

6 Conclusions

Modern hardware masking schemes based on robust probing secure and composable gadgets are known to consume a large number of random bits per clock cycle, especially in parallel implementations and at higher orders. The required random numbers need to be uniformly distributed and unpredictable to adversaries. However, the secure and efficient generation of these bits has not yet received the attention it deserves from the research community. In this work, we improve upon this state of the art and provide contributions in multiple different directions. First of all, we clarify the relevant security properties that must be satisfied in the context of concurrent randomness generation for hardware masking. Then, after arguing that True Random Number Generators (TRNGs) are not cost-efficient when large quantities of bits are needed per cycle, we compare multiple potential candidates for building cost efficient Pseudo-Random Number Generators (PRNGs) instead, to stretch an initial seed obtained once at power-up into many pseudo-random bits during runtime. Our comparison includes block ciphers, permutations, stream ciphers and LFSRs. We arrive at the conclusion that the stream cipher Trivium and its reduced security variant Bivium B (for more aggressive optimizations) are impressively well-suited for our targeted application scenario when considering their unrolled implementation. Unrolled Trivium is basically a ready-to-use, trivial-to-instantiate, cheap, flexible, cryptographically strong and high-performance PRNG for hardware applications that already survived 18 years of cryptanalysis and heuristically inherits some of the properties of leakage-resilient stream ciphers. Hence, we highly recommend its adaptation for concurrent masking randomness generation and provide guidelines for its usage together with parametrizable source code. The only alternatives with even better performance according to our analysis are unrolled LFSRs, which offer no cryptographic strength. For that reason we also studied in detail what the security implications might be when using such a simple linear primitive for mask generation instead of a cryptographically strong PRNG and present two case studies where security degradation occurs in practice. Concrete results like that have, to the best of our knowledge, not been presented in the masking literature before. And while it is always easy to argue that the need for cryptographic strength is obvious for PRNGs in masking contexts, both the related side-channel literature (see Section 1.5) and concrete real-world examples (see OpenTitan) are showing that the opposite assumption is more prevalent, i.e., LFSR-based approaches are the norm, not the exception. Hence, we believe our work is an important cautionary study that still culminates in a positive result, as the secure options we recommend are also more efficient than many of the less secure ones used in previous works (e.g., a separate LFSR for each bit of randomness). While it may indeed be possible to instantiate certain LFSRs together with specific masked implementations securely, it at least requires a thorough analysis of the randomness generator and the masked circuit jointly. When instantiating a cryptographically strong PRNG like Trivium, the (SCA) security of randomness generation and masked implementation can be assessed separately, which is a desirable and advisable approach.

We demonstrate that securely generating n bits of randomness per cycle using our proposed approaches has an asymptotic cost of approximately $30n$ GE (ASIC) or $4n$ LUTs (FPGA) for Trivium (80-bit security), and approximately $20n$ GE (ASIC) and $3n$ LUTs (FPGA) for Bivium. These values are at least one order of magnitude better than what has been used as an estimate for the cost of producing random bits as recently as CHES 2022. For completeness, we also evaluated our solutions using NIST's 800-22 test suite for random and pseudo-random number generators for cryptographic applications. Unsurprisingly, the random values generated by Trivium and Bivium passed all statistical tests, while the unrolled 64-bit LFSR failed all linear complexity tests. We believe that our results can have a considerable impact on hardware masking research as they help to decide on crucial optimization trade-offs, and can guide future research directions, such as moving low randomness to a secondary design goal for hardware masking in the future.

We provide source code related to this work, including the unrolled stream cipher implementations, in the following GitHub repository:

`https://github.com/uclcrypto/randomness_for_hardware_masking`

Acknowledgments

François-Xavier Standaert is a senior research associate of the Belgian Fund for Scientific Research (F.R.S.-FNRS). This work has been funded in part by the EU through the ERC project SWORD (724725), the German Research Foundation (DFG) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972, the Walloon Region through the Win2Wal project PIRATE (1910082) and through the FEDER project USERMedia (501907-379156), and by SGS.

A Extended performance comparison

Table 6: Comparison of the critical path delay of relevant building blocks when synthesized for maximum operating frequency.

		Critical Path [ns]			
		Commercial Foundry		NanGate OCL	
Primitive	pip.	90 nm LP	65 nm LP	45 nm	15 nm
ES-TRNG		0.181586	0.135803	0.208436	0.021313
LFSR64		0.198822	0.155045	0.169445	0.014804
LFSR64_X32		0.195796	0.157433	0.197540	0.017748
LFSR64_X64		0.318669	0.224628	0.249999	0.022856
LFSR64_X96		0.311533	0.219201	0.242679	0.022425
LFSR64_X128		0.346743	0.265965	0.329463	0.028568
Bivium		0.217899	0.134968	0.194500	0.016976
Bivium_X32		0.195342	0.165267	0.218067	0.019418
Bivium_X48		0.242034	0.168228	0.192918	0.019445
Bivium_X64		0.236545	0.174493	0.223668	0.018651
Bivium_X72		0.378802	0.276818	0.328879	0.028219
Trivium		0.217623	0.118064	0.194513	0.016977
Trivium_X32		0.186734	0.180408	0.205709	0.020549
Trivium_X48		0.249180	0.166565	0.222982	0.018831
Trivium_X64		0.243608	0.173686	0.205199	0.018959
Trivium_X72		0.380121	0.274877	0.281385	0.028132
Kreyvium		0.279769	0.205444	0.234201	0.021628
Kreyvium_X32		0.291147	0.207658	0.241899	0.020947
Kreyvium_X48		0.280961	0.208351	0.237580	0.021515
Kreyvium_X64		0.272732	0.209060	0.266117	0.022508
Kreyvium_X72		0.396187	0.282755	0.332073	0.033164
GrainV1_80		0.552327	0.351267	0.431476	0.040580
GrainV1_80_X8		0.518995	0.406393	0.495407	0.044568
GrainV1_80_X16		0.559680	0.407079	0.565964	0.050532
GrainV1_80_X32		0.967295	0.788402	0.885287	0.089198
GrainV1_128		0.475218	0.322891	0.419144	0.039935
GrainV1_128_X16		0.503052	0.359735	0.472844	0.043818
GrainV1_128_X32		0.485202	0.341568	0.452880	0.044034
GrainV1_128_X48		0.769497	0.549393	0.705566	0.071954
MICKEY2_80		0.356980	0.234147	0.318378	0.033913
MICKEY2_80_X2		0.565007	0.385819	0.561346	0.057637
MICKEY2_80_X4		0.994678	0.747549	0.967886	0.096524
MICKEY2_80_X32		9.098467	6.634537	9.621321	0.897338
MICKEY2_128		0.357516	0.267185	0.319032	0.039074
MICKEY2_128_X2		0.630560	0.451635	0.605333	0.060051
MICKEY2_128_X4		1.159020	0.845300	1.085925	0.114799
MICKEY2_128_X32		10.208649	7.354148	10.711792	1.041437
Gimli-8		1.619769	1.175962	1.527335	0.148707
	✓	0.339260	0.287740	0.316472	0.029309
Gimli-16		2.894982	2.071035	2.758774	0.274899
	✓	0.356307	0.264966	0.337362	0.029584
Gimli-24		4.385863	3.127366	4.383278	0.421443
	✓	0.360362	0.266396	0.324092	0.029630
Subterranean2		0.405738	0.287102	0.406284	0.034361
Subterranean2_X2		0.722609	0.531270	0.719394	0.069042
Subterranean2_X4		1.366440	0.995288	1.351147	0.130649
Subterranean2_X8		2.635188	1.955079	2.662149	0.258758
SPEEDY-2-192		1.119609	0.806595	1.167997	0.110700
	✓	0.729113	0.521127	0.725931	0.066281
SPEEDY-3-192		1.733313	1.257028	1.842109	0.173992
	✓	0.737364	0.528380	0.728356	0.068753
SPEEDY-4-192		2.376308	1.709736	2.549025	0.237866
	✓	0.737837	0.521994	0.742094	0.069379
SPEEDY-5-192		2.994643	2.178075	3.187368	0.300466
	✓	0.745205	0.520147	0.722302	0.070076

Table 7: Comparison of the area consumption of relevant building blocks when synthesized for maximum operating frequency.

		Area [GE]			
		Commercial Foundry		NanGate OCL	
Primitive	pip.	90 nm LP	65 nm LP	45 nm	15 nm
ES-TRNG		306.25	297.00	237.00	273.00
LFSR64		465.50	496.75	502.67	635.75
LFSR64_X32		710.00	764.50	741.33	756.25
LFSR64_X64		903.50	954.25	830.33	915.50
LFSR64_X96		1045.25	1097.25	1020.67	1293.00
LFSR64_X128		1284.25	1288.50	1169.33	1588.25
Bivium		1266.25	1381.25	1450.67	1730.00
Bivium_X32		2310.75	2487.25	2050.67	2386.00
Bivium_X48		2624.25	2945.75	2315.33	2546.75
Bivium_X64		3019.00	3410.50	2607.67	2830.50
Bivium_X72		2840.75	2865.00	2684.67	3150.50
Trivium		1950.50	2080.25	2348.33	2649.50
Trivium_X32		3341.00	3834.25	3322.00	3777.50
Trivium_X48		3961.75	4560.75	3734.67	4195.50
Trivium_X64		4460.50	4994.00	4023.67	4483.75
Trivium_X72		4458.50	4388.00	4158.33	4825.25
Kreyvium		3641.25	3924.00	4193.00	5016.00
Kreyvium_X32		5099.25	5614.00	5582.67	6535.00
Kreyvium_X48		5780.75	6295.00	6013.00	6848.75
Kreyvium_X64		6318.00	6763.25	6376.00	7324.75
Kreyvium_X72		6543.50	6658.25	6747.33	7437.75
GrainV1_80		1236.75	1409.25	1466.67	1662.25
GrainV1_80_X8		2411.75	2508.00	2216.00	2592.00
GrainV1_80_X16		3455.00	3618.75	3130.33	3643.00
GrainV1_80_X32		5687.50	5779.00	5050.00	5841.25
GrainV1_128		1937.75	2086.25	2183.00	2544.50
GrainV1_128_X16		3732.25	3879.50	3470.33	4004.00
GrainV1_128_X32		5314.25	5880.50	5048.67	5555.753
GrainV1_128_X48		7003.25	7294.25	6076.67	7037.25
MICKEY2_80		2288.00	2363.25	2258.33	2611.50
MICKEY2_80_X2		3660.75	3600.00	3214.67	3775.50
MICKEY2_80_X4		6475.00	6410.50	5415.00	6253.00
MICKEY2_80_X32		62447.25	62259.50	48821.67	55493.00
MICKEY2_128		3594.00	3872.00	3542.33	4097.25
MICKEY2_128_X2		5520.50	5418.50	5068.00	5945.25
MICKEY2_128_X4		9591.25	9539.75	8320.67	9671.25
MICKEY2_128_X32		94555.25	95573.00	76321.67	85654.50
Gimli-8		27505.00	30970.50	19582.33	23870.00
	✓	26474.00	30086.75	24636.67	27280.00
Gimli-16		53388.50	61880.75	32578.33	38078.50
	✓	52309.25	60988.25	47771.33	53210.00
Gimli-24		69833.75	79734.25	50366.33	55725.75
	✓	78828.00	90802.75	71544.00	79725.25
Subterranean2		5526.25	6244.75	5045.67	5945.25
Subterranean2_X2		9391.75	10550.50	8504.33	9991.50
Subterranean2_X4		17274.00	18951.25	13559.00	16087.75
Subterranean2_X8		33381.00	38775.00	24922.33	30368.50
SPEEDY-2-192		16443.00	18983.25	10792.67	12859.25
	✓	13344.00	14274.50	10827.67	11939.50
SPEEDY-3-192		26658.50	31058.75	16391.67	19988.00
	✓	20808.25	21642.00	17287.33	20742.75
SPEEDY-4-192		37013.75	40820.25	21994.33	27143.25
	✓	27989.00	28356.25	23492.33	28574.00
SPEEDY-5-192		47364.00	53856.00	27903.33	34649.00
	✓	34604.00	36380.00	30011.00	36056.00

Table 8: Comparison of the power consumption of relevant building blocks when synthesized for maximum operating frequency, estimated for 100 MHz operation.

		Power [mW]			
		Commercial Foundry		NanGate OCL	
Primitive	pip.	90 nm LP	65 nm LP	45 nm	15 nm
`ES-TRNG`		0.0382	0.0219	0.0220	0.0074
`LFSR64`		0.1782	0.1054	0.0531	0.0252
`LFSR64_X32`		0.2241	0.1325	0.0881	0.0301
`LFSR64_X64`		0.2564	0.1556	0.1058	0.0360
`LFSR64_X96`		0.2708	0.1675	0.1311	0.0465
`LFSR64_X128`		0.3007	0.1853	0.1516	0.0581
`Bivium`		0.4833	0.2529	0.1618	0.0681
`Bivium_X32`		0.6458	0.4078	0.2550	0.0902
`Bivium_X48`		0.7168	0.4558	0.2910	0.0990
`Bivium_X64`		0.7872	0.5110	0.3310	0.1086
`Bivium_X72`		0.7304	0.4166	0.3406	0.1193
`Trivium`		0.7157	0.3991	0.2776	0.0974
`Trivium_X32`		0.9599	0.5950	0.4127	0.1439
`Trivium_X48`		1.0750	0.6864	0.4704	0.1583
`Trivium_X64`		1.1559	0.7276	0.5108	0.1705
`Trivium_X72`		1.1301	0.6370	0.5299	0.1859
`Kreyvium`		1.4335	0.7247	0.4449	0.2018
`Kreyvium_X32`		1.6782	0.9694	0.6799	0.2571
`Kreyvium_X48`		1.7909	1.0446	0.7525	0.2730
`Kreyvium_X64`		1.8916	1.0999	0.8017	0.2923
`Kreyvium_X72`		1.9194	1.0440	0.8620	0.2986
`GrainV1_80`		0.4479	0.2424	0.1742	0.0661
`GrainV1_80_X8`		0.6472	0.3750	0.2829	0.0998
`GrainV1_80_X16`		0.8222	0.4983	0.4107	0.1387
`GrainV1_80_X32`		1.2170	0.7298	0.7144	0.2369
`GrainV1_128`		0.7005	0.3705	0.2532	0.1016
`GrainV1_128_X16`		1.0080	0.5954	0.4455	0.1554
`GrainV1_128_X32`		1.2855	0.8108	0.6683	0.2167
`GrainV1_128_X48`		1.5699	0.9661	0.8331	0.2785
`MICKEY2_80`		0.6057	0.3425	0.2700	0.0942
`MICKEY2_80_X2`		0.8498	0.4772	0.3981	0.1399
`MICKEY2_80_X4`		1.3253	0.7962	0.7055	0.2344
`MICKEY2_80_X32`		18.3180	11.7230	11.9660	3.5713
`MICKEY2_128`		0.9650	0.5482	0.4274	0.1536
`MICKEY2_128_X2`		1.2706	0.7234	0.6327	0.2212
`MICKEY2_128_X4`		1.9906	1.1607	1.1222	0.3654
`MICKEY2_128_X32`		26.8160	17.6274	18.8430	5.6178
`Gimli-8`		8.0824	5.5274	4.1302	1.4268
	✓	9.8002	6.1129	4.5818	1.5078
`Gimli-16`		13.4989	9.8060	6.4017	2.0360
	✓	18.1331	11.7633	8.8154	2.8464
`Gimli-24`		17.4224	13.0373	9.7293	2.9185
	✓	26.6511	17.3752	13.1350	4.2043
`Subterranean2`		0.8247	0.5043	0.3999	0.1492
`Subterranean2_X2`		1.1644	0.7594	0.6346	0.2423
`Subterranean2_X4`		2.8752	1.9857	1.6683	0.5661
`Subterranean2_X8`		9.3689	7.1013	5.2655	1.7625
`SPEEDY-2-192`		4.6353	3.0388	1.9893	0.7162
	✓	3.8586	2.3635	1.9718	0.5813
`SPEEDY-3-192`		7.3420	5.0057	3.0958	1.1275
	✓	5.8516	3.5152	3.1976	1.1105
`SPEEDY-4-192`		10.1531	6.6220	4.1915	1.5378
	✓	7.8388	4.6164	4.3498	1.5172
`SPEEDY-5-192`		12.9344	8.7471	5.3422	1.9655
	✓	9.6633	5.8834	5.5623	1.9002

Table 9: Comparison of the maximum throughput of relevant building blocks when synthesized for maximum operating frequency.

		Throughput [Gbit/s]			
		Commercial Foundry		NanGate OCL	
Primitive	pip.	90 nm LP	65 nm LP	45 nm	15 nm
ES-TRNG		5.5070	7.3636	4.7976	46.9197
LFSR64		5.0296	6.4497	5.9016	67.5493
LFSR64_X32		163.4354	203.2611	161.9925	1803.0201
LFSR64_X64		200.8353	284.9155	256.0010	2800.1400
LFSR64_X96		308.1536	437.9542	395.5843	4280.9365
LFSR64_X128		369.1495	481.2663	388.5110	4480.5377
Bivium		4.5893	7.4092	5.1414	58.9067
Bivium_X32		163.8153	193.6261	146.7439	1647.9555
Bivium_X48		198.3192	285.3271	248.8104	2468.5009
Bivium_X64		270.5616	366.7769	286.1384	3431.4514
Bivium_X72		190.0729	260.0987	218.9255	2551.4724
Trivium		4.5951	8.4700	5.1410	58.9032
Trivium_X32		171.3668	177.3757	155.5596	1557.2534
Trivium_X48		192.6318	288.1758	215.2640	2548.9884
Trivium_X64		262.7172	368.4811	311.8924	3375.7055
Trivium_X72		189.4134	261.9353	255.8772	2559.3630
Kreyvium		3.5744	4.8675	4.2698	46.2364
Kreyvium_X32		109.9101	154.0995	132.2866	1527.6651
Kreyvium_X48		170.8422	230.3805	202.0372	2231.0016
Kreyvium_X64		234.6626	306.1322	240.4957	2843.4334
Kreyvium_X72		181.7324	254.6374	216.8198	2171.0288
GrainV1_80		1.8105	2.8468	2.3176	24.6427
GrainV1_80_X8		15.4144	19.6854	16.1483	179.5010
GrainV1_80_X16		28.5878	39.3044	28.2703	316.6310
GrainV1_80_X32		33.0819	40.5884	36.1465	358.7524
GrainV1_128		2.1043	3.0970	2.3858	25.0407
GrainV1_128_X16		31.8059	44.4772	33.8378	365.1467
GrainV1_128_X32		65.9519	93.6856	70.6589	726.7112
GrainV1_128_X48		62.3784	87.3692	68.0305	667.0929
MICKEY2_80		2.8013	4.2708	3.1409	29.4872
MICKEY2_80_X2		3.5398	5.1838	3.5629	34.6999
MICKEY2_80_X4		4.0214	5.3508	4.1327	41.4405
MICKEY2_80_X32		3.5171	4.8232	3.3259	35.6610
MICKEY2_128		2.7971	3.7427	3.1345	25.5925
MICKEY2_128_X2		3.1718	4.4284	3.3040	33.3050
MICKEY2_128_X4		3.4512	4.7320	3.6835	34.8435
MICKEY2_128_X32		3.1346	4.3513	2.9874	30.7268
Gimli-8		237.0708	326.5412	251.4183	2582.2591
	✓	1131.8753	1334.5381	1213.3775	13101.7776
Gimli-16		132.6433	185.4145	139.1923	1396.8767
	✓	1077.7223	1449.2425	1138.2432	12979.9892
Gimli-24		87.5540	122.7870	87.6057	911.1552
	✓	1065.5952	1441.4631	1184.8487	12959.8380
Subterranean2		78.8686	111.4586	78.7626	931.2884
Subterranean2_X2		88.5680	120.4661	88.9638	926.9720
Subterranean2_X4		93.6741	128.6060	94.7343	979.7243
Subterranean2_X8		97.1468	130.9410	96.1629	989.3414
SPEEDY-2-192		171.4884	238.0377	164.3840	1734.4173
	✓	263.3337	368.4323	264.4879	2896.7577
SPEEDY-3-192		110.7705	152.7412	104.2284	1103.4990
	✓	260.3870	363.3748	263.6074	2792.6054
SPEEDY-4-192		80.7976	112.2980	75.3229	807.1772
	✓	260.2201	367.8203	258.7273	2767.4080
SPEEDY-5-192		64.1145	88.1512	60.2378	639.0074
	✓	257.6472	369.1264	265.8168	2739.8824

Table 10: Comparison of the energy consumption per bit of relevant building blocks when synthesized for maximum operating frequency.

		Energy per bit [fJ/bit]			
		Commercial Foundry		NanGate OCL	
Primitive	pip.	90 nm LP	65 nm LP	45 nm	15 nm
`ES-TRNG`		382.0000	219.0000	220.0000	74.0000
`LFSR64`		1782.0000	1054.0000	531.0000	252.0000
`LFSR64_X32`		70.0312	41.4062	27.5312	9.4062
`LFSR64_X64`		40.0625	24.3125	16.5312	5.6250
`LFSR64_X96`		28.2083	17.4479	13.6562	4.8438
`LFSR64_X128`		23.4922	14.4766	11.8438	4.5391
`Bivium`		4833.0000	2529.0000	1618.0000	681.0000
`Bivium_X32`		201.8125	127.4375	79.6875	28.1875
`Bivium_X48`		149.3333	94.9583	60.6250	20.6250
`Bivium_X64`		123.0000	79.8438	51.7188	16.9688
`Bivium_X72`		101.4444	57.8611	47.3056	16.5694
`Trivium`		7157.0000	3991.0000	2776.0000	974.0000
`Trivium_X32`		299.9688	185.9375	128.9688	44.9688
`Trivium_X48`		223.9583	143.0000	98.0000	32.9792
`Trivium_X64`		180.6094	113.6875	79.8125	26.6406
`Trivium_X72`		156.9583	88.4722	73.5972	25.8194
`Kreyvium`		14335.0000	7247.0000	4449.0000	2018.0000
`Kreyvium_X32`		524.4375	302.9375	212.4687	80.3438
`Kreyvium_X48`		373.1042	217.6250	156.7708	56.8750
`Kreyvium_X64`		295.5625	171.8594	125.2656	45.6719
`Kreyvium_X72`		266.5833	145.0000	119.7222	41.4722
`GrainV1_80`		4479.0000	2424.0000	1742.0000	661.0000
`GrainV1_80_X8`		809.0000	468.7500	353.6250	124.7500
`GrainV1_80_X16`		513.8750	311.4375	256.6875	86.6875
`GrainV1_80_X32`		380.3125	228.0625	223.2500	74.0312
`GrainV1_128`		7005.0000	3705.0000	2532.0000	1016.0000
`GrainV1_128_X16`		630.0000	372.1250	278.4375	97.1250
`GrainV1_128_X32`		401.7188	253.3750	208.8438	67.7188
`GrainV1_128_X48`		327.0625	201.2708	173.5625	58.0208
`MICKEY2_80`		6057.0000	3425.0000	2700.0000	942.0000
`MICKEY2_80_X2`		4249.0000	2386.0000	1990.5000	699.5000
`MICKEY2_80_X4`		3313.2500	1990.5000	1763.7500	586.0000
`MICKEY2_80_X32`		5724.3750	3663.4375	3739.3750	1116.0312
`MICKEY2_128`		9650.0000	5482.0000	4274.0000	1536.0000
`MICKEY2_128_X2`		6353.0000	3617.0000	3163.5000	1106.0000
`MICKEY2_128_X4`		4976.5000	2901.7500	2805.5000	913.5000
`MICKEY2_128_X32`		8380.0000	5508.5625	5888.4375	1755.5625
`Gimli-8`		210.4792	143.9427	107.5573	37.1563
	✓	255.2135	159.1901	119.3177	39.2656
`Gimli-16`		351.5339	255.3646	166.7109	53.0208
	✓	472.2161	306.3359	229.5677	74.1250
`Gimli-24`		453.7083	339.5130	253.3672	76.0026
	✓	694.0391	452.4792	342.0573	109.4870
`Subterranean2`		257.7188	157.5938	124.9688	46.6250
`Subterranean2_X2`		181.9375	118.6562	99.1562	37.8594
`Subterranean2_X4`		224.6250	155.1328	130.3359	44.2266
`Subterranean2_X8`		365.9727	277.3945	205.6836	68.8477
`SPEEDY-2-192`		241.4219	158.2708	103.6094	37.3021
	✓	200.9688	123.0990	102.6979	30.2760
`SPEEDY-3-192`		382.3958	260.7135	161.2396	58.7240
	✓	304.7708	183.0833	166.5417	57.8385
`SPEEDY-4-192`		528.8073	344.8958	218.3073	80.0938
	✓	408.2708	240.4375	226.5521	79.0208
`SPEEDY-5-192`		673.6667	455.5781	278.2396	102.3698
	✓	503.2969	306.4271	289.7031	98.9688

Table 11: Comparison of the product between power consumption, area consumption and critical path delay of relevant building blocks when synthesized for maximum operating frequency.

		Power-Area-Time product [mW][GE][ns]			
		Commercial Foundry		NanGate OCL	
Primitive	pip.	90 nm LP	65 nm LP	45 nm	15 nm
`ES-TRNG`		2.1243	0.8833	1.0868	0.0431
`LFSR64`		16.4927	8.1178	4.5228	0.2372
`LFSR64_X32`		31.1533	15.9474	12.9016	0.4040
`LFSR64_X64`		73.8220	33.3531	21.9621	0.7533
`LFSR64_X96`		88.1806	40.2868	32.4728	1.3483
`LFSR64_X128`		133.9031	63.5016	58.4040	2.6362
`Bivium`		133.3495	47.1468	45.6527	2.0000
`Bivium_X32`		291.5054	167.6304	114.0318	4.1791
`Bivium_X48`		455.2811	225.8752	129.9806	4.9026
`Bivium_X64`		562.1626	304.1004	193.0565	5.7332
`Bivium_X72`		785.9701	330.3986	300.7265	10.6062
`Trivium`		303.7958	98.0200	126.8023	4.3811
`Trivium_X32`		598.8608	411.5790	282.0249	11.1701
`Trivium_X48`		1061.2280	521.4315	391.7323	12.5066
`Trivium_X64`		1256.0165	631.1114	421.7436	14.4938
`Trivium_X72`		1915.2590	768.3241	620.0316	25.2348
`Kreyvium`		1460.3192	584.2258	436.8939	21.8925
`Kreyvium_X32`		2491.5083	1130.1188	918.1657	35.1941
`Kreyvium_X48`		2908.7176	1370.0655	1074.9978	40.2268
`Kreyvium_X64`		3259.4553	1555.1762	1360.2941	48.1902
`Kreyvium_X72`		4975.9478	1965.4902	1931.4025	73.6543
`GrainV1_80`		305.9562	119.9936	110.2395	4.4587
`GrainV1_80_X8`		810.0913	382.2126	310.5738	11.5289
`GrainV1_80_X16`		1589.8835	734.0543	727.6183	25.5330
`GrainV1_80_X32`		6695.3137	3325.0966	3193.8676	123.4315
`GrainV1_128`		645.0580	249.5804	231.6758	10.3240
`GrainV1_128_X16`		1892.5360	830.9354	731.0320	27.2645
`GrainV1_128_X32`		3314.6421	1628.5653	1528.0290	53.0139
`GrainV1_128_X48`		8460.1595	3871.5587	3571.9094	141.0208
`MICKEY2_80`		494.7177	189.5217	194.1307	8.3427
`MICKEY2_80_X2`		1757.6833	662.8062	718.3882	30.4434
`MICKEY2_80_X4`		8535.6477	3815.5201	3697.5979	141.4755
`MICKEY2_80_X32`		10407815.7900	4842337.0373	5620776.7213	177836.3749
`MICKEY2_128`		1239.9406	567.1350	483.0118	24.5907
`MICKEY2_128_X2`		4422.9668	1770.2931	1941.0146	78.9724
`MICKEY2_128_X4`		22128.4065	9359.8275	10139.7768	405.6853
`MICKEY2_128_X32`		25884984.9057	12389558.8766	15404941.1574	501128.9139
`Gimli-8`		360085.0347	201308.6328	123529.2349	5064.6208
	✓	88021.1749	52920.3622	35723.4526	1205.5608
`Gimli-16`		2086373.0632	1256709.4941	575360.7881	21312.3218
	✓	337967.4729	190092.7241	142071.0266	4480.7022
`Gimli-24`		5336154.6290	3250957.4325	2147933.9245	68541.6358
	✓	757067.5566	420297.2160	304559.1178	9931.6462
`Subterranean2`		1849.1503	904.1495	819.7850	30.4793
`Subterranean2_X2`		7902.2740	4256.5616	3882.4599	167.1466
`Subterranean2_X4`		67865.8889	37454.1775	30563.5933	1189.8564
`Subterranean2_X8`		824137.2619	538336.6870	349349.8962	13849.8877
`SPEEDY-2-192`		85334.6251	46529.4812	25076.7302	1019.5243
	✓	37541.4147	17581.6685	15498.6266	460.0187
`SPEEDY-3-192`		339255.6457	195431.1298	93478.4328	3921.1655
	✓	89782.5878	40197.0149	40262.0385	1583.7132
`SPEEDY-4-192`		893026.7767	462161.6370	234992.6627	9928.7385
	✓	161881.5656	68330.9943	75832.3129	3007.7512
`SPEEDY-5-192`		1834592.9331	1026055.8860	475125.5513	20462.5187
	✓	249188.2304	111331.2714	120574.0067	4801.1598

References

[ABL08] François Arnault, Thierry P. Berger, and Cédric Lauradoux. F-FCSR stream ciphers. In Matthew J. B. Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 170–178. Springer, 2008.

[Alf96] P Alfke. Efficient shift registers, lfsr counters, and long-pseudo-random generators, 1996.

[AZ21] Mark D. Aagaard and Nusa Zidaric. ASIC benchmarking of round 2 candidates in the NIST lightweight cryptography standardization process: (preliminary results). *IACR Cryptol. ePrint Arch.*, page 49, 2021.

[BBC$^+$08] Côme Berbain, Olivier Billet, Anne Canteaut, Nicolas T. Courtois, Blandine Debraize, Henri Gilbert, Louis Goubin, Aline Gouget, Louis Granboulan, Cédric Lauradoux, Marine Minier, Thomas Pornin, and Hervé Sibert. Decimv2. In Matthew J. B. Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 140–151. Springer, 2008.

[BBC$^+$20] Davide Bellizia, Olivier Bronchain, Gaëtan Cassiers, Vincent Grosso, Chun Guo, Charles Momin, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. Mode-level vs. implementation-level physical security in symmetric cryptography - A practical guide through the leakage-resistance jungle. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 369–400. Springer, 2020.

[BBD$^+$16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 116–129. ACM, 2016.

[BBP$^+$16] Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Randomness complexity of private circuits for multiplication. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 616–648. Springer, 2016.

[BCF$^+$15] Sonia Belaïd, Jean-Sébastien Coron, Pierre-Alain Fouque, Benoît Gérard, Jean-Gabriel Kammerer, and Emmanuel Prouff. Improved side-channel analysis of finite-field multiplication. *IACR Cryptol. ePrint Arch.*, page 542, 2015.

[BCG$^+$12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012.

[BCM23] Subhadeep Banik, Daniel Collins, and Willi Meier. Near collision attack against grain V1. In Mehdi Tibouchi and Xiaofeng Wang, editors, *Applied Cryptography and Network Security - 21st International Conference, ACNS 2023, Kyoto, Japan, June 19-22, 2023, Proceedings, Part I*, volume 13905 of *Lecture Notes in Computer Science*, pages 178–207. Springer, 2023.

[BCPZ16] Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 23–39. Springer, 2016.

[BD00] Eli Biham and Orr Dunkelman. Cryptanalysis of the A5/1 GSM stream cipher. In Bimal K. Roy and Eiji Okamoto, editors, *Progress in Cryptology - INDOCRYPT 2000, First International Conference in Cryptology in India, Calcutta, India, December 10-13, 2000, Proceedings*, volume 1977 of *Lecture Notes in Computer Science*, pages 43–51. Springer, 2000.

[BD08] Steve Babbage and Matthew Dodd. The MICKEY stream ciphers. In Matthew J. B. Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 191–209. Springer, 2008.

[BDPA10] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge-based pseudo-random number generators. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 2010.

[BDPA13] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 313–314. Springer, 2013.

[Ber08] Daniel J. Bernstein. The salsa20 family of stream ciphers. In Matthew J. B. Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 84–97. Springer, 2008.

[BFG14] Sonia Belaïd, Pierre-Alain Fouque, and Benoît Gérard. Side-channel analysis of multiplications in GF(2128) - application to AES-GCM. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 306–325. Springer, 2014.

[BGN$^+$14a] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-order threshold implementations. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2014.

[BGN+14b] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. A more efficient AES threshold implementation. In David Pointcheval and Damien Vergnaud, editors, *Progress in Cryptology - AFRICACRYPT 2014 - 7th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 28-30, 2014. Proceedings*, volume 8469 of *Lecture Notes in Computer Science*, pages 267–284. Springer, 2014.

[BGW98] Marc Briceno, Ian Goldberg, and David Wagner. A pedagogical implementation of a5/1. 1998.

[Bir08] Alex Biryukov. Design of a new stream cipher-lex. In Matthew J. B. Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 48–56. Springer, 2008.

[BKL+17] Daniel J. Bernstein, Stefan Kölbl, Stefan Lucks, Pedro Maat Costa Massolino, Florian Mendel, Kashif Nawaz, Tobias Schneider, Peter Schwabe, François-Xavier Standaert, Yosuke Todo, and Benoît Viguier. Gimli : A Cross-Platform Permutation. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 299–320. Springer, 2017.

[BKSQ07] Philippe Bulens, Kassem Kalach, François-Xavier Standaert, and Jean-Jacques Quisquater. Fpga implementations of estream phase-2 focus candidates with hardware profile. 2007.

[BLM+05] An Braeken, Joseph Lano, Nele Mentens, Bart Preneel, and Ingrid Verbauwhede. Sfinks: A synchronous stream cipher for restricted hardware environments. 2005.

[BM82] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo random bits. In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 112–117. IEEE Computer Society, 1982.

[BMV07] Sanjay Burman, Debdeep Mukhopadhyay, and Kamakoti Veezhinathan. LFSR based stream ciphers are vulnerable to power attacks. In K. Srinathan, C. Pandu Rangan, and Moti Yung, editors, *Progress in Cryptology - INDOCRYPT 2007, 8th International Conference on Cryptology in India, Chennai, India, December 9-13, 2007, Proceedings*, volume 4859 of *Lecture Notes in Computer Science*, pages 384–392. Springer, 2007.

[BRS+10] Lawrence E. Bassham, Andrew L. Rukhin, Juan Soto, James R. Nechvatal, Miles E. Smid, Elaine B. Barker, Stefan D. Leigh, Mark Levenson, Mark Vangel, David L. Banks, Nathanael Alan Heckert, James F. Dray, and San Vo. A statistical test suite for random and pseudorandom number generators for cryptographic applications - rev. 1a. *NIST Special Publication (SP) 800-22*, 2010.

[Can06] Christophe De Cannière. Trivium: A stream cipher construction inspired by block cipher design principles. In Sokratis K. Katsikas, Javier López, Michael Backes, Stefanos Gritzalis, and Bart Preneel, editors, *Information Security, 9th International Conference, ISC 2006, Samos Island, Greece, August 30 - September 2, 2006, Proceedings*, volume 4176 of *Lecture Notes in Computer Science*, pages 171–186. Springer, 2006.

[Can11a] Anne Canteaut. Combination generator. In Henk C. A. van Tilborg and Sushil Jajodia, editors, *Encyclopedia of Cryptography and Security, 2nd Ed*, pages 222–224. Springer, 2011.

[Can11b] Anne Canteaut. Correlation attack for stream ciphers. In Henk C. A. van Tilborg and Sushil Jajodia, editors, *Encyclopedia of Cryptography and Security, 2nd Ed*, pages 261–262. Springer, 2011.

[Can11c] Anne Canteaut. Filter generator. In Henk C. A. van Tilborg and Sushil Jajodia, editors, *Encyclopedia of Cryptography and Security, 2nd Ed*, pages 458–460. Springer, 2011.

[CBG$^+$17] Thomas De Cnudde, Begül Bilgin, Benedikt Gierlichs, Ventzislav Nikov, Svetla Nikova, and Vincent Rijmen. Does coupling affect the security of masked implementations? In Sylvain Guilley, editor, *Constructive Side-Channel Analysis and Secure Design - 8th International Workshop, COSADE 2017, Paris, France, April 13-14, 2017, Revised Selected Papers*, volume 10348 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2017.

[CCF$^+$16] Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 313–333. Springer, 2016.

[CFAF13] Abdelkarim Cherkaoui, Viktor Fischer, Alain Aubert, and Laurent Fesquet. A self-timed ring based true random number generator. In *19th IEEE International Symposium on Asynchronous Circuits and Systems, ASYNC 2013, Santa Monica, CA, USA, May 19-22, 2013*, pages 99–106. IEEE Computer Society, 2013.

[CFFA13] Abdelkarim Cherkaoui, Viktor Fischer, Laurent Fesquet, and Alain Aubert. A very high speed true random number generator with entropy assessment. In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings*, volume 8086 of *Lecture Notes in Computer Science*, pages 179–196. Springer, 2013.

[CGLS21] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware Private Circuits: From Trivial Composition to Full Verification. *IEEE Trans. Computers*, 70(10):1677–1690, 2021.

[CGP$^+$12] Jean-Sébastien Coron, Christophe Giraud, Emmanuel Prouff, Soline Renner, Matthieu Rivain, and Praveen Kumar Vadnala. Conversion of security proofs from one leakage model to another: A new issue. In Werner Schindler and Sorin A. Huss, editors, *Constructive Side-Channel Analysis and Secure Design - Third International Workshop, COSADE 2012, Darmstadt, Germany, May 3-4, 2012. Proceedings*, volume 7275 of *Lecture Notes in Computer Science*, pages 69–81. Springer, 2012.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.

[CMM14] Abhishek Chakraborty, Bodhisatwa Mazumdar, and Debdeep Mukhopadhyay. Fibonacci LFSR vs. galois LFSR: which is more vulnerable to power attacks? In Rajat Subhra Chakraborty, Vashek Matyas, and Patrick Schaumont, editors, *Security, Privacy, and Applied Cryptography Engineering - 4th International*

Conference, SPACE 2014, Pune, India, October 18-22, 2014. Proceedings, volume 8804 of *Lecture Notes in Computer Science*, pages 14–27. Springer, 2014.

[CP08] Christophe De Cannière and Bart Preneel. Trivium. In Matthew J. B. Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 244–266. Springer, 2008.

[CRB$^+$16] Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Masking AES with d+1 shares in hardware. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 194–212. Springer, 2016.

[CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Trans. Inf. Forensics Secur.*, 15:2542–2555, 2020.

[CS21] Gaëtan Cassiers and François-Xavier Standaert. Provably secure hardware masking in the transition- and glitch-robust probing model: Better safe than sorry. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(2):136–158, 2021.

[DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 423–440. Springer, 2014.

[DEMS20] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläer. Status update on ascon v1. 2. *Submission to the NIST LWC competition*, 2020.

[DFH$^+$16] Stefan Dziembowski, Sebastian Faust, Gottfried Herold, Anthony Journault, Daniel Masny, and François-Xavier Standaert. Towards sound fresh re-keying with hard (physical) learning problems. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 272–301. Springer, 2016.

[DFS15] Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. Making masking security proofs concrete - or how to evaluate the security of any leaking device. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 401–429. Springer, 2015.

[DK05] Joan Daemen and Paris Kitsos. The self-synchronizing stream cipher mosquito: estream documentation, version 2. 2005.

[DK08] Joan Daemen and Paris Kitsos. The self-synchronizing stream cipher moustique. In Matthew J. B. Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 210–223. Springer, 2008.

[DMMR20] Joan Daemen, Pedro Maat Costa Massolino, Alireza Mehrdad, and Yann Rotella. The Subterranean 2.0 Cipher Suite. *IACR Trans. Symmetric Cryptol.*, 2020(S1):262–294, 2020.

[E02] Specification of the bluetooth system - version 1.1. http://www.tscm.com/BluetoothSpec.pdf. Accessed: 2023-07-15.

[FD02] Viktor Fischer and Milos Drutarovský. True random number generator embedded in reconfigurable hardware. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 415–430. Springer, 2002.

[FGP$^+$18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence of physical defaults & the robust probing model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):89–120, 2018.

[FL14] Viktor Fischer and David Lubicz. Embedded evaluation of randomness in oscillator based elementary TRNG. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, volume 8731 of *Lecture Notes in Computer Science*, pages 527–543. Springer, 2014.

[Fon11a] Caroline Fontaine. Clock-controlled generator. In Henk C. A. van Tilborg and Sushil Jajodia, editors, *Encyclopedia of Cryptography and Security, 2nd Ed*, pages 211–212. Springer, 2011.

[Fon11b] Caroline Fontaine. Shrinking generator. In Henk C. A. van Tilborg and Sushil Jajodia, editors, *Encyclopedia of Cryptography and Security, 2nd Ed*, pages 1197–1198. Springer, 2011.

[GB08] Tim Good and Mohammed Benaissa. ASIC hardware performance. In Matthew J. B. Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 267–293. Springer, 2008.

[GBC$^+$08] Benedikt Gierlichs, Lejla Batina, Christophe Clavier, Thomas Eisenbarth, Aline Gouget, Helena Handschuh, Timo Kasper, Kerstin Lemke-Rust, Stefan Mangard, Amir Moradi, and Elisabeth Oswald. Susceptibility of estream candidates towards side channel analysis. 2008.

[GGK05] Berndt M. Gammel, Rainer Göttfert, and Oliver Kniffler. The achterbahn stream cipher. 2005.

[GGV05] Carmi Gressel, Ran Granot, and Gabi Vago. Zk-crypt - a compact stream cipher and more. 2005.

[GLB$^+$06] Frank K. Gürkaynak, Peter Luethi, Nico Bernold, René Blattmann, Victoria M Goode, Marcel Marghitola, Hubert Kaeslin, Norbert Felber, and Wolfgang Fichtner. Hardware evaluation of estream candidates. 2006.

[GM17] Hannes Groß and Stefan Mangard. Reconciling d+1 masking in hardware and software. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 115–136. Springer, 2017.

[GMK08] Danilo Gligoroski, Smile Markovski, and Svein J. Knapskog. The stream cipher edon80. In Matthew J. B. Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 152–169. Springer, 2008.

[GMK16] Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In Begül Bilgin, Svetla Nikova, and Vincent Rijmen, editors, *Proceedings of the ACM Workshop on Theory of Implementation Security, TIS@CCS 2016 Vienna, Austria, October, 2016*, page 3. ACM, 2016.

[GMK17] Hannes Groß, Stefan Mangard, and Thomas Korak. An efficient side-channel protected AES implementation with arbitrary protection order. In Helena Handschuh, editor, *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings*, volume 10159 of *Lecture Notes in Computer Science*, pages 95–112. Springer, 2017.

[GSB07] Kris Gaj, Gabriel Southern, and Ramakrishna Bachimanchi. Comparison of hardware performance of selected phase ii estream candidates. 2007.

[GSF13] Vincent Grosso, François-Xavier Standaert, and Sebastian Faust. Masking vs. multiparty computation: How large is the gap for aes? In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings*, volume 8086 of *Lecture Notes in Computer Science*, pages 400–416. Springer, 2013.

[HCK$^+$08] David Hwang, Mark Chaney, Shashi Prashanth Karanam, Nick Ton, and Kris Gaj. Comparison of fpga-targeted hardware implementations of estream stream cipher candidates. 2008.

[HJM07] Martin Hell, Thomas Johansson, and Willi Meier. Grain: a stream cipher for constrained environments. *Int. J. Wirel. Mob. Comput.*, 2(1):86–93, 2007.

[HJMM06] Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. A stream cipher proposal: Grain-128. In *Proceedings 2006 IEEE International Symposium on Information Theory, ISIT 2006, The Westin Seattle, Seattle, Washington, USA, July 9-14, 2006*, pages 1614–1618. IEEE, 2006.

[HKM17] Matthias Hamann, Matthias Krause, and Willi Meier. LIZARD - A lightweight stream cipher for power-constrained devices. *IACR Trans. Symmetric Cryptol.*, 2017(1):45–79, 2017.

[HL11] Zhenyu Huang and Dongdai Lin. Attacking bivium and trivium with the characteristic set method. In Abderrahmane Nitaj and David Pointcheval, editors, *Progress in Cryptology - AFRICACRYPT 2011 - 4th International Conference on Cryptology in Africa, Dakar, Senegal, July 5-7, 2011. Proceedings*, volume 6737 of *Lecture Notes in Computer Science*, pages 77–91. Springer, 2011.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[JD06] Antoine Joux and Pascal Delaunay. Galois lfsr, embedded devices and side channel weaknesses. In Rana Barua and Tanja Lange, editors, *Progress in*

Cryptology - INDOCRYPT 2006, 7th International Conference on Cryptology in India, Kolkata, India, December 11-13, 2006, Proceedings, volume 4329 of *Lecture Notes in Computer Science*, pages 436–451. Springer, 2006.

[JHK08] Cees J. A. Jansen, Tor Helleseth, and Alexander Kholosha. Cascade jump controlled sequence generator and pomaranch stream cipher. In Matthew J. B. Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 224–243. Springer, 2008.

[KDB$^+$22] Satyam Kumar, Vishnu Asutosh Dasu, Anubhab Baksi, Santanu Sarkar, Dirmanto Jap, Jakub Breier, and Shivam Bhasin. Side channel attack on stream ciphers: A three-step approach to state/key recovery. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(2):166–191, 2022.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[KM22a] David Knichel and Amir Moradi. Composable Gadgets with Reused Fresh Masks First-Order Probing-Secure Hardware Circuits with only 6 Fresh Masks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(3):114–140, 2022.

[KM22b] David Knichel and Amir Moradi. Low-latency hardware private circuits. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 1799–1812. ACM, 2022.

[KMMS22] David Knichel, Amir Moradi, Nicolai Müller, and Pascal Sasdrich. Automated generation of masked hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):589–629, 2022.

[KSM20] David Knichel, Pascal Sasdrich, and Amir Moradi. SILVER - statistical independence and leakage verification. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I*, volume 12491 of *Lecture Notes in Computer Science*, pages 787–816. Springer, 2020.

[KSM22] David Knichel, Pascal Sasdrich, and Amir Moradi. Generic hardware private circuits towards automated generation of composable secure gadgets. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):323–344, 2022.

[KSPS13] Paris Kitsos, Nicolas Sklavos, George Provelengios, and Athanassios N. Skodras. Fpga-based performance analysis of stream ciphers zuc, snow3g, grain v1, mickey v2, trivium and E0. *Microprocess. Microsystems*, 37(2):235–245, 2013.

[KY10] Elif Bilge Kavun and Tolga Yalçin. A lightweight implementation of keccak hash function for radio-frequency identification applications. In Siddika Berna Örs Yalçin, editor, *Radio Frequency Identification: Security and Privacy Issues - 6th International Workshop, RFIDSec 2010, Istanbul, Turkey, June 8-9, 2010, Revised Selected Papers*, volume 6370 of *Lecture Notes in Computer Science*, pages 258–269. Springer, 2010.

[LBS22] Itamar Levi, Davide Bellizia, and François-Xavier Standaert. Tight-ES-TRNG: Improved Construction and Robustness Analysis. *SN Comput. Sci.*, 3(4):321, 2022.

[LLL20] Bohan Li, Meicheng Liu, and Dongdai Lin. FPGA implementations of grain v1, mickey 2.0, trivium, lizard and plantlet. *Microprocess. Microsystems*, 78:103210, 2020.

[LMMR21] Gregor Leander, Thorben Moos, Amir Moradi, and Shahram Rasoolzadeh. The SPEEDY Family of Block Ciphers Engineering an Ultra Low-Latency Cipher from Gate Level for Secure Processor Architectures. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):510–545, 2021.

[MAM16] Vasily Mikhalev, Frederik Armknecht, and Christian Müller. On ciphers that continuously access the non-volatile key. *IACR Trans. Symmetric Cryptol.*, 2016(2):52–79, 2016.

[MB07] Alexander Maximov and Alex Biryukov. Two trivial attacks on trivium. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *Selected Areas in Cryptography, 14th International Workshop, SAC 2007, Ottawa, Canada, August 16-17, 2007, Revised Selected Papers*, volume 4876 of *Lecture Notes in Computer Science*, pages 36–55. Springer, 2007.

[MCB$^+$22] Awaleh Houssein Meraneh, Christophe Clavier, Hélène Le Bouder, Julien Maillard, and Gaël Thomas. Blind side channel on the elephant LFSR. In Sabrina De Capitani di Vimercati and Pierangela Samarati, editors, *Proceedings of the 19th International Conference on Security and Cryptography, SECRYPT 2022, Lisbon, Portugal, July 11-13, 2022*, pages 25–34. SCITEPRESS, 2022.

[MCS22] Charles Momin, Gaëtan Cassiers, and François-Xavier Standaert. Handcrafting: Improving automated masking in hardware with manual optimizations. In Josep Balasch and Colin O'Flynn, editors, *Constructive Side-Channel Analysis and Secure Design - 13th International Workshop, COSADE 2022, Leuven, Belgium, April 11-12, 2022, Proceedings*, volume 13211 of *Lecture Notes in Computer Science*, pages 257–275. Springer, 2022.

[MM22] Nicolai Müller and Amir Moradi. PROLEAD A probing-based hardware leakage detection tool. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):311–348, 2022.

[MMSS19] Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. Glitch-resistant masking revisited or why proofs in the robust probing model are needed. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):256–292, 2019.

[MMW18] Lauren De Meyer, Amir Moradi, and Felix Wegener. Spin me right round rotational symmetry for fpga-specific AES. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):596–626, 2018.

[Moo19] Thorben Moos. Static power SCA of sub-100 nm CMOS asics and the insecurity of masking schemes in low-noise environments. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(3):202–232, 2019.

[MPG05] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-channel leakage of masked CMOS gates. In Alfred Menezes, editor, *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.

[MRB18] Lauren De Meyer, Oscar Reparaz, and Begül Bilgin. Multiplicative masking for AES in hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):431–468, 2018.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer, 2006.

[NRS08] Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure hardware implementation of non-linear functions in the presence of glitches. In Pil Joong Lee and Jung Hee Cheon, editors, *Information Security and Cryptology - ICISC 2008, 11th International Conference, Seoul, Korea, December 3-5, 2008, Revised Selected Papers*, volume 5461 of *Lecture Notes in Computer Science*, pages 218–234. Springer, 2008.

[oEiCE04] European Network of Excellence in Cryptology (ECRYPT). estream: the ecrypt stream cipher project. 2004.

[OGL05] Sean O'Neil, Benjamin Gittins, and Howard Landman. Vest - hardware-dedicated stream ciphers. 2005.

[oSN17] National Institute of Standards and Technology (NIST). Lightweight cryptography. 2017.

[PMB+16] Oto Petura, Ugo Mureddu, Nathalie Bochard, Viktor Fischer, and Lilian Bossuet. A survey of AIS-20/31 compliant TRNG cores suitable for FPGA devices. In Paolo Ienne, Walid A. Najjar, Jason Helge Anderson, Philip Brisk, and Walter Stechele, editors, *26th International Conference on Field Programmable Logic and Applications, FPL 2016, Lausanne, Switzerland, August 29 - September 2, 2016*, pages 1–10. IEEE, 2016.

[PP10] Christof Paar and Jan Pelzl. *Understanding Cryptography - A Textbook for Students and Practitioners.* Springer, 2010.

[PR13] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 142–159. Springer, 2013.

[PYR+16] Stjepan Picek, Bohan Yang, Vladimir Rozic, Jo Vliegen, Jori Winderickx, Thomas De Cnudde, and Nele Mentens. PRNGs for Masking Applications and Their Mapping to Evolvable Hardware. In Kerstin Lemke-Rust and Michael Tunstall, editors, *Smart Card Research and Advanced Applications - 15th International Conference, CARDIS 2016, Cannes, France, November 7-9, 2016, Revised Selected Papers*, volume 10146 of *Lecture Notes in Computer Science*, pages 209–227. Springer, 2016.

[Rad06] Havard Raddum. Cryptanalytic results on trivium. *eSTREAM, ECRYPT Stream Cipher Project, Report 2006/039*, 2006.

[RBN+15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings,* *Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 764–783. Springer, 2015.

[Rep15] Oscar Reparaz. A note on the security of higher-order threshold implementations. *IACR Cryptol. ePrint Arch.*, page 1, 2015.

[Rog07] Marcin Rogawski. Hardware evaluation of estream candidates: Grain, lex, mickey128, salsa20 and trivium. 2007.

[RSV09] Mathieu Renauld, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. Algebraic side-channel attacks on the AES: why time also matters in DPA. In *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 97–111. Springer, 2009.

[SBHM20] Pascal Sasdrich, Begül Bilgin, Michael Hutter, and Mark E. Marson. Low-latency hardware masking with application to AES. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):300–326, 2020.

[Sch96] Bruce Schneier. *Applied cryptography - protocols, algorithms, and source code in C, 2nd Edition.* Wiley, 1996.

[SM15] Tobias Schneider and Amir Moradi. Leakage Assessment Methodology - A Clear Roadmap for Side-Channel Evaluations. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, volume 9293 of *Lecture Notes in Computer Science*, pages 495–513. Springer, 2015.

[SM21] Aein Rezaei Shahmirzadi and Amir Moradi. Second-order SCA security with almost no fresh randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):708–755, 2021.

[SNO] Specification of the 3gpp confidentiality and integrity algorithms uea2 & uia2. document 2: Snow 3g specification. https://www.gsma.com/aboutus/wp-content/uploads/2014/12/snow3gspec.pdf. Accessed: 2023-07-15.

[SPY$^+$10] François-Xavier Standaert, Olivier Pereira, Yu Yu, Jean-Jacques Quisquater, Moti Yung, and Elisabeth Oswald. Leakage resilient cryptography in practice. In Ahmad-Reza Sadeghi and David Naccache, editors, *Towards Hardware-Intrinsic Security - Foundations and Practice*, Information Security and Cryptography, pages 99–134. Springer, 2010.

[SSD19] Shravani Shahapure, Virendra Sule, and Rohin D. Daruwala. Internal state recovery attack on stream ciphers: Breaking BIVIUM. In Shivam Bhasin, Avi Mendelson, and Mridul Nandi, editors, *Security, Privacy, and Applied Cryptography Engineering - 9th International Conference, SPACE 2019, Gandhinagar, India, December 3-7, 2019, Proceedings*, volume 11947 of *Lecture Notes in Computer Science*, pages 34–49. Springer, 2019.

[TIM$^+$18] Yosuke Todo, Takanori Isobe, Willi Meier, Kazumaro Aoki, and Bin Zhang. Fast correlation attack revisited - cryptanalysis on full grain-128a, grain-128, and grain-v1. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 129–159. Springer, 2018.

[UHM$^+$20] Rei Ueno, Naofumi Homma, Sumio Morioka, Noriyuki Miura, Kohei Matsuda, Makoto Nagata, Shivam Bhasin, Yves Mathieu, Tarik Graba, and Jean-Luc Danger. High throughput/gate AES hardware architectures based on datapath compression. *IEEE Trans. Computers*, 69(4):534–548, 2020.

[WSLM05] Doug Whiting, Bruce Schneier, Stefan Lucks, and Frederic Muller. Phelix: Fast encryption and authentication in a single cryptographic primitive. 2005.

[YRG+18] Bohan Yang, Vladimir Rozic, Milos Grujic, Nele Mentens, and Ingrid Verbauwhede. ES-TRNG: A High-throughput, Low-area True Random Number Generator based on Edge Sampling. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):267–292, 2018.

[YSPY10] Yu Yu, François-Xavier Standaert, Olivier Pereira, and Moti Yung. Practical leakage-resilient pseudorandom generators. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, pages 141–151. ACM, 2010.

[ZUC] Specification of the 3gpp confidentiality and integrity algorithms 128-eea3 & 128-eia3. document 2: Zuc specification. `https://www.gsma.com/aboutus/wp-content/uploads/2014/12/eea3eia3zucv16.pdf`. Accessed: 2023-07-15.