SCARF – A Low-Latency Block Cipher for Secure Cache-Randomization

Federico Canale[1], Tim Güneysu[1,4], Gregor Leander[1], Jan Philipp Thoma[1], Yosuke Todo[2], and Rei Ueno[3]

[1] Ruhr University Bochum, Bochum, Germany firstname.lastname@rub.de
[2] NTT Social Informatics Laboratories, Tokyo, Japan yosuke.todo.xt@hco.ntt.co.jp
[3] Tohoku University, Sendai-shi, Japan. rei.ueno.a8@tohoku.ac.jp
[4] DFKI, Bremen, Germany.

Abstract. Randomized cache architectures have proven to significantly increase the complexity of contention-based cache side channel attacks and therefore present an important building block for side-channel secure microarchitectures. By randomizing the address-to-cache-index mapping, attackers can no longer trivially construct minimal eviction sets which are fundamental for contention-based cache attacks. At the same time, randomized caches maintain the flexibility of traditional caches, making them broadly applicable across various CPU-types. This is a major advantage over cache partitioning approaches.

A large variety of randomized cache architectures has been proposed. However, the actual randomization function received little attention and is often neglected in these proposals. Since the randomization operates directly on the critical path of the cache lookup, the function needs to have extremely low latency. At the same time, attackers must not be able to bypass the randomization which would nullify the security benefit of the randomized mapping. In this paper we propose SCARF (Secure CAche Randomization Function), the first dedicated cache randomization cipher which achieves low latency and is cryptographically secure in the cache attacker model. The design methodology for this dedicated cache cipher enters new territory in the field of block ciphers with a small 10-bit block length and heavy key-dependency in few rounds.

1 Introduction

In the recent past, we have witnessed a significant increase in attacks on the microarchitectural level of widely deployed desktop- and server-grade CPUs for which side-channel attacks on caches play an important role. By measuring the latency of a memory access, attackers can observe if the access was served from the cache or from main memory. This ability has been exploited to leak secret keys from many cryptographic algorithms, among others, including the widely used encryption schemes AES [6,35] and RSA [34,56]. Cache-based keyloggers that observe the activity of keystroke handlers in the cache to leak user input on co-located VMs have been presented in [56,17]. Moreover, cache side-channel attacks are a commonly used building block for speculative execution attacks like Spectre [20] and Meltdown [26]. A variety of cache attack primitives including FLUSH+RELOAD [56] and PRIME+PROBE [35,50] have been proposed. Using these primitives, attackers can reliably observe the access behavior of a victim process and hence, leak sensitive information like secret key material. Cache side-channel attacks can be categorized in two distinct groups. The first group, flush-based attacks like FLUSH+RELAOD [56] and FLUSH+FLUSH [17], require shared memory between the attacker and the victim process. Furthermore, the attacker must be able to flush targeted entries from the cache using a special instruction. Such attacks can be prevented by either making the flushing instruction privileged on the ISA level, or by duplicating shared memory addresses in the cache [37]. The second group, contention-based cache attacks, are much harder to prevent. Those attacks exploit the internal structure of modern caches that is essential to their performance.

In an effort to design side-channel secure cache architectures that prevent contention-based attacks, two approaches have emerged: cache partitioning and cache randomization. The former splits the cache into n distinct partitions for different security domains to avoid leakage across domain boundaries [36]. The main disadvantage of partitioning is the limited flexibility and therefore large performance overhead. For dynamically partitioned caches, adjusting the size of partitions has shown to be difficult under security considerations [53]. As a consequence, randomization-based cache designs have received more attention in response to such attacks [54,38,39,55,45,41,43,47]. These designs randomize the address-to-cache-index mapping and therefore prevent the attacker from efficiently finding addresses that contend for cache entries with the victim address. All these designs use a randomization function that takes the memory address as input and returns a set of pseudorandom cache indices. The exact instantiation of the randomization function varies between the schemes for which designers carefully consider performance interests versus security requirements. Since the randomization process is within the critical path of the cache lookup, low latency is obviously extremely important. At the same time, if the attacker can construct conflicting pairs of addresses, the security of the scheme collapses and PRIME+PROBE-like attacks become feasible again [37]. A conservative choice in terms of security is to use a low-latency block cipher like PRINCE [12] as proposed in [55,41,47]. However, full encryption of the address using a 64-bit block cipher is not ideal for two reasons: First, a 64-bit block cipher introduces a significant storage overhead in the tag computation since the address contains offset bits that must not be used for the randomization. Second, the attacker never observes the ciphertext since the cache functions as a black-box; i.e., the attacker will never observe the actual output of the randomization function. The only opportunity for an attacker to learn something about the randomization function is when two addresses map to the same randomized index. Hence, the randomization could in principle be simpler than a full block cipher. However, previous work has demonstrated that randomization functions with insufficient cryptographic properties can easily be broken by an attacker [9]. Specifically, the Feistel-structure proposed to be used in CEASER [38] has shown to be insufficient for secure randomization. Bodduna *et al.* [9] demonstrate an attack on the randomization scheme and conclude: *"This [attack] opens up a need for specialized low-latency encryption techniques that are exclusively designed for cache address encryption, that can provide the security guarantees with acceptable performance overheads."* Similarly, Prunal *et al.* [37] state that *"it is still an open challenge to choose a strong and fast encryption algorithm for randomized caches."*

In this paper we present SCARF, the first cryptographically sound, tailor-made cache cipher. We define the attacker model for cache randomization functions and analyze and discuss design requirements. We carefully design SCARF to minimize the latency and thoroughly evaluate its security thanks to our new framework. We evaluate the latency and area requirements of SCARF on ASIC hardware through logic synthesis with the 45 nm and 15 nm Nangate open cell libraries (OCLs). Consequently, we confirm that SCARF achieves a latency less than half of that of PRINCE (the pioneering and most major low-latency block cipher), Mantis, and QARMA (state-of-the-art low-latency tweakable block ciphers). In addition, we quantify the performance of SCARF in comparison to the above low-latency (tweakable) block ciphers in the cache setting using the gem5 simulator [29].

1.1 Related Work

Different randomized cache architectures have been presented in [54,38,39,55,45,41,43,47]. Purnal *et al.* generalize the idea of randomized cache architectures in [37] and present a generic algorithm to construct generalized eviction sets for solely randomization-based caches. Several designs [55,41,47,9] use the PRINCE [12] block cipher for randomization. The authors of PhantomCache [45] introduce a randomization function based on Toeplitz hashes [22] but do not investigate the security properties of this function in the cache application. The randomized cache architecture CEASER presented a custom low-latency randomization function for their cache design [38]. However, the proposed randomization function did not contain nonlinear functions. An attack on the randomization function of CEASER has been presented by Bodduna *et al.* [9]. Ribes-González *et al.* [40] formally define security properties of randomized caches. For the formal proofs, they assume an abstract randomization function based on an PRF.

Format-preserving encryption algorithms have been presented in [5,4,42]. These encryption algorithms map a given plaintext to a ciphertext of same length and hence, preserve the format. However, these schemes usually do not target low latency use cases like SCARF. The K-Cipher has been presented by Kounavis *et al.* in [21] and allows encryption with arbitrary ciphertext length between 24 and 1024 bit. The K-cipher has been used in the context of memory safety in [25]. Recent analysis has revealed a practical key-recovery attack of the K-Cipher, thus rendering it insecure [30].

1.2 Organization of the Paper

The remainder of the paper is organized as follows: In Section 2, we introduce background on caches, cache randomization, and block ciphers. Moreover, we introduce and justify the parameter choice on which this paper is based. Section 3 introduces the attacker model. In Section 4, we introduce SCARF. The following Section 5 discusses the design rationale of SCARF. We summarize the security of our design in Section 6 and details on the extensive security analysis we performed in the appendix. The results of the performance evaluations, both in hardware as well as on a system level, are given in Section 7 before we conclude in Section 8.

2 Background and Requirements

In this section we introduce background on caches and cache attacks. We provide reasoning for the parameter choice of SCARF and introduce the concept of cache randomization as a countermeasure.

2.1 Caches

Due to the large performance gap between the CPU and main memory, modern CPUs store frequently accessed data in small memory modules in close physical proximity to the core. Most modern processors divide this cache memory into multiple levels ranked from the smallest and fastest L1 cache to the largest and slowest L3 cache. Each physical core is equipped with private L1 and L2 caches whereas the L3 cache is typically shared among all CPU cores. For every memory access, the respective cache levels are queried for the accessed address. If the data associated with the requested address is stored in the cache, a cache *hit* occurs and it is returned directly. In this case, the CPU does not need to wait for the main memory. If the data is not cached, a cache *miss* occurs. Such a cache miss leads to significantly slower access times which is measurable from user level code. To determine if the data belonging to a given address is cached, part of the address is stored as a *tag* alongside the data. Upon access, the cache is searched for the tag and the corresponding data is returned if the access resulted in a hit. For small L1 caches, one can simply search the entire cache for the given tag upon access. Caches implementing this search strategy are called *fully-associative*. For larger caches like the L2 and L3 cache which can often store multiple megabytes of data, searching the entire cache upon access is not feasible for performance reasons. Therefore, most caches deployed in real CPUs use a *set-associative* addressing scheme.

The set-associative layout can be imagined like a table structure with m byte *entries*. The table rows are called *sets* and the columns are called *ways*. The accessed address is split into a *tag*, an *index*, and an *offset*. The offset is $log_2(m)$ bits and selects which word within the m-byte entry is returned. The index bits select the cache set (i.e., the table row) in which the entry is placed. Finally, the tag bits are stored alongside the data and are used in combination with the implicit index to uniquely identify the address. When an address is accessed, all cache ways at the index of the requested address are searched for the corresponding tag. On a cache hit, the correct data is returned. If a cache miss occurs, the data is loaded from memory or from other cache levels. In this case, a *replacement policy* is used which selects one entry from the cache set to be evicted and replaced by the new data. In many cases, this replacement policy selects the least-recently used (LRU) entry for replacement.

2.2 Cache Attacks and Randomization

The timing difference between a cache hit and a cache miss is easily measurable from user level code and in combination with the set-associative structure allows attackers to observe the cache behavior of processes that operate on the CPU in parallel. Cache attacks can be divided into flush-based attacks [56,17] and contention-based attacks [35,50]. For flush-based attacks, the attacker requires a shared memory address between himself and the victim process. This may for example be a library function. Then, the attacker leverages a special cache-line flush instruction (e.g., `clflush` in x86) to evict the shared address from the cache. Flush-based attacks can be prevented by duplicating shared memory in the cache as proposed by Werner *et al.* [55]. Contention-based attacks are much more challenging to prevent since they directly exploit the set-associative structure of modern caches. Here, the attacker constructs an *eviction set*, i.e., a set of addresses that map to the same cache set as the victim address. In a w-way cache, an eviction set with exactly w addresses is called minimal. By accessing the eviction set addresses, the attacker *primes* the cache set – that is, all addresses that were stored in the cache set prior to the access are replaced by addresses from the eviction set. When the victim address is accessed, one of the eviction set addresses must be evicted by the replacement policy. The attacker can then probe whether the victim address was accessed by measuring the access latency of the eviction set addresses. If accessing one of these addresses results in a cache miss, the victim address was accessed. This exact technique is used in the PRIME+PROBE attack [35,50] which was, among others, used to leak GnuPG keys from the shared last-level cache [28].

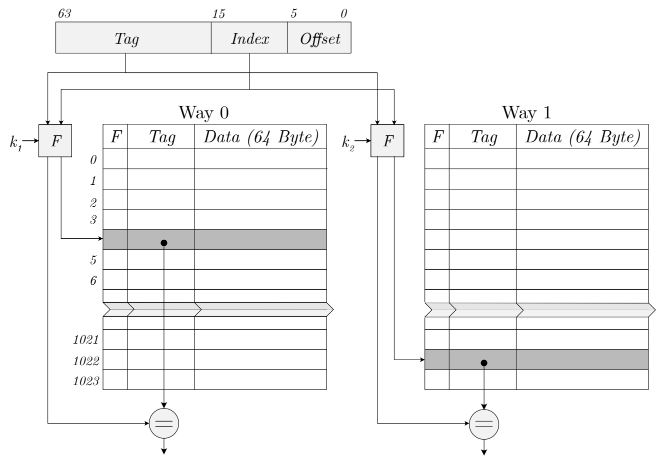


Fig. 1. Schematic overview of a 2-way randomized cache architecture with 10-bit indices.

Even though the addresses used for cache addressing are not directly visible to the attacker due to the virtual memory abstraction, constructing minimal eviction sets can be done very efficiently for set-associative caches [51,44,46]. By randomizing the address-to-cache-index mapping in each cache way separately, the concept of eviction sets can be weakened [38,39,55,45,41,43,47] and the effort to construct eviction sets increases significantly [37]. Addresses that only differ in the (usually 6) offset bits must still map to the same cache indices to ensure cache coherency. Therefore, the offset bits of the address are not considered for randomization. The index-width depends on the number of available cache sets. Many recent CPUs feature 2^{10} cache sets, and hence, an index width of 10 bits [46]. By using different number of cache ways and instantiating multiple cache slices (load balanced parallel caches) CPU designers can still create arbitrary sized caches with a fixed number of cache sets. For the remainder of this paper, we consider caches with an index width of 10 bits. Our approach and results can be used as groundwork for further cache randomization functions with different index-width. A schematic overview of a randomized cache is shown in Figure 1. The randomization is applied to the address in each cache way. This way it is possible to have different mappings in each way by selecting a unique random key. Since the cache can only be searched for the queried address after the randomization function is complete, the latency of the randomization function is crucial both in the hit-, and the miss scenario. At the same time it must not be possible for an attacker to break the randomization since in this case, the construction of eviction sets becomes trivial.

In [37], Purnal *et al.* present PRIME+PRUNE+PROBE, a generic attack on randomized cache architectures. It applies to cache architectures that purely rely on randomization as a protection mechanism, e.g, SCATTERCACHE [55]. By priming the cache with a set of addresses k, the attacker is able to observe conflicts between addresses from k and the victim address. Then, they construct a generalized eviction set G which contains addresses that collide with the victim address in at least one cache way. The generalized eviction set can be used to evict the victim address with probability p_e and can therefore be used similarly to a traditional eviction set. The profiling effort of PRIME+PRUNE+PROBE is significant and hence, frequent re-keying of pure randomization designs like SCATTERCACHE [55] can prevent attackers from being able to construct generalized eviction sets. More recent randomized cache architectures use additional measures to increase the complexity of PRIME+PRUNE+PROBE attacks beyond feasible boundaries [41,47].

2.3 (Tweakable) Block Ciphers

A block cipher takes two inputs, a plaintext $P \in \mathbb{F}_2^n$ and a key $K \in \mathbb{F}_2^k$, and produces a ciphertext $C \in \mathbb{F}_2^n$. A tweakable block cipher is a cryptographic primitive that extends block ciphers [27] by allowing an additional input $T \in \mathbb{F}_2^t$ (called tweak) that, along with the plaintext P and the key K, produces the ciphertext C. The idea is that a tweakable block cipher is a family of independent block ciphers, one for every tweak T. Many dedicated tweakable block ciphers have recently been proposed, such as Skinny and Mantis [3], Deoxys [18], and QARMA [1].

2.4 Design Rationales

For cache randomization, we need to map an address consisting of a t-bit tag-, an i-bit index-, and an o-bit offset to a pseudorandom i-bit randomized index. Since caches are spread over a huge amount of CPU classes, ranging from small embedded devices over smartphones and desktop PCs to high-end server clusters, the sizes and parameters of caches can vary significantly. Therefore, there cannot be one randomization function that suits all caches perfectly. We chose to design SCARF targeting the parameters of recent desktop-grade CPUs.

SCARF is a tweakable block cipher where the tag is used as tweak and the index is used as plaintext. Since addresses that only differ in the offset bits must map to the same randomized index, the offset bits are neglected for the randomization. For the remainder of the paper, we denote addresses as (x, T), where x is the index part and T is the tag part of the address. Our cipher uses a small block size of 10 bits - since as motivated earlier many recent CPUs feature 2^{10} cache sets, and hence, $i = 10$. Therefore, we achieve a format-preserving encryption of the index while retaining the original 48-bit tag. The offset that is typically 6 bits ($log_2(64)$) is discarded. In Figure 1, SCARF would be used as F-function with the tag used as tweak and the index used as plaintext.

Low latency is a key requirement for cache randomization. This holds especially for the encryption, since the randomization function is applied on the critical path of every cache access. The index of the accessed address is encrypted using SCARF and the result is used for the cache lookup. The latency of the decryption function is less critical since it is only used to write back dirty entries from the cache to main memory. This usually happens on a cache miss where the CPU needs to wait for the slow main memory to respond before the cache entry can be replaced. The decryption of the index (and therefore, the reconstruction of the stored address) can be executed in parallel to the memory access.

Opposed to a traditional 64-bit block cipher, our design approach has three key advantages: (i) we avoid overhead in the ciphertext resulting from the discarded offset bits, (ii) the construction allows more latency focused designs, and (iii) it enables a much more elegant attacker model as described in the following section.

Our cipher design features a nominal security level of 80 bits (even though the key used is 240 bits).

Hence, an attacker must perform at least 2^{80} encryptions or decryptions for a successful attack in the given attacker model (see Section 3). The 80-bit security level is often taken as a practical complexity limit for brute force attacks on modern hardware, although a higher security level is required for general purpose block ciphers to cover future developments. Furthermore, we limit the data complexity by the attacker to 2^{40}. Since each unique address (ignoring the offset bits) maps 64 bytes of memory on the device under attack, these limitations are irrelevant for the cache use-case. 2^{40} addresses would map about 70 terabytes of RAM which is far beyond current system configurations. Hence, the attacker is constrained by the available addresses for attacks on the randomization key and the chosen limits leave a healthy security margin.

3 Attacker Model

One of the most interesting aspects of the problem we are facing is actually the attacker model. From a system-perspective, we model the cache as a black-box which the attacker can query with arbitrary physical addresses. For each access, the attacker can observe based on timing whether it resulted in a cache hit or cache miss. Moreover, we assume that two addresses are sufficient for the attacker to tell if they map to the same cache set. In reality, the attacker would have to find w addresses that map to the same cache set before observing this. The attacker cannot observe other cache internals which especially include the set-index of a given address. From a cryptographic point of view, this corresponds to an attacker that is able to choose plaintexts (index part of the address) and tweaks (tag part of the address) to be encrypted. However, ciphertexts will not be revealed. The only information the attacker is able to learn is in the collisions in the ciphertexts. Hence, we must ensure that an attacker cannot learn how to construct conflicting addresses from observing random conflicts. More formally, we require the following security property for SCARF:

Security Requirement 1 *Let O be the oracle that takes addresses* (x_1, T_1) *and* (x_2, T_2)*, and returns* 1 *if* $E_{T_1}(x_1) = E_{T_2}(x_2)$ *and* 0 *otherwise.*

Given a challenge address (x_v, T_v)*, an adversary is allowed to make at most* 2^{40} *queries to O. Let* $\mathcal{X}$ *be the set of queried (single) addresses. Then, the adversary is unable to output* $(x_r, T_r) \notin \mathcal{X}$ *that satisfies* $E_{T_v}(x_v) = E_{T_r}(x_r)$ *with probability significantly larger than*

$$\frac{1}{|\{x \mid \{(x, T_r)\} \notin \mathcal{X}\}|}$$

and in time significantly smaller than 2^{80} *evaluations of* E_T *for* $T_r \neq T_v$*.*

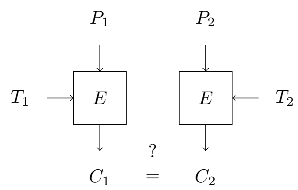
In other words, the attacker does not have significant advantage over guessing when the data complexity and the time complexity are restricted as stated above.

The security requirement relates to contention-based cache attacks as follows: the attacker wants to find addresses (x, T) that collide with a given target address (x_v, T_v) in the cache. Therefore, in a profiling phase, the attacker may query arbitrary addresses and observe addresses that collide in the randomized cache index. Finally, the attacker is challenged to output a new address (x_r, T_r) which has not been part of a query to the oracle (i.e., it cannot belong to $\mathcal{X}$). Security Requirement 1 ensures that the attacker cannot do this with a significant advantage over guessing addresses. This prevents the attacker from constructing eviction sets more efficiently than in the generic PRIME+PRUNE+PROBE attack [37].

Notice that in reality, Security Requirement 1 is more complex since the oracle memorizes past queries and the attacker can learn small bits of information that are not covered in the formal requirement. For example, the attacker can access two distinct sets of addresses that do not contain conflicting addresses within each set. By accessing first Set 1, then Set 2, and then Set 1 again, the attacker can learn if there are addresses from Set 2 that collide with addresses from Set 1. Doing this, in general they do not learn *which exact* address pairs collide. On the other hand, there are factors that reduce the success probability of the attacker which outweigh the small benefit the attacker can gain from the oracle memory. Most severely is the fact that caches feature multiple ways. Hence, the attacker only learns if two addresses collide with some probability determined by the replacement policy. If the oracle returns zero, the addresses may still collide in a different cache way, but the attacker does not learn that information. Similarly, in a real cache attack, the attacker does not have full control over the addresses that are queried since the address bits above the page offset (usually Bit 12 upwards) are determined by the operating system and not by the attacker. This limits the control of the attacker over the queried tags. Finally, on real systems there is always some noise from concurrent processes. Hence, even if the attacker observes an address collision, he cannot be certain about it. This effect increases if the attacker queries large sets of addresses at once since the probability that such a set is affected by noise increases. The security requirement underestimates the information provided by the oracle (e.g., the memory aspect) but it also overestimates the abilities of the attacker (e.g., choosing addresses and telling apart colliding from non-colliding addresses with only two addresses). We designed Security Requirement 1 such that in practice, the overestimation of the attacker outweighs the simplified oracle. This allows us to analyse the security of SCARF w.r.t. Security Requirement 1, yielding a secure cipher in real-world environments.

From a designer perspective, Security Requirement 1 does not allow to use the many well-understood cryptanalysis arguments and techniques that have led to the design of modern block ciphers. Therefore, we now address this problem by framing the attacker setting in the cache scenario in a novel and convenient way. In fact, by observing a random conflict, the attacker can learn if two addresses collide in the cache, i.e., if two plaintexts P_1 and P_2 and two tweaks T_1 and T_2 lead to the same ciphertext and hence satisfy

$$E_{T_1}(P_1) = E_{T_2}(P_2).$$



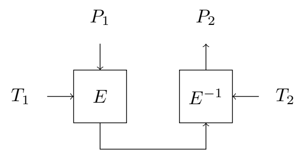
Then, and this is a key point of our work, P_2 is actually the decryption of $C_1 = C_2$ under T_2, where $C_i = E_{T_i}(P_i)$ $(i = 1,2)$. Indeed, the attacker learns the evaluation of the function

$$E_{T_2}^{-1} \circ E_{T_1}(P_1) = P_2$$

in case of a collision and learns that

$$E_{T_2}^{-1} \circ E_{T_1}(P_1) \neq P_2$$

in case there is no collision. In other words, we can turn the attacker's view actually into the following view that will guide our design approach.



So, as designers, we simplify this situation by assuming that the attacker is allowed to query

$$\tilde{E}_{T_1,T_2}(P) := E_{T_2}^{-1} \circ E_{T_1}(P) = C$$

directly for chosen P, and tweak pairs T_1, T_2. Note that, in practice, querying this function actually requires the attacker to perform a non-negligible amount of work by basically randomly searching for those collisions.

Assume we design E as an iterative function using r rounds. The great advantage of this attacker model and the designer's view is that we have to implement, and consider the latency of r rounds, while the attacker actually faces a primitive consisting of $2r$ rounds. This, among other ideas and insights described below, is the reason why SCARF enables security with an exceptionally small latency.

Note that, even if E_T is an ideal tweakable block cipher, $\tilde{E}_{T_1,T_2}$ is not. The easiest way to see that is to note that there is an important special choice of the tweak pair. Indeed

$$\tilde{E}_{T_1,T_1}(P) = P$$

for all P, i.e., for identical tweaks, the function is the identity. While this constitutes weak-tweaks for the tweakable block cipher $\tilde{E}$, it actually does not correspond to any knowledge the attacker gains as this just means that the same plaintext with the same tweak always yields the same ciphertext using E.

There are more examples of non-ideal behaviour of $\tilde{E}_{T_1,T_2}$, e.g., it holds for all T_1, T_2, T_3 that

$$\tilde{E}_{T_3,T_1} \circ \tilde{E}_{T_2,T_3} \circ \tilde{E}_{T_1,T_2}$$

is the identify function.

The Security Requirement 1 can then be translated into the following security requirement for $\tilde{E}$.

Security Requirement 2 *Let $\widetilde{O}$ be the encryption oracle of $\tilde{E}$ that takes a plaintext P and a pair of tweaks T_1, T_2 as input and returns C such that $C = \tilde{E}_{T_1,T_2}(P)$.*

Given a challenge (P_v, T_v), an adversary is allowed to make at most 2^{40} queries to $\widetilde{O}$. Let $\mathcal{X} \subseteq \mathbb{F}_2^{10} \times \mathbb{F}_2^{48}$ be the set containing all pairs (P, T_1) and (C, T_2) for queries $P, (T_1, T_2)$ made to the oracle.

Then, the adversary is unable to output $(C_r, T_r) \notin \mathcal{X}$ that satisfies $C_r = E_{T_v,T_r}(P_v)$, with probability significantly larger than

$$\frac{1}{|\{x \mid \{(x, T_r)\} \notin \mathcal{X}\}|}$$

and in time significantly less than 2^{80} evaluations of E_T and $T_r \neq T_v$.

Both (P, T_1) and (C, T_2) for the query (P, T_1) and T_2 are added to $\mathcal{X}$, i.e., $\mathcal{X}$ is the equivalent of the set of queries in Security Requirement 1. Notice that $\widetilde{O}$ is a stronger oracle than O, but that it is possible to construct the oracle $\widetilde{O}$ from O. For each query to $\widetilde{O}$, the oracle needs to query multiple addresses to O. Thus, given that in both cases the query complexity is limited to 2^{40}, Security Requirement 2 is actually a stronger requirement than Security Requirement 1.

Why Existing Ciphers Are Not Enough. If we use a traditional block cipher to encrypt both the address and tag instead of the tweakable block cipher, the above simplification is no longer possible. In fact, a cache hit would then imply only a partial collision of the ciphertexts, so that we cannot model the target cipher as encryption-then-decryption as in the tweakable case. Thus, we conclude that the security of a round-reduced version of a secure block cipher like PRINCE is questionable without a careful analysis.

Finally note that, as a matter of fact, designing a secure tweakable block cipher as $E_{T_2}^{-1} \circ E_{T_1}$, where E_T has r rounds, is more challenging than designing a secure $2r$-round tweakable block cipher. Even if we would design a secure tweakable block cipher, it is unlikely that a half-round reduced version would yield a suitable solution for our setting. The target cipher in our attack model must have the structure $E_{T_2}^{-1} \circ E_{T_1}$, and without taking special care, it is likely that tweaks can be chosen by the attacker such that the last rounds of E_{T_1} are canceled by the first rounds of $E_{T_2}^{-1}$.

4 SCARF

We now present SCARF (Secure Cache Randomization Function), a tweakable block cipher with 48-bit tweak and 10-bit block size. SCARF uses a 240-bit secret key. An overview of SCARF is shown in Figure 2. The cipher consists of a tweakey schedule and a data encryption path. Before giving the detailed specification of the design, we discuss the security we expect from SCARF. Reference implementations are available at `https://anonymous.4open.science/r/SCARF-D434/README.md`[5].

4.1 Security Claims

We claim that SCARF satisfies Security Requirement 1 against any adversary running in time at most 2^{80} using at most 2^{40} queries to O. We also claim that SCARF satisfies Security Requirement 2 against any adversary running in time at most 2^{80} using at most 2^{40} queries to $\widetilde{O}$. Note that the latter is a significantly stronger claim for the actual use case of SCARF because the real attacker never gets corresponding plaintexts with a query complexity 1. The purpose of the claim is to encourage cryptanalysis and gain a better understanding of the security of SCARF.

We do not claim security against related- or known-key attacks because (i) these attacks do not apply to the cache randomization use case for which SCARF is designed and (ii) should be irrelevant for any properly used tweakable block cipher. Indeed, there is no situation where SCARF is used with multiple keys with a specific relation simultaneously, and the 240-bit secret key is always chosen at random.

The key is maintained by the hardware which is responsible for storing it in a secure manner. Due to the dense packaging of modern CPUs, physical side-channel attacks (e.g., power or EM) on SCARF are immensely difficult to carry out and therefore considered out of scope for this work. Moreover, while recent CPUs feature software-level voltage monitoring [33], the values are not actually measured but instead extrapolated from the CPU load. Hence, these measurements cannot leak information about the SCARF key.

4.2 Specification of SCARF

[5] Anonymous Git repo for double-blind review.

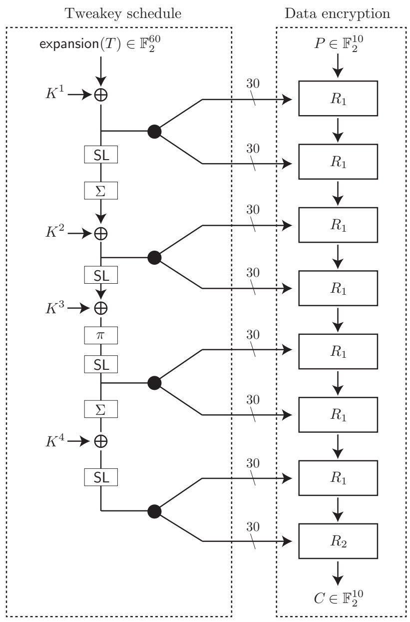


Fig. 2. Overview of SCARF

The Round Function R_1 and R_2. The round function R_1 has a 10-bit input x and a 30-bit subkey k generated by the tweakey schedule. The input x is divided into two halves, i.e., $x = x_L \| x_R$ of 5 bit each. The subkey k is also divided into six 5-bit values as $k = k_6 \| k_5 \| k_4 \| k_3 \| k_2 \| k_1$. Let τ_i be an i-bit left rotation, i.e., $\tau_i(x) = x \lll i$. Then, the round function R_1 updates (x_L, x_R) as follows:

$$\begin{aligned} y &= G(x_L, k_1, k_2, k_3, k_4, k_5) \oplus x_R, \\ x_R &= S(x_L \oplus k_6), \\ x_L &= y, \end{aligned}$$

where G is

$$G(x, k_1, k_2, k_3, k_4, k_5) = \left[\bigoplus_{i=0}^{4} (\tau_i(x) \wedge k_{i+1}) \right] \oplus (\tau_1(x) \wedge \tau_2(x))$$

and S is

$$S(x) = \Big((\tau_0(x) \vee \tau_1(x)) \wedge (\overline{\tau_3(x)} \vee \overline{\tau_4(x)}) \Big) \oplus \Big((\tau_0(x) \vee \tau_2(x)) \wedge (\overline{\tau_2(x)} \vee \tau_3(x)) \Big).$$

The round function R_2 is a slight variation of R_1. Specifically, the order of applying the S-box and XORing the subkey is swapped, and the last swap is omitted. The round

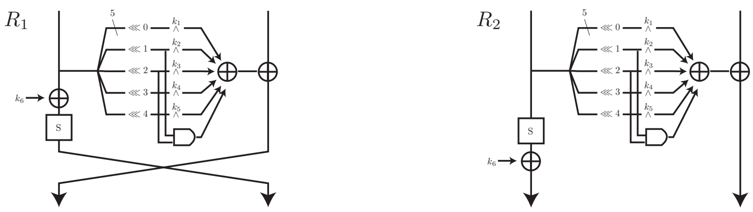


Fig. 3. Function $R_1(x,k)$ and $R_2(x,k)$

functions are depicted in Figure 3.

$$x_R = G(x_L, k_1, k_2, k_3, k_4, k_5) \oplus x_R,$$
$$x_L = S(x_L) \oplus k_6.$$

The Tweakey Schedule. The tweakey schedule generates four 60-bit subkeys T^i from a 48-bit tweak T and a 240-bit secret key $K^4 \| K^3 \| K^2 \| K^1$:

$$T^1 = \mathsf{expansion}(T) \oplus K^1,$$
$$T^2 = \Sigma(\mathsf{SL}(T^1)) \oplus K^2,$$
$$T^3 = \mathsf{SL}(\pi(\mathsf{SL}(T^2) \oplus K^3)),$$
$$T^4 = \mathsf{SL}(\Sigma(T^3) \oplus K^4),$$

each subkey T^i is split into two parts of 30 bits. Those 30 bits are then used as actual round keys in two consecutive rounds, e.g., T^1 provides the round keys for rounds 1 and 2.

In the following, the bits of the states $T[i]$ are also labeled starting from 1, from right to left. The tweakey schedule first expands the 48-bit tweak to a 60-bit value as follows:

$$\begin{aligned}\mathsf{expansion}(T) = & 0 \parallel T[48] \parallel T[47] \parallel T[46] \parallel T[45] \parallel \\ & 0 \parallel T[44] \parallel T[43] \parallel T[42] \parallel T[41] \parallel \cdots \parallel \\ & 0 \parallel T[4] \parallel T[3] \parallel T[2] \parallel T[1].\end{aligned}$$

The function SL applies 12 identical 5-bit S-boxes S in parallel, where S is the same S-box as in the round function. The function Σ is a linear function defined as

$$\Sigma(x) = x \oplus \tau_6(x) \oplus \tau_{12}(x) \oplus \tau_{19}(x) \oplus \tau_{29}(x) \oplus \tau_{43}(x) \oplus \tau_{51}(x),$$

and the function π is a bit permutation, where x_i is mapped to x_{p_i} and p_i is represented as:

$$\begin{aligned} p =&1,6,11,16,21,26,31,36,41,46,51,56,\\ &2,7,12,17,22,27,32,37,42,47,52,57,\\ &3,8,13,18,23,28,33,38,43,48,53,58,\\ &4,9,14,19,24,29,34,39,44,49,54,59,\\ &5,10,15,20,25,30,35,40,45,50,55,60. \end{aligned}$$

Finally, rk_i denotes a subkey for the i-th round.

$$rk_2 \| rk_1 = T^1, \qquad rk_4 \| rk_3 = T^2,$$
$$rk_6 \| rk_5 = T^3, \qquad rk_8 \| rk_7 = T^4.$$

5 Design Rationale

We provide details on the design rationale of SCARF.

5.1 Overall Structure

SCARF is a dedicated tweakable block cipher for randomizing the cache. It was designed only for this specific use case and achieves a substantial improvement of the latency compared not only to common block ciphers like AES, but also to existing low-latency block ciphers like PRINCE or QARMA. We first present two exclusive design philosophies on which SCARF is built.

Very Short Block Length. One of the essential differences between SCARF from a common block cipher is its short block length. To the best of our knowledge, SCARF is the first block cipher with such a short block length. The small block length makes SCARF well suited for cache randomization but not for other classical uses, e.g., as a symmetric-key encryption scheme or an authenticated encryption.

We first discuss the choice of the structure for the design of a block cipher. There are two most prominent structures: Substitution-Permutation-Network (SPN) and Feistel structures. AES is an example of a SPN cipher, while DES is an example of a Feistel cipher. Many low-latency block ciphers such as PRINCE [11] or MANTIS [3] adopt the SPN structure. Indeed, the SPN is suited to the low-latency design because it updates the entire state nonlinearly with a simultaneous operation. However, it usually uses subkey XOR, which means that at most a number of subkey bits equal to the block length can be absorbed every round. This would be problematic for our purpose since the block length is only 10 bits. Furthermore, adding extra bijective functions to absorb more subkey bits would work against our low-latency goal. On the other hand, the Feistel structure only uses a half-block length nonlinear operation every round, thus having a generally higher latency than the SPN. However, it can absorb many subkey bits efficiently because a non-bijective nonlinear function can be used.

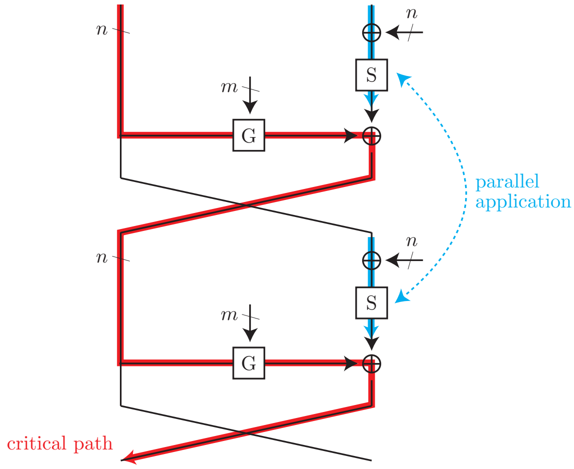


Fig. 4. Two-round structure of SCARF. The position of S equivalently moves from the left branch to the right branch due to an easy understanding of the parallel application of two S-boxes.

In light of the above observations, we adopted a new structure that combines the advantages of the SPN and Feistel structures. We can also view the new structure as a modification of the MISTY structure [32]. Figure 4 shows our two-round structure. Similar to the MISTY structure, two S-boxes can be computed in parallel. The main purpose of the G function is to result in high key-dependency, i.e., absorb many subkey bits with minimal latency. Note that the G function does not need to be bijective, which allows for a more flexible design.

One of the most interesting aspects of this structure is that the S-box does not lie in the critical path when the latency of the Sbox is smaller than the latency of two consecutive G functions. In fact, as depicted in Figure 4, it is the two G functions that make up the critical path in the encryption.

Taking the Attack Model into Account. Most modern block ciphers are designed under the assumption that plaintexts and ciphertexts can be observed by potential attackers. However, as motivated in Section 3, ciphertexts of E_T are not observable in the cache use case. As mentioned in Section 3, the target of an attacker using ciphertexts is then $\tilde{E}_{T_1,T_2} = E_{T_2}^{-1} \circ E_{T_1}$ and not E_T.

We designed the round function and the tweakey schedule to reflect this attack model. As explained above, special care has to be taken in order to avoid the internal cancellation of rounds. When the subkeys for the last round of E_{T_1} and E_{T_2} are the same, two rounds in the middle of $E_{T_2}^{-1} \circ E_{T_1}$ are cancelled by each other. We call this a *last-round cancellation*. We counter the last-round cancellations in two ways: first, we make finding such T_1, T_2 as *hard* as guessing the subkey involved in the computation of the last round function. For this, we adopted a nonlinear tweakey schedule, and the last subkey is generated as a "ciphertext", i.e., an encrypted tweak by the secret key. Secondly, SCARF guarantees that the last two rounds behave differently if the 60-bit subkeys are different. Note that the tweakey schedule always generates different 60-bit subkeys from different tweaks. In addition to the typical case like the last-round cancellation, as a general rule, the tweakey schedule should generate, on different tweaks, subkeys whose Hamming distance is as far as possible. Indeed, if the Hamming distance was close, the applied round function wold be almost the same, and it might cause many fixed points in $\tilde{E}$. To avoid this, the tweakey schedule guarantees that the Hamming distance of the subkeys generated by different tweaks is at least 17. Besides, since we guarantee that the differential characteristic probability is low enough in the tweakey schedule, it is not easy for the attackers to choose the subkey difference by controlling the tweak difference.

5.2 Design of the Round Function

The round function is the main component in the data encryption. The design is based on a modified MISTY structure that additionally has a low-latency G function absorbing many subkey bits.

The G function. The main goal of the G function is to absorb many subkey bits very quickly. It is well known that the AND / NAND gate has lower latency than XOR / XNOR gate. Thus, an initial idea for designing the G function is $G(x,k) = x \wedge k$, but it involves only 5-bit subkey. To absorb more subkey bits, we use bit rotations, AND gates, and their sum, i.e., $G(x,k_1,k_2,k_3,k_4,k_5) = \bigoplus_{i=0}^{4}(\tau_i(x) \wedge k_{i+1})$. It can absorb $5 \times 5 = 25$-bit subkey. To avoid G being the zero function for a specific choice of the subkey, we additionally XOR $\tau_1(x) \wedge \tau_2(x)$.

$$G(x,k_1,k_2,k_3,k_4,k_5) = (\tau_1(x) \wedge \tau_2(x)) \oplus \left[\bigoplus_{i=0}^{4}(\tau_i(x) \wedge k_{i+1})\right].$$

Then, G is described by the sum of six 5-bit values. Including XORing with x_R, it is described by the sum of seven 5-bit values. In other words, the critical path of G and XORing with x_R is one AND gate and an XOR tree of depth three.

The S-Box. The S-box is the main component to randomize the data nonlinearly. We design the S-box so that the critical path of the data encryption is still the iterative applications of the G function. We adopt the following design criteria, where we give more importance to the algebraic degree than to the linearity and differential uniformity when designing the S-box because the MISTY structure is potentially weak against the integral / higher-order differential attack [49]:

– The latency is competitive with two consecutive applications of XOR, so that the latency of S and XORing of the key is competitive with three consecutive XOR (that is, competitive with G). Thus, we can expect that this S-box is not on the critical path.
– Algebraic degree is 4.

In order to satisfy the above criteria, we have followed the ideas used for the design of the S-box of SPEEDY [24], and opted to search for S-boxes that are obtained by the composition of a OAI gate (that is, the gate that represents the logic function $(A,B,C,D) \mapsto \overline{(A \vee B) \wedge (C \vee D)}$) followed by an XOR, that is

$$S(x) = \overline{((\tau_a(x_0) \vee \tau_b(x_1)) \wedge (\tau_c(x_2) \vee \tau_d(x_3)))} \oplus \overline{((\tau_e(x_4) \vee \tau_f(x_5)) \wedge (\tau_g(x_6) \vee \tau_h(x_7)))}$$

for some $0 \leq a,b,c,d,e,f,g,h \leq 4$ and $x_i \in \{x, \bar{x}\}$. Among all S-boxes fulfilling those properties, we choose the ones that provided the best resistance against linear and differential attacks, i.e., that had minimal differential uniformity and linearity. We found 15 different S-boxes that satisfied the above criteria with minimal differential uniformity and linearity, and chose S to be the one given by the first (a,b,c,d,e,f,g,h) in lexicographic order. In particular, the differential uniformity and linearity of S are 4 and 12, respectively.

No Last-Round Cancellation. Considering the model $\tilde{E}_{T_1,T_2} = E_{T_2}^{-1} \circ E_{T_1}$, the last round of E_{T_1} and E_{T_2} can be cancelled out when the last 30-bit subkeys rk_8 are the same, resulting in effectively reducing $\tilde{E}_{T_1,T_2}$ by two rounds. The existence of tweak pairs that make the subkeys collide is unfortunately unavoidable, since the size of the tweak is 48 bits. However, we can prove that the full cancellation can only happen in this case.

Proposition 1. *For any $k, k' \in \mathbb{F}_2^{30}$ and $i = 1,2$, then $R_i(\cdot,k) \neq R_i(\cdot,k')$ if $k \neq k'$.*

Proposition 1 guarantees that R_i (and R_2 in particular) always generate different maps for different 30-bit subkeys so that we can rule out the possibility of the full last-round cancellation. The proof of this result is given in Appendix A.3.

Note that Proposition 1 does not exclude the possibility of a partial last-round cancellation, i.e., the existence of many fixed points. However, we expect that this does not cause critical vulnerability because our tweakey schedule is nonlinear, and generated subkeys have good Hamming distance with different tweaks.

No Last-Two-Round Cancellation. While we cannot avoid that the last round of E_T is canceled, we can show that it is impossible to cancel more rounds. In particular, we now consider the possibility of the last two rounds of E_{T_1} and E_{T_2} being the same map, effectively reducing $\tilde{E}_{T_1,T_2}$ by four rounds. As we have seen, if the collisions are only in the subkeys rk_8 or only in the subkeys rk_7, this is not a problem thanks to Proposition 1. However, partial collisions in subkeys rk_7 and rk_8 cannot result in full round cancellation. In fact, an analogous of the above result for R_i can be proved for $R_2 \circ R_1$ (see Proposition 2), so that the cancellation of the last two rounds of E_{T_1} and E_{T_2} can only occur when both rk_7 and rk_8 collide fully, which cannot happen for different tweaks, thanks to the fact that the tweakey schedule is a permutation on the set of tweaks.

5.3 Design of the Tweakey Schedule

The tweakey schedule generates subkeys from a tweak and the secret key which are used by the data encryption. We carefully designed the tweakey schedule such that it does not affect the critical path of the block cipher to meet the low-latency requirement.

For the design of the tweakey schedule, there are two possibilities: linear or nonlinear. The low-latency block cipher PRINCE [11,12] and the lightweight tweakable block cipher Skinny [3] use linear tweakey / key schedules. On the other hand, AES uses a nonlinear key schedule. With linear tweakey schedules, attackers can generate tweak pairs such that they yield a given subkey difference at no cost. In particular, the attacker can immediately construct two tweaks such that rk_8 are identical. In other words, they can cause the last-round cancellation with no real effort. To avoid such a potential risk, SCARF uses a nonlinear tweakey schedule.

We design the tweakey schedule using a block-cipher-design paradigm, i.e., the tweak is linearly / nonlinearly updated while XORing the secret key. The tweakey schedule first expands the 48-bit tweak to a 60-bit value, i.e., double the size of the subkey of each round.

The nonlinear layer SL is given by twelve parallel applications of the S-box S. These outputs are diffused by the linear layer Σ, which is represented by the sum of 7 bits. Including the key XOR, it consists of the sum of 8 bits, and the critical path is an XOR tree of depth three. We impose the following security criteria on security to pick a good linear layer:

- bijectivity;
- word-wise (5 bits) branch number of 8;
- word-wise (5 bits) branch number of 9 when the Hamming weight of the input is more than 1.

These criteria imply at least $8+9=17$ active S-boxes when the tweak is active. In other words, the Hamming distance of (30×6)-bit subkeys generated by different tweaks is at least 17. Moreover, we can guarantee low differential and linear characteristic probabilities when the tweak is active. In particular, the maximum differential characteristic probability in the tweakey schedule is at most $2^{-3\times 17}=2^{-51}$, while the maximum squared linear trail correlation in the tweakey schedule is at most $2^{-2.83\times 17}=2^{-48.11}$. Both probabilities are lower than 2^{-48}.

We chose the bit permutation π such that each output bit of a single S-box becomes an input bit of a different S-box.

6 Security

In this section, we discuss the cryptanalyis of $\tilde{E}_{T_1,T_2}$ against some major attacks such as the differential [8], linear [31], impossible differential [7], integral [14,19], and meet-in-the-middle attacks [15]. As we will see, no key-recovery or distinguishing attack works for $\tilde{E}_{T_1,T_2}$. This in particular implies that the Security Requirement 1 is achieved.

In the following, we will indicate the round-reduced version of E_T to r rounds as r-round SCARF, while the round reduced version of $\tilde{E}_{T_1,T_2}=E_{T_2}^{-1}\circ E_{T_1}$ to r rounds of E_{T_1} and r rounds of $E_{T_2}^{-1}$ as $(r+r)$-round SCARF.

More details on the cryptanalys conducted to assess the secruity of SCARF can be found in Appendix B.

6.1 Statistical Attacks

Statistical attacks are among the most popular cryptanalysis techniques, based on observable statistical properties that should not be exhibited by randomly chosen functions, therefore making the attacked primitive *distinguishable* from random.

In order to show the security margins of SCARF against the most prominent families of statistical attacks, we can take advantage of the small block size that allows to carry out experiments that are normally not possible for the more common block sizes, like the possibility of computing the Differential Distribution Table (DDT) or the Linear Approximation Table (LAT) of the entire cipher for fixed tweak and key. For most of these families (differential, linear, boomerang and differential-linear), we have conducted experiments that study the distribution of the relevant statistical property (like the linearity or differential uniformity) for $T \mapsto E_T$ and $(T_1, T_2) \mapsto \tilde{E}_{T_1,T_2}$, by computing it experimentally for 2^{10} different tweaks and comparing it to the distribution obtained by drawing 2^{10} random permutations.

In this way, we can avoid any of the typical independency assumptions made for estimating the probability of such distinguishers with larger block sizes for a fixed tweakey. Besides, we can also observe what happens when considering weak tweaks and how frequently we can expect them. For example, we will see that whenever the last 5 bits of the subkey of E_{T_1} and E_{T_2} are the same, the composition of the respective last rounds becomes an affine function (see Section A for details). It allows for the existence (every 2^5 pairs of tweaks) of longer trails than what is expected assuming independence.

On the other hand, the rather limited amount of tweaks used for our experiments cannot fully capture the dependency between T_1 and T_2. For instance, the fact that every 2^5 pairs of tweaks we expect a collision in the last 5-bit subkey, and thus the existence of a differential distinguisher for $(3+3)$-round $\tilde{E}$, implies the existence of a differential distinguisher for $4+4$ rounds every 2^{35} tweaks. In fact, this is to be expected whenever the last 30-bit round key collides (canceling two rounds of $\tilde{E}$), as well as the last 5-bit subkey of the last but one round, making the composition of the last two rounds (that is, four rounds of $\tilde{E}$) an affine function. Even though such rare (and unavoidable) collisions cannot clearly be observable when considering 2^{10} tweaks, we expect that the existence of such rare dependencies does not pose a threat for the security of SCARF when the data is limited to 2^{40}, given the ample margin of security provided by the chosen number of rounds.

A more detailed analysis of the resistance of SCARF against linear and differential attacks (as well as boomerang, differential-linear and impossible differential attacks) and how SCARF compares to ideal permutations is discussed in Appendix B. All in all, we argue that it is not possible to distinguish $5+5$ rounds of SCARF with any of the aforementioned cryptanalysis, given the data limitation.

Related-Tweak Attacks. When considering related-tweak attacks, we can no longer assess the security experimentally because the domain size becomes $48+10=58$ bits. We therefore assume the usual independence assumption, under which we can estimate a low enough probability. As discussed in Sect. Section 5.3, the tweakey schedule guarantees at least 17 active S-boxes in the related-tweak differential/linear attacks. The probability is less than 2^{-48}. Therefore, we expect that related-tweak differential/linear attack does not threaten SCARF.

Multiple-Tweak Attacks. We now focus on the multiple-tweak attack to SCARF. It exploits the fact that the block length of SCARF is smaller than the tweak length and significantly smaller than the security level, so that a significantly smaller bias than what is commonly considered in cryptanalysis can be observed over multiple tweaks. Those attacks have been introduced in [16] with a focus on format preserving encryption, and we adopt here to the case of SCARF.

As a concrete example, the classical differential attack exploits a certain differential property that can be observed with a probability of $p \gg 2^{-n}$, where n is the block length. The multiple-tweak variant exploits the fact that a certain differential property with a probability $2^{-n} + \varepsilon$ can be observed even though $\varepsilon \ll 2^{-n}$. In fact, since the tweak length is relatively large compared to the block length, it is possible to collect an amount of pairs satisfying a certain differential property that is also many times larger than the block length, allowing to observe biases that are extremely small with respect to n. Furthermore, the technique explained in Appendix A.1 can also be used to collect significantly more data than the amount that is queried.

We estimate that the multiple-tweak differential over (5+5)- and (6+6)-round SCARF has a bias of about 2^{-30} and 2^{-40}, respectively. The data limitation does not allow to collect more than 2^{51} and 2^{67} pairs satisfying a certain differential in Security Requirements 1 and 2, respectively (see Appendix A.1). Therefore, even for the bold security claim (Security Requirement 2), distinguishing the (6+6)-round SCARF with a significantly higher advantage is non-trivial. Besides, even if (6+6) rounds were distinguishable, a 60-bit subkey guess is required to attack the full (8+8) rounds. Therefore, we believe SCARF provide a sufficient margin for 80-bit security, particularly for Security Requirement 1. We refer to Appendix B.5 for a more detailed analysis and experiments.

6.2 Impossible Differential Attack

SCARF has the full diffusion in 3 rounds for any subkey. It implies that miss-in-the-middle approach finds at most $3+3=6$-round impossible differential. In our attack model, $\tilde{E}$ consists of $(8+8)$ rounds. Thus, there is plenty of security margin against the impossible differential.

6.3 Integral Attack

The integral attack (also known as higher-order differential attack) exploits the low degree of a cipher. Given an encryption network, we use the division property [48] to detect such distinguishers. We evaluated all integral distinguishers using 2^9 chosen plaintexts. As a result, we found 3-round integral distinguishers, where the left branch is balanced in any 9th order differential. On the other hand, we do not find any 4-round integral distinguisher.

We also considered an extension to the related-tweak setting, where we focus on the sum of many ciphertexts with multiple tweaks. The highest cost distinguisher is constructed by 2^9 chosen plaintexts and 2^{48} tweaks. Again, we used the division property and found 4-round related-tweak integral distinguishers. However, even this extension cannot detect any 5-round distinguishers because the tweakey schedule is a nonlinear function with a high degree.

6.4 Meet-in-the-Middle Attacks

In a Meet-in-the-Middle (MitM) attack, an attacker guesses subkeys for fixed T_1 and T_2 and checks the following equation

$$E_{T_1}(P) = E_{T_2}(C).$$

The attacker can evaluate E_{T_1} and E_{T_2} independently. When κ_1 and κ_2 bits are involved to check this equation for E_{T_1} and E_{T_2}, respectively, the attack requires $N \times (2^{\kappa_1} + 2^{\kappa_2})$, where N is the number of required plaintexts / ciphertexts.

SCARF uses subkeys that involve independently-generated secret-key bits. Therefore, each bit of the subkey is independent. The size of the involved key material is $8 \times 30 = 240$ bits, and it is unlikely that such a straightforward MitM attack works. When we use the 1-bit matching instead of the 10-bit matching, we can bypass guessing subkey bits in the last few rounds. For example, when we focus on the MSB, it is enough to guess only $rk_{7,1}$, $rk_{7,2}$, $rk_{7,3}$, $rk_{7,4}$, $rk_{7,5}$, and $rk_{8,6}$ in the last two rounds. The size of involved subkey bits is reduced from 60 bits to 30 bits. Even if we assume the attacker can have the last-round cancellation at no additional cost, the attack would still involve $30 \times 5 + 30 = 180$-bit subkey. Therefore, we expect that the MitM attack does not invalidate the 80-bit security.

There are several kinds of variants of the MitM attack. In a multi-dimensional meet-in-the-middle (MD-MitM) attack [57], an attacker guesses the intermediate state and applies the MitM with multiple dimensions. In a 3-subset meet-in-the-middle [10], an attacker guesses subkey bits in one subset, and apply the MitM by guessing each remaining subset independently. These variants need to exploit the key schedule. In particular, very simple key schedules are required to successfully apply the attack. The tweakey schedule of SCARF is however nonlinear, and each subkey bit involves many secret key bits. Moreover, SCARF uses a 240-bit random secret key for the 80-bit security. Therefore, we expect that such variants cannot be successfully applied to SCARF.

7 Evaluation

In this section, we evaluate the efficiency of SCARF in hardware and analyze the effects on the system performance when SCARF is used to randomize the cache indexing.

7.1 Hardware Efficiency

We first evaluate and validate the hardware performance of SCARF through a logic synthesis. We implemented SCARF hardware in a fully-unrolled manner, meaning that our implementation including all round functions and key scheduling datapaths is a solely combinational circuit with registers only to store the plaintext P, the initial

Table 1. Synthesis results using Nangate OCLs

Technology	45 nm		15 nm	
	Latency [ns]	Area [GE]	Latency [ps]	Area [GE]
PRINCE	4.74	12,554	628.49	17,484
Mantis	4.73	13,129	630.07	17,641
QARMA	5.11	13,915	654.62	21,102
SCARF	**2.26**	**7,335**	**305.76**	**8,118**

key K, the tweak T, and the encryption result (i.e., ciphertext) C. Note that these registers are used for defining the timing constraints at logic synthesis and evaluating the critical path delay / maximum operational frequency. For the logic synthesis, we employed Synoposys Design Compiler Q-2019.12SP-1 and Nangate 45 nm and 15 nm Open Cell Libraries (OCLs). We synthesized the circuits using `compile -boundary_optimization -map_effort high` without the hierarchy broken (which is suitable to unrolled implementations), and did not apply incremental syntheses.

Table 1 reports the synthesis results of SCARF, where "Latency" denotes the critical path delay which corresponds to a latency of one block encryption and "Area" denotes the circuit area in gate equivalents (GE). Note that Latency includes that for a D-FF (i.e., register) to store plaintext / key / tweak / ciphertext and related control logic. We utilized an area optimization option and a speed optimization (i.e., a frequency constraint such that the latency is minimized as much as possible) for the synthesis. For a comparison, the table also reports the results of PRINCE, Mantis, and QARMA implemented and synthesized in the same manner, as PRINCE is the pioneering and most major conventional low-latency block cipher and Mantis and QARMA are state-of-the-art low-latency tweakable block ciphers. We here focused on the 64-bit 12-round version of Mantis and QARMA (i.e., $\texttt{MANTIS}_6$-64 and $\texttt{QARMA}_6$-64-σ_0) as recommended for the security against practical attacks in [1]. Note that, for the synthesis of each cipher, the speed optimization was set such that its latency is minimized. The synthesis results confirm that the latency of SCARF is less than half of that of PRINCE, Mantis, and QARMA, which reveals the advantage of SCARF for the low-latency application including the cache-randomization. Moreover, the SCARF critical path lies on the round datapath in addition to the first key-tweak XOR, whereas the difference in latency between round and key schedule datapaths is almost equivalent to each other. This might indicate that SCARF would be reasonable as a low-latency tweakable block cipher for the block, key, and tweak lengths.

7.2 Performance Benchmark

To evaluate the performance of SCARF, we implement a randomized cache using PRINCE [12] and SCARF in the gem5 simulator [29]. We simulate a two level cache hierarchy with a 16 kB L1 data cache, 16 kB L1 instruction cache and 1 MB unified L2 cache. The L1 caches have 8 ways, and the L2 cache has 16 ways. The system is clocked at 2 GHz (500 ps period) equipped with 2 GB memory with 100 ns ($\pm 10\ ns$) latency. The default (non-randomized) caches have a tag-latency of 1 clock cycle for the

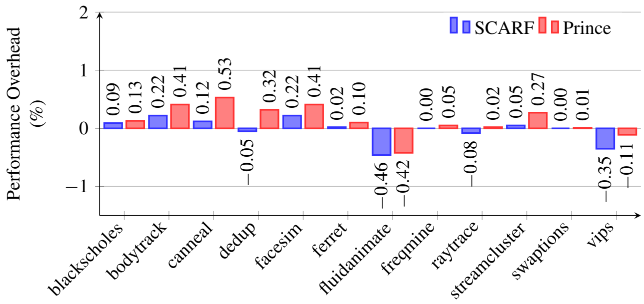


Fig. 5. Performance improvement of SCARF compared to PRINCE in percent using the PARSEC benchmark suite.

L1 caches and 10 clock cycles for the L2 cache. In Section 7.1 we found that SCARF has a latency of about 306 ps and PRINCE has a latency of 628 ps in 15 nm technology. Hence, for the simulation we assume that SCARF adds one clock cycle delay to the L2 cache access and PRINCE adds two cycles.

Figure 5 shows the performance results of the PARSEC benchmark suite using SCARF and PRINCE for cache randomization in comparison to a traditional, non-randomized cache. We averaged the benchmarks over 10 executions with random keys to account for differences in scheduling decisions and noise from parallel processes. The first observation is that despite the added delay for the randomized caches, some benchmarks are faster that in the non-randomized setup. This is consistent with prior work [55] and an artifact of frequent evictions within the data used by the benchmarks. For example, if a given benchmark uses $w+1$ addresses that map to the same cache index frequently, many cache misses occur in a w-way cache which slows down the computation. In the randomized setting, the probability that all those addresses map to the exact same entries is very small. Hence, it is more likely that the addresses can be co-located in the cache without causing frequent evictions.

Despite the speedup that is achieved for some of the benchmarks, the results indicate in general that the cache is a very timing-sensitive part of the CPU. Even as little as two added clock cycles in the latency result in up to 0.53% reduced overall performance. On average, our simulation using PRINCE results in a 0.11% overhead while SCARF causes on average a 0.003% *performance increase*. In combination with the reduced area requirements of SCARF, this is an important improvement on the way to randomized cache architectures in real-world CPUs.

8 Conclusion

In this work we presented SCARF, the first purpose-built cache randomization function. Our design uses a 10-bit tweakable block-cipher that allows for an elegant attacker model, contributing to the crucial low-latency property. We implement SCARF in hardware and compare it to other low-latency block ciphers. Our design outperforms other block ciphers by a factor of two, both in area and performance. We implemented

SCARF in gem5 to evaluate the effect on system level performance. Using SCARF, the overhead of cache randomization can be compensated entirely, matching the performance of traditional caches.

Finally, our findings and design approach can be used as groundwork for specific cache randomization designs with other parameter sizes.

Acknowledgments

The authors would like to thank Eran Lambooij and Shahram Rasoolzadeh for the fruitful discussions.

References

1. Roberto Avanzi. The QARMA Block Cipher Family – Almost MDS Matrices Over Rings With Zero Divisors, Nearly Symmetric Even-Mansour Constructions With Non-Involutory Central Rounds, and Search Heuristics for Low-Latency S-Boxes. *IACR Trans. Symmetric Cryptol.*, 2017(1):4–44, 2017.
2. Achiya Bar-On, Orr Dunkelman, Nathan Keller, and Ariel Weizman. DLCT: A new tool for differential-linear cryptanalysis. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 313–342. Springer, 2019.
3. Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. 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 123–153. Springer, 2016.
4. Mihir Bellare, Thomas Ristenpart, Phillip Rogaway, and Till Stegers. Format-Preserving Encryption. In Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *Selected Areas in Cryptography, 16th Annual International Workshop, SAC 2009, Calgary, Alberta, Canada, August 13-14, 2009, Revised Selected Papers*, volume 5867 of *Lecture Notes in Computer Science*, pages 295–312. Springer, 2009.
5. Mihir Bellare, Phillip Rogaway, and Terence Spies. The FFX mode of operation for format-preserving encryption. *NIST Draft*, 20:19, 2010.
6. Daniel J Bernstein. Cache-timing attacks on AES. 2005.
7. Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 12–23. Springer, 1999.
8. Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In Alfred Menezes and Scott A. Vanstone, editors, *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1990.
9. Rahul Bodduna, Vinod Ganesan, Patanjali SLPSK, Kamakoti Veezhinathan, and Chester Rebeiro. Brutus: Refuting the Security Claims of the Cache Timing Randomization Countermeasure Proposed in CEASER. *IEEE Comput. Archit. Lett.*, 19(1):9–12, 2020.

10. Andrey Bogdanov and Christian Rechberger. A 3-subset meet-in-the-middle attack: Cryptanalysis of the lightweight block cipher KTANTAN. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers*, volume 6544 of *Lecture Notes in Computer Science*, pages 229–240. Springer, 2010.
11. 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.
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 (Full version). *IACR Cryptol. ePrint Arch.*, page 529, 2012.
13. Carlos Cid, Tao Huang, Thomas Peyrin, Yu Sasaki, and Ling Song. Boomerang Connectivity Table: A New Cryptanalysis Tool. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 683–714. Springer, 2018.
14. Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The Block Cipher Square. In Eli Biham, editor, *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings*, volume 1267 of *Lecture Notes in Computer Science*, pages 149–165. Springer, 1997.
15. Whitfield Diffie and Martin E. Hellman. Special Feature Exhaustive Cryptanalysis of the NBS Data Encryption Standard. *Computer*, 10(6):74–84, 1977.
16. Orr Dunkelman, Abhishek Kumar, Eran Lambooij, and Somitra Kumar Sanadhya. Cryptanalysis of feistel-based format-preserving encryption. *IACR Cryptol. ePrint Arch.*, page 1311, 2020.
17. Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. Flush+Flush: A Fast and Stealthy Cache Attack. In Juan Caballero, Urko Zurutuza, and Ricardo J. Rodríguez, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment - 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings*, volume 9721 of *Lecture Notes in Computer Science*, pages 279–299. Springer, 2016.
18. Jérémy Jean, Ivica Nikolic, Thomas Peyrin, and Yannick Seurin. The Deoxys AEAD Family. *J. Cryptol.*, 34(3):31, 2021.
19. Lars R. Knudsen and David A. Wagner. Integral Cryptanalysis. In Joan Daemen and Vincent Rijmen, editors, *Fast Software Encryption, 9th International Workshop, FSE 2002, Leuven, Belgium, February 4-6, 2002, Revised Papers*, volume 2365 of *Lecture Notes in Computer Science*, pages 112–127. Springer, 2002.
20. Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre Attacks: Exploiting Speculative Execution. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 1–19. IEEE, 2019.

21. Michael E. Kounavis, Sergej Deutsch, Santosh Ghosh, and David Durham. K-Cipher: A Low Latency, Bit Length Parameterizable Cipher. In *IEEE Symposium on Computers and Communications, ISCC 2020, Rennes, France, July 7-10, 2020*, pages 1–7. IEEE, 2020.
22. Hugo Krawczyk. LFSR-based Hashing and Authentication. In Yvo Desmedt, editor, *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, volume 839 of *Lecture Notes in Computer Science*, pages 129–139. Springer, 1994.
23. Susan K. Langford and Martin E. Hellman. Differential-Linear Cryptanalysis. In Yvo Desmedt, editor, *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, volume 839 of *Lecture Notes in Computer Science*, pages 17–25. Springer, 1994.
24. 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.
25. Michael LeMay, Joydeep Rakshit, Sergej Deutsch, David M. Durham, Santosh Ghosh, Anant Nori, Jayesh Gaur, Andrew Weiler, Salmin Sultana, Karanvir Grewal, and Sreenivas Subramoney. Cryptographic Capability Computing. In *MICRO '21: 54th Annual IEEE/ACM International Symposium on Microarchitecture, Virtual Event, Greece, October 18-22, 2021*, pages 253–267. ACM, 2021.
26. Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading Kernel Memory from User Space. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 973–990. USENIX Association, 2018.
27. Moses D. Liskov, Ronald L. Rivest, and David A. Wagner. Tweakable Block Ciphers. In Moti Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2002.
28. Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. Last-Level Cache Side-Channel Attacks are Practical. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 605–622. IEEE Computer Society, 2015.
29. Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, Mohammad Alian, Rico Amslinger, Matteo Andreozzi, Adrià Armejach, Nils Asmussen, Srikant Bharadwaj, Gabe Black, Gedare Bloom, Bobby R. Bruce, Daniel Rodrigues Carvalho, Jerónimo Castrillón, Lizhong Chen, Nicolas Derumigny, Stephan Diestelhorst, Wendy Elsasser, Marjan Fariborz, Amin Farmahini Farahani, Pouya Fotouhi, Ryan Gambord, Jayneel Gandhi, Dibakar Gope, Thomas Grass, Bagus Hanindhito, Andreas Hansson, Swapnil Haria, Austin Harris, Timothy Hayes, Adrian Herrera, Matthew Horsnell, Syed Ali Raza Jafri, Radhika Jagtap, Hanhwi Jang, Reiley Jeyapaul, Timothy M. Jones, Matthias Jung, Subash Kannoth, Hamidreza Khaleghzadeh, Yuetsu Kodama, Tushar Krishna, Tommaso Marinelli, Christian Menard, Andrea Mondelli, Tiago Mück, Omar Naji, Krishnendra Nathella, Hoa Nguyen, Nikos Nikoleris, Lena E. Olson, Marc S. Orr, Binh Pham, Pablo Prieto, Trivikram Reddy, Alec Roelke, Mahyar Samani, Andreas Sandberg, Javier Setoain, Boris Shingarov, Matthew D. Sinclair, Tuan Ta, Rahul Thakur, Giacomo Travaglini, Michael Upton, Nilay Vaish, Ilias Vougioukas, Zhengrong Wang, Norbert Wehn, Christian Weis, David A. Wood, Hongil Yoon, and Éder F. Zulian. The gem5 simulator: Version 20.0+. *CoRR*, abs/2007.03152, 2020.
30. Mohammad Mahzoun, Liliya Kraleva, Raluca Posteuca, and Tomer Ashur. Differential Cryptanalysis of K-Cipher. *IACR Cryptol. ePrint Arch.*, page 1158, 2022.

31. Mitsuru Matsui. Linear Cryptanalysis Method for DES Cipher. In Tor Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer, 1993.
32. Mitsuru Matsui. New Block Encryption Algorithm MISTY. In Eli Biham, editor, *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings*, volume 1267 of *Lecture Notes in Computer Science*, pages 54–68. Springer, 1997.
33. Kit Murdock, David Oswald, Flavio D. Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. Plundervolt: Software-based Fault Injection Attacks against Intel SGX. In *Proceedings of the 41st IEEE Symposium on Security and Privacy (S&P'20)*, 2020.
34. Maria Mushtaq, Muhammad Asim Mukhtar, Vianney Lapotre, Muhammad Khurram Bhatti, and Guy Gogniat. Winter is here! A decade of cache-based side-channel attacks, detection & mitigation for RSA. *Information Systems*, 92:101524, 2020.
35. Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and Countermeasures: The Case of AES. *IACR Cryptol. ePrint Arch.*, page 271, 2005.
36. Dan Page. Partitioned Cache Architecture as a Side-Channel Defence Mechanism. *IACR Cryptol. ePrint Arch.*, 2005:280, 2005.
37. Antoon Purnal, Lukas Giner, Daniel Gruss, and Ingrid Verbauwhede. Systematic Analysis of Randomization-based Protected Cache Architectures. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 987–1002. IEEE, 2021.
38. Moinuddin K. Qureshi. CEASER: Mitigating Conflict-Based Cache Attacks via Encrypted-Address and Remapping. In *51st Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2018, Fukuoka, Japan, October 20-24, 2018*, pages 775–787. IEEE Computer Society, 2018.
39. Moinuddin K. Qureshi. New attacks and defense for encrypted-address cache. In Srilatha Bobbie Manne, Hillery C. Hunter, and Erik R. Altman, editors, *Proceedings of the 46th International Symposium on Computer Architecture, ISCA 2019, Phoenix, AZ, USA, June 22-26, 2019*, pages 360–371. ACM, 2019.
40. Jordi Ribes-González, Oriol Farràs, Carles Hernández, Vatistas Kostalabros, and Miquel Moretó. A Security Model for Randomization-based Protected Caches. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(3):1–25, 2022.
41. Gururaj Saileshwar and Moinuddin K. Qureshi. MIRAGE: Mitigating Conflict-Based Cache Attacks with a Practical Fully-Associative Design. In Michael Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 1379–1396. USENIX Association, 2021.
42. Rich Schroeppel. Hasty pudding cipher specification. In *First AES Candidate Workshop*, 1998.
43. Wei Song, Boya Li, Zihan Xue, Zhenzhen Li, Wenhao Wang, and Peng Liu. Randomized Last-Level Caches Are Still Vulnerable to Cache Side-Channel Attacks! But We Can Fix It. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 955–969. IEEE, 2021.
44. Wei Song and Peng Liu. Dynamically Finding Minimal Eviction Sets Can Be Quicker Than You Think for Side-Channel Attacks against the LLC. In *22nd International Symposium on Research in Attacks, Intrusions and Defenses, RAID 2019, Chaoyang District, Beijing, China, September 23-25, 2019*, pages 427–442. USENIX Association, 2019.
45. Qinhan Tan, Zhihua Zeng, Kai Bu, and Kui Ren. PhantomCache: Obfuscating Cache Conflicts with Localized Randomization. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society, 2021.

46. Jan Philipp Thoma and Tim Güneysu. Write Me and I'll Tell You Secrets - Write-After-Write Effects On Intel CPUs. In *RAID '22: 25th International Symposium on Research in Attacks, Intrusions and Defenses, Limassol, Cyprus, October 26-28, 2022*. ACM, 2022.
47. Jan Philipp Thoma, Christian Niesler, Dominic A. Funke, Gregor Leander, Pierre Mayr, Nils Pohl, Lucas Davi, and Tim Güneysu. ClepsydraCache - Preventing Cache Attacks with Time-Based Evictions. *CoRR*, abs/2104.11469, 2021.
48. Yosuke Todo. Structural Evaluation by Generalized Integral Property. 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 287–314. Springer, 2015.
49. Yosuke Todo. Integral Cryptanalysis on Full MISTY1. *J. Cryptol.*, 30(3):920–959, 2017.
50. Eran Tromer, Dag Arne Osvik, and Adi Shamir. Efficient Cache Attacks on AES, and Countermeasures. *J. Cryptol.*, 23(1):37–71, 2010.
51. Pepe Vila, Boris Köpf, and José F. Morales. Theory and Practice of Finding Eviction Sets. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 39–54. IEEE, 2019.
52. David A. Wagner. The Boomerang Attack. In Lars R. Knudsen, editor, *Fast Software Encryption, 6th International Workshop, FSE '99, Rome, Italy, March 24-26, 1999, Proceedings*, volume 1636 of *Lecture Notes in Computer Science*, pages 156–170. Springer, 1999.
53. Yao Wang, Andrew Ferraiuolo, Danfeng Zhang, Andrew C. Myers, and G. Edward Suh. SecDCP: Secure dynamic cache partitioning for efficient timing channel protection. In *Proceedings of the 53rd Annual Design Automation Conference, DAC 2016, Austin, TX, USA, June 5-9, 2016*, pages 74:1–74:6. ACM, 2016.
54. Zhenghong Wang and Ruby B. Lee. New cache designs for thwarting software cache-based side channel attacks. In Dean M. Tullsen and Brad Calder, editors, *34th International Symposium on Computer Architecture (ISCA 2007), June 9-13, 2007, San Diego, California, USA*, pages 494–505. ACM, 2007.
55. Mario Werner, Thomas Unterluggauer, Lukas Giner, Michael Schwarz, Daniel Gruss, and Stefan Mangard. ScatterCache: Thwarting Cache Attacks via Cache Set Randomization. In Nadia Heninger and Patrick Traynor, editors, *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, pages 675–692. USENIX Association, 2019.
56. Yuval Yarom and Katrina Falkner. FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In Kevin Fu and Jaeyeon Jung, editors, *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*, pages 719–732. USENIX Association, 2014.
57. Bo Zhu and Guang Gong. Multidimensional meet-in-the-middle attack and its applications to KATAN32/48/64. *Cryptogr. Commun.*, 6(4):313–333, 2014.

A Some Properties of SCARF

In this section, we are going to discuss some notable properties that arise when considering the security of $\tilde{E}$. For better readability, we are going to add a prime symbol for the later part of $\tilde{E}_{T,T'} = E_{T'}^{-1} \circ E_T$, e.g., rk'_8 denotes the subkey of the 8th round function of $E_{T'}$.

A.1 Learning Full Queries with Birthday Queries

A unique property that arises from the attacker model (and is not due to the design of SCARF, as discussed in Section 3) is structural queries that can collect N^2 plaintext-ciphertext pairs by only N queries to SCARF. In fact, when a fixed tweak T_1 and a plaintext P_1 are chosen, we can query P_1 to $\tilde{E}_{T_1,T_i}$ with chosen tweak T_i and get $P_i = \tilde{E}_{T_1,T_i}(P_1)$. Then, N queries allows us to lean about N^2 queries, i.e., $P_j = \tilde{E}_{T_i,T_j}(P_i) = \tilde{E}_{T_1,T_j} \circ \tilde{E}_{T_i,T_1}(P_i)$ for any $i \in \{1,2,\ldots,N\}$ and $j \in \{1,2,\ldots,N\}$.

Note that we cannot always use these structural queries in arbitrary cases. For example, an attacker can choose P_1 and T_i but cannot choose P_i for $i \geq 2$. Therefore, when we learn $P_j = \tilde{E}_{T_i,T_j}(P_i)$, the attacker can make use of these additional queries in a known-plaintext attack.

More importantly, this also has implications for chosen-ciphertext attacks by querying the full code book every tweak. In fact, suppose the adversary queries the full code-book for $2 \cdot N_T$ different tweaks divided in two disjoint sets $\mathcal{T}$ and $\mathcal{T}'$. The attacker then queries $\tilde{E}_{T_1,T'}$ for some fixed $T_1 \in \mathcal{T}$ and for all $T' \in \mathcal{T}'$. Similarly, they query $\tilde{E}_{T,T'_1}$ for all $T \in \mathcal{T}$ and some fixed $T'_1 \in \mathcal{T}'$. This requires $2 \cdot 2^{10} \cdot N_T - 2^{10}$ queries. However, the attacker can learn the full codebook for all $T \in \mathcal{T}, T' \in \mathcal{T}'$ since $\tilde{E}_{T,T'} = \tilde{E}_{T_1,T'} \circ \tilde{E}_{T_1,T'_1} \circ \tilde{E}_{T,T'_1}$. In other words, the attacker can then learn $2^{10} \cdot N_T^2$ pairs with $\cdot 2^{10} \cdot N_T - 2^{10}$ queries.

In Security Requirement 2, an attacker can query $\tilde{E}$ 2^{40} times. It implies that the attacker can learn the full code book for $2^{29+29} = 2^{58}$ tweaks, i.e., 2^{68} data.

On contrary, in Security Requirement 1, an attacker can query O, which returns 0 or 1 only. For fixed T and T', we need to ask about 2^{18} (P,T,P',T') to O to get the full code book. With $N_T = 2^{21}$, the query complexity is close to $2 \times N_T \times 2^{18} = 2^{40}$. It implies that the attacker can learn the full code book for $2^{21} \times 2^{21} = 2^{42}$ tweaks, i.e., 2^{52} data.

A.2 Partial Last-Two-Round Cancellation

One of the most notable properties when considering the security of $\tilde{E}$ is the so-called last-round cancellation shown in Sect.5. Since SCARF has 8 rounds, when $rk_8 = rk'_8$, the last round is cancelled out in the composition. Therefore, the total number of rounds of $\tilde{E}_{T,T'}$ decreases to 7+7 rounds. Due to the property of the tweakey schedule, there is no chance that $rk_7 \| rk_8 = rk'_7 \| rk'_8$ implying that two rounds cannot be cancelled out, as we show in Proposition 2. However, partial collision can still happen.

We first consider tweak pairs with a 35-bit collision, $rk_{7,6} \| rk_8 = rk'_{7,6} \| rk'_8$. Let $(x'_L, x'_R) = (R'^{-1}_1 \circ R'^{-1}_2 \circ R_2 \circ R_1)(x_L, x_R)$ representing the last 2+2 rounds, then in the case of collision this function is actually the following key-dependent affine transformation

$$x'_L = x_L,$$
$$x'_R = x_R \oplus \left[\bigoplus_{i=1}^{4} (\tau_i(x_L) \wedge (rk_{7,i+1} \oplus rk'_{7,i+1})) \right] \oplus (rk_{7,1} \oplus rk'_{7,1}).$$

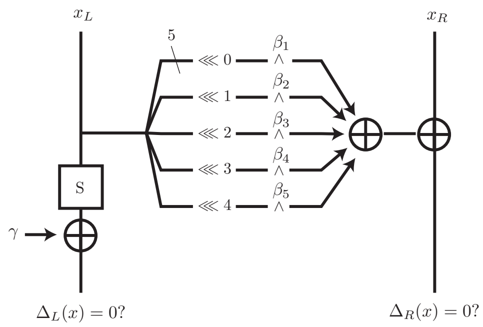


Fig. 6. The function $(\Delta_L, \Delta_R)(x) = R_2(x,k) \oplus R_2(x,k')$.

The 35-bit collision derives the left-branch collision in the input of the 7th round function. Moreover, when the jth bit of $rk_{7,i}$ collides with the jth bit of $rk'_{7,i}$ for all $i \in \{1,2,3,4,5\}$, the jth bit of the right-branch also collides in the input of the 7th round function. In other words, $(35+5c)$-bit subkey collision derives a 5-bit collision in the left branch and a c-bit collision in the last two rounds.

A.3 No Last-Round Cancellation

In this section, we first prove Proposition 1 and, furthermore, that it is not possible to cancel the last two rounds of E_{T_1} and E_{T_2} unless the last two subkeys collide, which implies that $T_1 = T_2$.

Proof (Proof of Proposition 1). For simplicity, we are going to prove the result for R_2, since it is the one of interest for our cipher and the proof for R_1 is analogous.

We want to show that the function $x \mapsto (\Delta_L, \Delta_R)(x) = R_2(x,k) \oplus R_2(x,k')$ is the zero function if and only if $k \oplus k' = 0$. This function is shown in Figure 6, where $k = (k_1, \ldots, k_6)$ and $k' = (k'_1, \ldots, k'_6)$ such that $k \oplus k' = (\beta_1, \ldots, \beta_5, \gamma)$, for some $\beta = (\beta_1, \ldots, \beta_5) \in F_2^{25}$ (difference in the subkeys $k_1, \ldots, k_5$) and $\gamma \in \mathbb{F}_2^5$ (difference in the subkey k_6). Notice that any non key-dependent component of R_2, like $\tau_1(x) \wedge \tau_2(x)$ in the G function, gets cancelled out since the difference is in the key.

With these new notations, we show that the function

$$(\Delta_L, \Delta_R)(x) = R_2(x,k) \oplus R_2(x,(k \oplus (\beta,\gamma))$$

is the zero function, then β and γ are all zero.

Let M_κ be the 5×5 matrix that represents $x \mapsto \bigoplus_{i=0}^{4} (\tau_i(x) \wedge k_{i+1})$, with $\kappa = (k_1, \ldots, k_5) \in \mathbb{F}_2^{25}$. Then, we can write that

$$\Delta_R(x_L, x_R) = M_\beta(x_L).$$

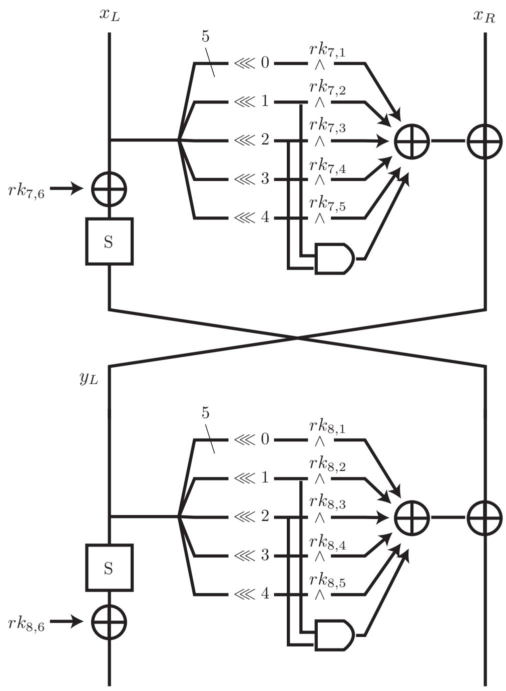


Fig. 7. The last two rounds of E_T.

Then $\Delta_R(x_L, x_R) \neq 0$ for any x_L outside the kernel of M_β; in particular, such an x_L does not exist if and only if M_β is the zero matrix, that is $\beta = 0$. Therefore, $\Delta_R = 0$ if and only if β is the zero difference.

Finally, if $\beta = 0$, let us consider

$$\Delta_L(x_L, x_R) = \gamma.$$

then it is clear that $\Delta_L = 0$ if and only if $\gamma = 0$. Therefore, we have that $\Delta_L = \Delta_R = 0$ if and only if β and γ are the zero difference. □

Proposition 2. *For any $k, k' \in \mathbb{F}_2^{60}$ then $R_2 \circ R_1(\cdot, k) \neq R_2 \circ R_1(\cdot, k')$ if $k \neq k'$.*

Proof. Following the notations of Figure 7 of the last two rounds of SCARF, we are going to show that for any fixed $k \in \mathbb{F}_2^{60}$, $\beta = (\beta_1, \beta_2, \beta_3, \beta_5) \in F_2^{25}$ (difference in $rk_{7,1}, \ldots, rk_{7,5}$), $\delta = (\delta_1, \delta_2, \delta_3, \delta_5) \in F_2^{25}$ (difference in $rk_{8,1}, \ldots, rk_{8,5}$) and $\gamma, \varepsilon \in \mathbb{F}_2^5$ (differences in $rk_{7,6}, rk_{8,6}$ respectively), then the function

$$x \mapsto (\Delta_L, \Delta_R)(x) = R_2 \circ R_1(x, k) \oplus R_2 \circ R_1(x, (k \oplus (\beta, \gamma, \delta, \varepsilon))$$

is the zero function then $\beta, \gamma, \delta, \varepsilon$ are all the zero difference, that is $k = k'$.

Let M_κ, be the 5×5 matrix that represents $x \mapsto \bigoplus_{i=0}^{4}(\tau_i(x) \wedge k_{i+1})$, with $\kappa = (k_1, \ldots, k_5) \in \mathbb{F}_2^{25}$. Then, if we let y_L be the output value of the left branch of R_1 with the key k and y'_L with key k', we have that

$$y'_L = y_L \oplus M_\beta(x_L).$$

Let us consider

$$\Delta_L(x_L, x_R) = S(y_L) \oplus S(y_L \oplus M_\beta(x_L)) \oplus \varepsilon. \tag{1}$$

If $\varepsilon \neq 0$, given that $x_R \mapsto y_L$ is a bijection for any fixed x_L, we can always choose a y_L (and therefore x_R) such that $S(y_L) \oplus S(y_L \oplus M_\beta(x_L)) \neq \varepsilon$, otherwise S would have a non-trivial differential of probability 1.

Therefore, in order to have $\Delta_L(x_L, x_R) = 0$ for all x_L and x_R, we must have that $\varepsilon = 0$. In this case, Equation (1) implies that $\Delta_L(x_L, x_R) = 0$ if and only if $y_L = y'_L$, that is $M_\beta(x_L) = 0$. In other words, $\Delta_L(x_L, x_R) = 0$ for all x_L, x_R if and only if $M_\beta(x_L) = 0$, that is $\beta = 0$.

Let us then assume that β and ε are all the zero difference, so that Δ_L is the zero function. Then we can write

$$\Delta_R(x_L, x_R) = M_\delta(y_L) \oplus S(x_L) \oplus S(x_L \oplus \gamma). \tag{2}$$

If $\gamma \neq 0$, for any fixed x_L, we have that $S(x_L) \oplus S(x_L \oplus \gamma)$ is also non-zero, so that if we choose $y_L = 0$ (by choosing x_R appropriately) we have that

$$\Delta_R(x_L, x_R) = S(x_L) \oplus S(x_L \oplus \gamma) \neq 0$$

for any x_L and δ. Therefore we must have $\gamma = 0$. But if $\gamma = 0$, we would have from Equation (2) that Δ_R is the zero function if and only if M_δ is also the zero function, that is $\delta = 0$.

B Detailed Cryptanalysis Reports of SCARF

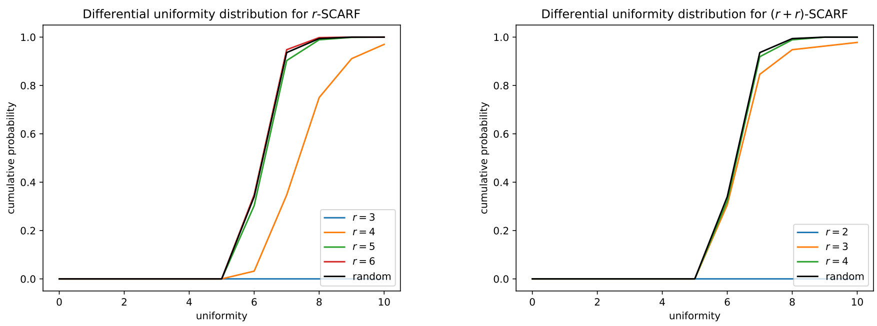
We analyze the statistical behavior of SCARF on well-studied cryptanalysis: differential, linear, boomerang, and differential-linear attacks. We briefly address the security of SCARF against invariant attacks.

B.1 Differential Cryptanalysis

A differential attack [8] exploits a non ideal behaviour of a cipher in the propagation of differences. As a rule of thumb, it is possible to mount a differential attack on E (of block size n) whenever there exist α, β such that

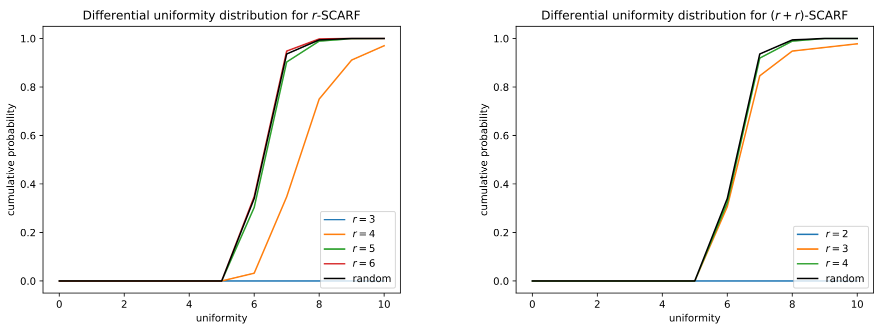
$$\Pr_x(E(x) \oplus E(x \oplus \alpha) = \beta) > 2^{-n}.$$

$\alpha \to \beta$ is then called a differential trail.



(a) r-round SCARF (b) $(r+r)$-round SCARF

Fig. 8. Cumulative probability distribution for the differential uniformity of SCARF.



(a) r-round SCARF (b) $(r+r)$-round SCARF

Fig. 9. Cumulative probability distribution for the differential uniformity of SCARF.

The maximum differential probability of the S-box is $4/32 = 2^{-3}$. Moreover, the maximum differential probability of the G function is $16/32 = 2^{-1}$. Therefore, for any subkey, the maximum differential characteristic probability (MDCP) is 2^{-4} every two rounds. The MDCP of the 6-round cipher is is lower than 2^{-10}. Namely, we do not expect that any differential property is observable neither for the 8 rounds of E nor for the $8+8$ rounds of $\tilde{E}$, with plenty of security margin against differential cryptanalysis.

Figure 9 shows the distribution of the differential uniformity (that is, the maximum probability of any possible differential trail multiplied by the cardinality of the domain, 2^{10}) of E and $\tilde{E}$ over 2^{10} random tweaks. While for $r = 5$ and 6 the distribution of the differential uniformity of E almost perfectly matches the one obtained for random permutations, the same is not true for $\tilde{E}$ and $r = 3$, despite the MDCP being lower than 2^{-10} for fixed tweakeys. As far as we could tell, this is due to the unavoidable collision of the last 5-bit subkeys every 2^5 tweaks. When the last 5-bit subkey of T_1 and T_2 collide, the composition of the last two rounds of E_{T_1} and E_{T_2} become an affine function: as a consequence, for such tweaks there exists differential trails for $(3+3)$-round $\tilde{E}$ with $p < 2^{-10}$, that would not exist otherwise.

Furthermore, we cannot observe any non-ideal behaviour of $\tilde{E}$ for $r = 4$ for this amount of tweaks, although this does happen whenever the last 35-bit subkeys collide, which we expect to happen every 2^{35} pairs of tweaks approximately (for a more detailed discussion on these partial collisions, we refer to Appendix A.2). We expect that $5+5$ rounds cannot be distinguished, even taking into account the observation made in A, since a cancellation of two (or more) last rounds implies a collision of 60-bits subkeys, that is the 48-bits tweaks are actually the same.

A possible extension of differential attacks for tweakable block ciphers are related-tweak differential attacks. However, the tweakey schedule of SCARF has many nonlinear components, and the tweakey schedule itself ensures 17 active S-boxes, from which it follows that the differential probability of a characteristic of the tweakey schedule is at most 2^{-51}. Therefore, we expect that the related-tweak differential attack is not of concern.

B.2 Linear Cryptanalysis

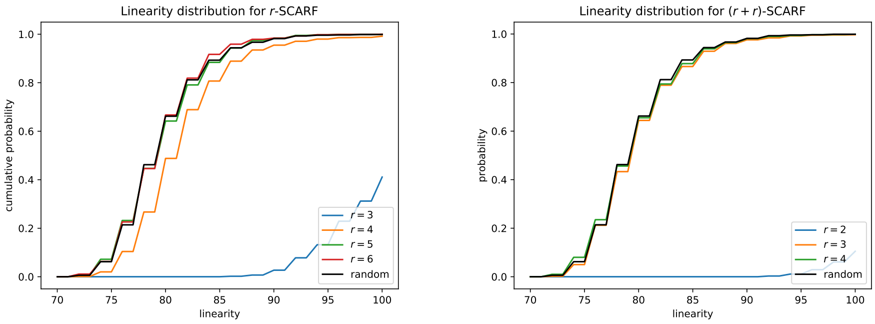
A linear attack on an n-bit block length cipher E is possible whenever there exists masks α, β such that if

$$\Pr_x(\langle \alpha, x\rangle \oplus \langle \beta, E(x)\rangle) = \frac{1}{2} + \frac{c}{2}$$

then the correlation of the linear approximation c must be smaller than $2^{n/2}$. We say that $\alpha \to \beta$ is a linear trail.

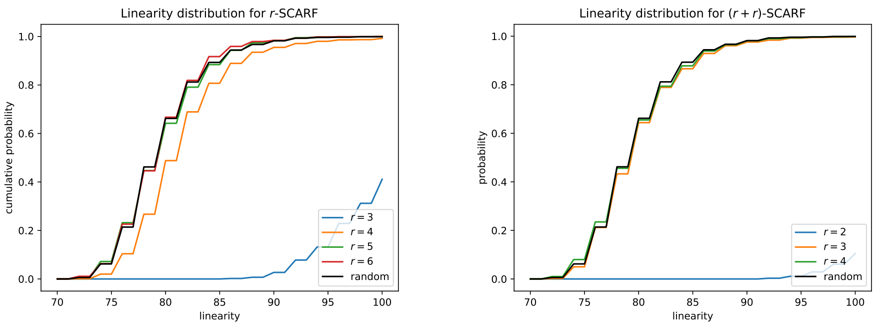
The maximum squared correlation of the 5-bit S-box is $(12/32)^2 = 2^{-2.83}$. Moreover, the maximum squared correlation of the G function is $(16/32)^2 = 2^{-2}$. Since there is at least one active S-box and active G function every two rounds for any subkey, the maximum linear squared trail correlation is $2^{-4.83}$ over two rounds, and of 6-round SCARF is lower than $2^{-14.49}$. Similarly to the differential attack, we conclude that 8-round E does not have any non-trivial linear property, as well as $(8+8)$-round $\tilde{E}$.

Figure 11 shows the distribution of the linearity (that is, the maximum absolute correlation over all possible linear trails, multiplied by 2^{10}) of E and $\tilde{E}$ over 2^{10} tweaks.



(a) r-round SCARF (b) $(r+r)$-round SCARF

Fig. 10. Cumulative probability distribution for the linearity of SCARF.



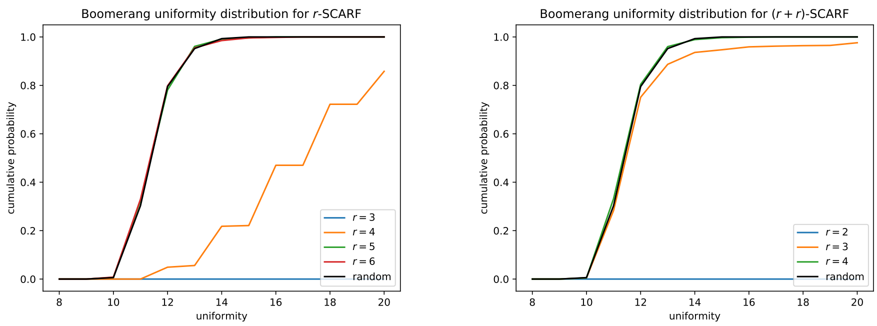
(a) r-round SCARF (b) $(r+r)$-round SCARF

Fig. 11. Cumulative probability distribution for the linearity of SCARF.

We observe that for $r = 5$ and 6 the distribution of the linearity of E, as well as that of $\tilde{E}$ with $r \geq 3$ matches the one drawn for random permutations very closely.

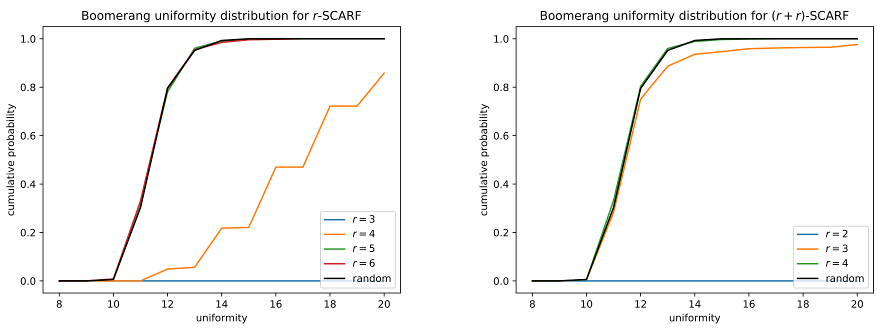
As for the related-tweak scenario, we observe that any linear characteristic has at least 17 active Sboxes so that its linear probability is at most $2^{-48.11}$. Therefore, we expect that the related-tweak linear attack cannot be applied successfully.

B.3 Boomerang Attacks



(a) r-round SCARF (b) $(r+r)$-round SCARF

Fig. 12. Cumulative probability distribution for the boomerang uniformity of SCARF.



(a) r-round SCARF (b) $(r+r)$-round SCARF

Fig. 13. Cumulative probability distribution for the boomerang uniformity of SCARF.

A boomerang distinguisher [52] of E is given by α, β such that

$$\Pr_x(E^{-1}(E(x) \oplus \beta, k) \oplus E^{-1}(E(x \oplus \alpha) \oplus \beta) = \alpha) > 2^{-n}$$

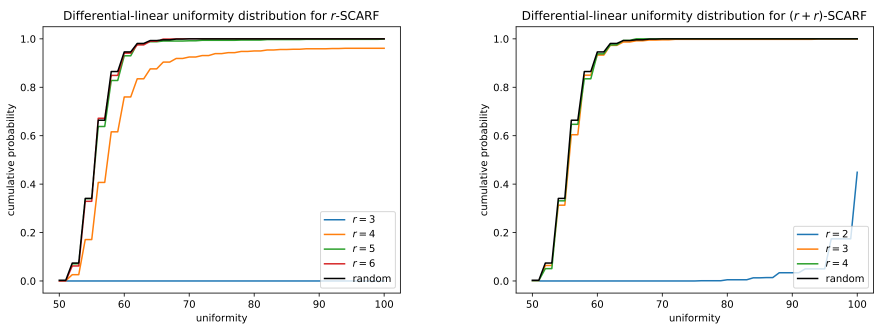
if n is the block size of E.

Usually, boomerang distinguishers are obtained by the composition of two differential trails: for a cipher $E = E_1 \circ E_m \circ E_2$ can be estimated as $p^2 r q^2$, where p and q are the probability of two differential trails for E_1 and E_2, and r is the probability of *connecting* the two trails over E_m [13].

As previously mentioned, the MDP of SCARF is 2^{-4} every two rounds, so that E_1 and E_2 cannot both be chosen to cover more than two rounds. Furthermore, the boomerang uniformity of S is 6 (that is the maximum probability of any possible boomerang distinguisher of S, multiplied by the cardinality of the domain 2^5). Therefore, we expect that there does not exist a distinguisher over 5 rounds of probability higher than 2^{-10}. Once again, the fact that E is 8 rounds and $\tilde{E}$ is $8+8$ rounds guarantees ample margin of security against this class of attacks.

Figure 13 shows the distribution of the boomerang uniformity of E and $\tilde{E}$ over 2^{10} tweaks. Once again, for $r = 5$ and 6 the distribution of the uniformity of E, as well as that of $\tilde{E}$ with $r \geq 3$ matches the one drawn for random permutations very closely.

B.4 Differential-linear Attacks

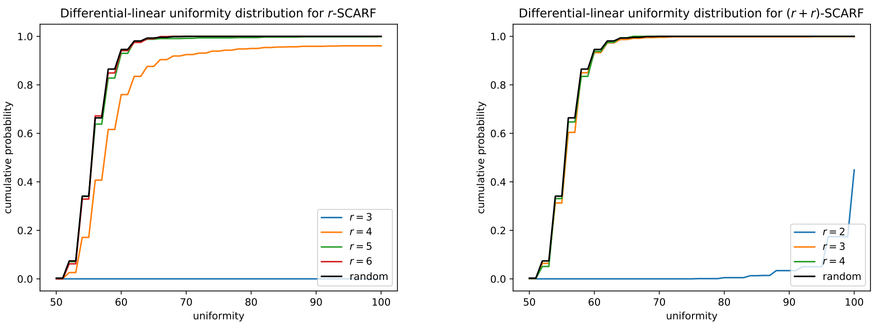


(a) r-round SCARF (b) $(r+r)$-round SCARF

Fig. 14. Cumulative probability distribution for the differential linear uniformity of SCARF.

In a nutshell, a differential-linear attack [23] is obtained by combining a differential and linear trail. More formally, a differential-linear trail of E is given by α, β such that the correlation c given by

$$\Pr_x(\beta, E(x) \oplus E(x \oplus \alpha) = 0) = \frac{1}{2} + \frac{c}{2}$$



(a) r-round SCARF (b) $(r+r)$-round SCARF

Fig. 15. Cumulative probability distribution for the differential linear uniformity of SCARF.

is greater than $2^{-n/2}$, if n is the block size of E.

Similarly to boomerang attacks, the correlation of such a distinguisher for a cipher $E = E_1 \circ E_m \circ E_2$ can be estimated as prq^2 where p is the probability of the differential trail over E_1, q is the correlation of the linear trail over E_2 and r is the correlation of *connecting* the trails [2]. Despite the maximum DLCT of S being 16, since the MDP and maximum linear correlation over two rounds are respectively 2^{-4} and $2^{-4.83}$, E_1 and E_2 cannot be both two or more rounds, so that we do not think it is possible to find differential-linear distinguishers over 6 rounds with probability higher than 2^{-10}, leaving a significant margin of security for the $8+8$ round-cipher $\tilde{E}$ against this kind of attacks.

Figure 15 shows the distribution of the differential linear uniformity (that is, the maximum absolute correlation of any differential-linear trail, multiplied by the cardinality of the domain 2^{10}) of E and $\tilde{E}$ over 2^{10} tweaks. As expected, for $r = 5$ and 6 the distribution of the uniformity of E, as well as that of $\tilde{E}$ with $r \geq 3$, matches the one drawn for random permutations.

B.5 Multiple-Tweak Differential

The multiple-tweak differential cryptanalysis is one of the most powerful attack strategies against SCARF. As reported in Appendix B.1, with a non-zero α and a non-zero β,

$$\Pr_x(E(x) \oplus E(x \oplus \alpha) = \beta)$$

behaves like a random permutation for both E_T and $\tilde{E}_{T_1,T_2}$. The multiple-tweak differential cryptanalysis observes slight bias from the random, i.e.,

$$\Pr_{x,T_1,T_2}(\tilde{E}_{T_1,T_2}(x) \oplus \tilde{E}_{T_1,T_2}(x \oplus \alpha) = \beta) = \frac{1}{1023} + \varepsilon.$$

Algorithm 1 Efficient algorithm to observe very low bias

function COMPUTING_BIAS(N_T, α, β)
    Store N_T random tweaks to $\mathcal{T}_1$
    Store N_T random tweaks to $\mathcal{T}_2$ and $\mathcal{T}_1 \cap \mathcal{T}_2 = \phi$
    Prepare an array S of 2^{20} elements.
    for all $T_1 \in \mathcal{T}_1$ **do**
        for all $P_1 \in \{0,1\}^{10}$ **do**
            if $P_1 \oplus \alpha > P_1$ **then**
                continue
            end if
            $v = E_{T_1}(P_1) \| E_{T_1}(P_1 \oplus \alpha)$
            $S[v] \leftarrow S[v] + 1$
        end for
    end for
    $num \leftarrow 0$
    for all $T_2 \in \mathcal{T}_2$ **do**
        for all $P_2 \in \{0,1\}^{10}$ **do**
            $v = E_{T_2}(P_2) \| E_{T_2}(P_2 \oplus \beta)$
            $num \leftarrow num + S[v]$
        end for
    end for
    return num
end function

In random tweakable permutation, the probability is $\frac{1}{1023}$ because of the permutation, i.e., $\tilde{E}_{T_1,T_2}(x) \oplus \tilde{E}_{T_1,T_2}(x \oplus \alpha) \neq 0$. To observe a bias ε from a random permutation, we need at least $\varepsilon^{-2} \times \frac{1}{1023}$ pairs.

There is a very efficient method to experimentally verify the bias of a fixed input/output difference (α, β), thanks to the extremely small block length and peculiar structure of $\tilde{E}_{T_1,T_2}$. Algorithm 1 shows the pseudo code. We first prepare a set of N_T tweaks, $\mathcal{T}_1$, construct 2^9 pairs $(P_1, P_1 \oplus \alpha)$ for every tweak, and finally store the number of appearances of $(E_{T_1}(P_1) \| E_{T_1}(P_1 \oplus \alpha))$ of all $T_1 \in \mathcal{T}_1$. Similarly, we prepare another disjoint set of N_T tweaks, $\mathcal{T}_2$. Then, for every $T_2 \in \mathcal{T}_2$, we construct 2^{10} pairs $(P_2, P_2 \oplus \beta)$, and sum the number of $(E_{T_2}(P_2) \| E_{T_2}(P_2 \oplus \beta))$ in the stored above. This technique allows us to observe a bias using $N_T^2 \times 2^9$ pairs with a complexity of $N_T \times 2^{10}$ and 2^{20} memory. Usually, we cannot observe a low bias detected using incredibly many pairs first. However, this algorithm enables us to catch it, e.g., we can observe the bias using 2^{55} pairs.

Table 2 shows some examples of observed differential bias, where we used $N_T = 2^{23}$, i.e., $N_T^2 \times 2^9 = 2^{55}$ pairs. We generally observe significant biases when $\alpha = \beta$ and the left branch of α is inactive. When $\alpha \neq \beta$, the bias is lower, and sometimes, it derives a negative bias.

On our main security claim (Security Requirements 1), the number of collectable pair is $N_T^2 \times 2^9 = 2^{21+21+9} = 2^{51}$ even if we use the technique shown in Appendix A.1. These pairs are not always sufficient to distinguish (5+5)-round SCARF with a significant advantage. However, considering the multi-differential, it might be possible to

Round	α	β	Bias ε
2+2	(0x00,0x01)	(0x00,0x01)	$2^{-9.6792}$
3+3	(0x00,0x01)	(0x00,0x01)	$2^{-14.6761}$
4+4	(0x00,0x01)	(0x00,0x01)	$2^{-24.8467}$
5+5	(0x00,0x01)	(0x00,0x01)	$2^{-29.8025}$
4+4	(0x00,0x01)	(0x00,0x02)	$2^{-29.7363}$
4+4	(0x00,0x01)	(0x00,0x03)	$-2^{-29.8138}$
4+4	(0x00,0x01)	(0x00,0x04)	$2^{-26.5813}$
4+4	(0x00,0x01)	(0x00,0x05)	$-2^{-30.6340}$
4+4	(0x00,0x02)	(0x00,0x01)	$2^{-30.1689}$
4+4	(0x00,0x02)	(0x00,0x02)	$2^{-24.8833}$
4+4	(0x00,0x02)	(0x00,0x02)	$2^{-24.8833}$
4+4	(0x00,0x02)	(0x00,0x03)	$-2^{-27.9966}$
4+4	(0x00,0x02)	(0x00,0x04)	$2^{-30.0152}$
4+4	(0x00,0x1F)	(0x00,0x1C)	$2^{-28.1046}$
4+4	(0x00,0x1F)	(0x00,0x1D)	$2^{-28.1773}$
4+4	(0x00,0x1F)	(0x00,0x1E)	$2^{-27.9916}$
4+4	(0x00,0x1F)	(0x00,0x1F)	$2^{-24.4674}$

Table 2. Examples of observed differential bias given experimentally: we used $N_T = 2^{23}$ to get these results, i.e., $2^{23+23+9} = 2^{55}$ pairs.

distinguish (5+5)-round SCARF. Nevertheless, we cannot append (3+3)-round key recovery to this distinguisher faster than 2^{80} because 3+3 rounds involve a 90-bit key.

On the contrary, our bold security claim (Security Requirements 2) allows to collect $N_T^2 \times 2^9 = 2^{29+29+9} = 2^{67}$ pairs. Therefore, (5+5)-round SCARF is distinguishable. It might allow an attacker to distinguish (6+6)-round SCARF[6]. Even if we assume that (6+6) rounds are distinguishable by using 2^{40} oracle queries to $\tilde{O}$, we need to guess the 60-bit key involved in (2+2) rounds to attack the full round. While more effort is needed to better understand this new class of attacks, we believe that it is not possible to attack (8+8) rounds in time significantly lower than 2^{80}.

B.6 Invariant Attacks

Invariant subspace/nonlinear invariant attacks do not seem to pose a threat to SCARF, in particular as our tweakey schedule is nonlinear, and we use a 240-bit secret key generated randomly.

[6] Due to computational limitations, we cannot estimate the bias in the 6+6 rounds. However, as far as we experimentally evaluated the bias of $\alpha = \beta = (\texttt{0x00}, \texttt{0x01})$ using $N_T = 2^{28}$ once, we cannot observe a significant bias. It implies that the bias is not significantly higher than $2^{-37.5}$ (since no bias was observed using 2^{65} pairs, i.e., its standard deviation is about $\sqrt{2^{65-10}} = 2^{27.5}$, and its corresponding probability is $2^{27.5}/2^{65} = 2^{-37.5}$). We expect it to be about 2^{-40} heuristically.