A Minimal White-Box Dedicated Cipher Proposal Using Incompressible Lookup Tables: Space-Hard AES

Félix Carvalho Rodrigues[1], Ricardo Dahab[1], Julio López[1], Hayato Fujii[1] and Ana Clara Zoppi Serpa[1]

[1]Institute of Computing – University of Campinas (Unicamp)
Av. Albert Einstein, 1251 – 13083-852 – Campinas – SP – Brazil

{felix.rodrigues,rdahab,jlopez,hayato.fujii}@ic.unicamp.br,
ra165880@students.ic.unicamp.br

Abstract. *In a white-box context, an attacker has full access to the execution environment and the implementation of cryptographic algorithms. Dedicated white-box ciphers, such as WEM and SPNbox, provide incompressibility and key extraction protections in this context, at the cost of increased memory usage and performance loss compared to standard ciphers. Even when a pure white-box threat model is not warranted, the use of incompressible lookup tables can be helpful in deterring side-channel attacks. In this paper we present a simple threat model for such scenarios and propose a dedicated cipher, Space-Hard AES, which provides minimal incompressibility guarantees while presenting better performance for ARMv8 implementations than other dedicated ciphers.*

1. Introduction

The traditional *black box threat model* assumes that the end points of a communication channel are secure, and only the channel itself is vulnerable to attackers. Several scenarios challenge such assumptions, such as the increased threat of malware and other threats which might compromise machines running cryptographic operations. In a *white-box threat model*, an attacker is assumed to have complete access to the full implementation and the execution environment of a cryptographic algorithm.

White-box cryptography concerns the design and secure implementation of cryptographic algorithms running in untrusted environments. The concept was introduced in 2003 [Chow et al. 2003], exemplified in a software implementation of the AES cipher which attempts to obfuscate a secret key. This proposed implementation was shown to be susceptible to practical attacks [Billet et al. 2005], and every new proposed white-box implementation of the AES [Karroumi 2011] was also successfully attacked [Lepoint et al. 2014, Bos et al. 2016].

These attacks, while effective, require complete knowledge of the implementation. More recently, however, attacks adapting side channel analysis for a white-box context, such as *differential computation analysis* (DCA), have proven to be highly effective against standard AES white-box implementations [Bos et al. 2016], and thus also need to be taken into account. The difficulty in defending against white-box opponents in standard white-box implementations such as the AES block cipher has prompted efforts to design new symmetric ciphers, which take into account white-box-model threats from the start. Most of current proposals focus not only on ensuring that discovering the protected secret key is infeasible, but also on mitigating possible *code lifting* attacks [Bos et al. 2016],

in which the attacker extracts the cryptographic implementation itself, to use it as an effectively larger key, duplicating the functionality of the cipher. However, the countermeasures against code lifting are costly: the large table sizes make their usage unfeasible in several contexts and, even when their use is possible, performance loss is significant, impairing their adoption in realistic settings.

In several scenarios, the pessimistic assumptions of the pure white-box model are not realistic. While attackers have some degree of access to the execution environment, they often lack the full access assumed in a pure white-box model. An attacker might be able to execute software based side channel attacks such as some form of Differential Computation Analysis (DCA), however lack the capabilities of directly copying an implementation or the environment memory. In these scenarios the performance loss from the increased code size of current dedicated ciphers such as WEM and SPNBox are often not acceptable, and a minimal cipher that retains incompressibility is warranted.

In this paper we present a threat model which lies between a pure white-box threat model and the grey-box model, with an accompanying dedicated cipher proposal which efficiently achieves security in this model. We show its security with regards to space-hardness and key extraction. Furthermore, we delve into implementation details in an ARMv8 environment for our proposed cipher, showing how its performance and security is improved when compared to other dedicated ciphers such as SPNBox [Bogdanov et al. 2016] and WEM [Cho et al. 2017], even when using a constant-time implementation.

1.1. Related Work

After Chow et al. [Chow et al. 2003] presented the first white-box implementation of the AES cipher, Billet et al. [Billet et al. 2005] were the first to present a feasible way to discover the secret key embedded in the white-box implementation, nowadays referred as the BGE attack. Following this, several implementations were proposed for the AES cipher [Xiao and Lai 2009, Karroumi 2011]. Unfortunately, all known (academic) white-box implementations of these block ciphers have been broken [Billet et al. 2005, Goubin et al. 2007, Michiels et al. 2009, De Mulder et al. 2010, Lepoint et al. 2014, De Mulder et al. 2013] through implementation specific attacks. These attacks require knowledge of the underlying proposed implementation, which diminished their impact for secretly designed white-box ciphers. However, in [Bos et al. 2016], Bos et al. coined the term differential computation analysis to describe a novel technique which adapts a common side channel attack, the differential power analysis (DPA), to the white-box context. It uses software traces to discover information on secret keys. These software traces contain recorded memory accesses from cryptographic algorithm implementations, and using some manipulation can be turned into noise free power traces, which can be used by DPA statistical tools to recover secret information on white-box cipher implementations.

The first dedicated white-box cipher was proposed in 2014 [Biryukov et al. 2014]. Its design was based on an *ASASA* structure, and its security relied on the hardness of decomposing its layers. Subsequent cryptanalysis revealed possible vulnerabilities in this structure [Biryukov and Khovratovich 2015]. Bogdanov et al. [Bogdanov and Isobe 2015] introduced the SPACE family of dedicated block ciphers,

with a focus on using proven standard cryptographic primitives to guarantee their security. Shortly after this proposal, they also proposed the SPNbox [Bogdanov et al. 2016] as a less conservative design intent on making white-box implementations more practical. In [Cho et al. 2017], the WEM family of ciphers is introduced, based on an Even-Mansour scheme where the secret key layers are replaced by secret incompressible S-boxes. The WhiteBlock block cipher [Fouque et al. 2016] is similar to the WEM design as it uses a standard block cipher as a public permutation between each S-box layer, differing mostly on how the S-box layer is constructed. Its main contribution is in providing a more rigorous proof of its security goals when compared to other dedicated ciphers.

In the context of ARMv8 cryptographic implementations, most works focus on standard block ciphers and modes of operation, such as the AES and the GCM [Gouvêa and López 2015]. In [Rodrigues et al. 2019], the SPACE, WEM and SPNbox proposals were compared in regards to their performance for ARMv8 machines. They implemented several versions of the proposed ciphers for ARMv8 machines and provided fair comparisons of all of them using a CTR mode of operation to allow further parallelism. Their results indicate that SPNbox currently presents the best performance for ARMv8 implementations.

2. Preliminaries

A symmetric encryption scheme is a tuple $\mathcal{E} = (\mathcal{K}, \mathcal{M}, \mathcal{C}, G, E, D)$, where $\mathcal{K}, \mathcal{M}$ and $\mathcal{C}$ are the set of possible keys, plaintexts (messages) and ciphertexts, respectively, while G, E and D are the functions for key generation, encryption and decryption, respectively. For any $k \in \mathcal{K}$ and $m \in \mathcal{M}$, $D(E(m, k), k)$ must be equal to m. Note that we alternatively use $E(m)$ or $E_k(m)$ for denoting encryption (similarly for decryption), when the context is clear. A white-box compiler $C_\mathcal{E}$, takes a symmetric encryption scheme $\mathcal{E}$, a key $k \in K$, a nonce r (optionally) and returns a compiled white-box program $C_\mathcal{E}(k, r) = [E_k]$ (respectively $[D_k]$ for the decryption).

The main goal of a secure compiler is *unbreakability*: given a program $[E_k]$, the key k embedded in $[E_k]$ must not be discovered efficiently by any adversary program $\mathcal{A}$.

Definition 1 (Unbreakability)**.** *Let $\mathcal{A}$ be a PPT adversary algorithm. A program $C_\mathcal{E}(k, r) = [E_k]$ is considered unbreakable if for any key $k \in K$, any randomly generated number r, adversary $\mathcal{A}$ when run on $[E_k]$ returns a guess $k' \in \mathcal{K}$ different than k with a probability of $1 - \alpha(|[E_k]|)$, where α is a negligible function.*

Another important security notion for dedicated white-box ciphers is incompressibility, which is related to the mitigation of *code-lifting* attacks [Bos et al. 2016].

Space Hardness. The notion of incompressibility has been referred to with multiple denominations in the literature. Bogdanov et al. [Bogdanov et al. 2016] use the term *space hardness* to refer to a notion related to incompressibility, defining both weak and strong space hardness. In the weak space hardness security notion, an adversary must not be able to encrypt or decrypt a randomly drawn message or ciphertext with less than M bits of the compiled cipher's code, while in strong security notion, the adversary must not encrypt or decrypt any messages given M bits of the code.

Definition 2 (Weak $(M, Z)-$space hardness)**.** *Given an encryption scheme $\mathcal{E}$, a white-box compiler $C_\mathcal{E}$ is weakly $(M, Z)-$space hard if it is infeasible for an adversary $\mathcal{A}$ to*

encrypt (or decrypt) a randomly drawn plaintext (or ciphertext) with probability of more than 2^{-Z}, given access to M bits from $[E_k]$ (or $[D_k]$).

Additionally, we note the similarities between incompressibility as a code-lifting mitigation strategy with the *Bounded Retrieval Model* [Bellare et al. 2016]. In this model, an absolute leakage parameter l is defined, indicating how much data can be leaked without detection by the user's system. All sensitive data must therefore be incompressible and large enough so that a leakage of size l does not compromise them.

2.1. Software Protection Mechanisms

In practice, white-box cryptography is often only a small part of the software protection mechanism, where it is used in conjunction with additional protection techniques. Some examples of these techniques are summarized below.

Control flow obfuscation. This method essentially tries to shuffle the order of table accesses that are performed in the execution of each round of an encryption operation. As an example, each round in the AES encryption process can be seen as a parallel application of four encoded subrounds. The order of execution of those subrounds can be shuffled using a PRNG as it does not make any difference in the final result. Furthermore, inside each subround, the order of table accesses can be shuffled. In the general case, a dependency graph for table accesses can be built to assess which tables must be accessed before the others. Nodes residing at the same level in the graph represent table accesses whose execution order can be randomly shuffled.

Table location randomization. Such method is akin to the masking countermeasure applied in hardware architectures to thwart power attacks. Memory addresses are protected by adding random offsets to them. This effectively disperses tables at random locations in the address space.

Dummy operations. In a standard DPA attack, the attacker computes the correlation coefficient $p_j(t)$ for each key guess K_j for all the time range for which he obtains power traces. The internal variable V which the power attack targets is usually computed in the first round itself, and thus samples that indicate the power consumption for V are likely to be located in a small time range at the beginning of the power trace for each plaintext. For a system that does not employ disarrangement, the time range over which the intermediate variable V is calculated is likely to be very similar across distinct iterations of trace recordings, where each trace is obtained by feeding a new plaintext to the cipher. Thus, it becomes easy to compute the correlation coefficients. The correct key guess is likely the one which maximizes $p_j(t)$ for some t. The idea of disarrangement is to randomize the time instance at which V is computed for each execution of the encryption operation. As a result, for each new plaintext, V is likely to be computed at different time instances. Therefore, in the power traces, the time instances at which V is computed no longer align with each other. As so, it becomes more difficult to mount a power attack. In the context of white-box encryption, disarrangement is achieved by adding a random number of dummy table lookups in between each legitimate table access. This essentially breaks the alignment pattern for table accesses for each new plaintext for which traces are recorded.

2.2. ARM TrustZone

TrustZone is the ARM implementation of a Trusted Execution Environment (TEE), virtually providing two processors running the Normal World and the Secure World, respec-

tively. A TEE is used in many activities in a smart device: secure boot of an authorized OS by the manufacturer of the device, keyring implementation, OS protection against non-authorized modifications, secure payments, digital rights management (DRM), disk encryption, and more. Separation of the Normal and the Secure World is purely virtual, as the same physical core is used to run both contexts. This architecture extends the concept of *privilege rings*, in which certain operations, i. e., access to specified memory address, are limited only to privileged processes such as the OS but not by applications running in the userspace. In the ARMv8 scenario, those rings are called *Exception Levels* (EL), among which EL0 is the userspace mode, EL1 is the supervisor, EL2 is typically the hypervisor and EL3 runs the trusted firmware or Secure Monitor. In each level, except for EL2, the Secure World can be enabled. Typically, a richly-featured OS runs on the Non-Secure EL1 ring, and a *Secure OS* runs on the Secure EL1 ring. While the former runs all necessary tasks for an OS, the Secure OS has a smaller footprint and functionality to diminish the attack surface, concerning itself mostly with security functionality.

At any given time, the CPU can only be run in one of the modes: Secure or Non-secure. The transition between such modes is done by interruptions, called from the CPU from a few ways such as from a peripheral or from a timer or caused by an instruction trap. Separation of the worlds is enforced by hardware-based access control, allowing communication between the Secure OS with peripherals and a protected memory range, inaccessible by the Normal World. This allows secure communication between, for example, a fingerprint scanner and the process to acknowledge or refuse user authentication, without leaking private data through the Normal World. In addition, the hardware mechanism allows memory pages residing in the Normal World to be accessed by both worlds; the Secure World maps into a process virtual memory as needed.

As an example, for key management purposes, a device manufacturer may store in hardware (i. e., mask ROM) a secure key; then, in the Secure OS, the CPU generates keys and encrypts them using the hardware key, storing the result in persistent memory in a key blob, consisting of encrypted keys and related metadata. Applications running in the Normal World, which does not know the actual, secured key, may request encryption or decryption functionality through the Secure OS, which handles the usage of keys.

Side Channel Attacks on TrustZone. As main memory is shared between Normal and Secure worlds, both contexts also share different cache levels to improve access times. This can be leveraged to mount cache timing attacks. Thus, non-secured implementations of ciphers, such as a table-based implementation of AES is susceptible to having its secret keys revealed, even if execution occurs in a safe environment [Lapid and Wool 2018]. This vulnerability surfaces from the TrustZone design decision of sharing memory across worlds, specially when cache memory is available.

More powerful attacks, such as correlation power analysis (CPA) are capable of extracting the secrets while running an AES encryption operation, for example. Eletromagnetic emissions of an ARM CPU with TrustZone capabilities can be measured and correlated to a secret, effectively breaking security, even if the implementation runs in the Secure World [Bukasa et al. 2017]. While attackable, trace collection, taking 7 hours, and further data processing exceeding 70 hours of CPU time makes the attack limited for those with enough computational resources. It must be noted that, as expected from the TrustZone solution being a purely software-based protection, evidences of CPU work-

load under stress are still present, as no protection was given for such experiment. Thus, implementations must be secured against side-channel attacks, even if a TEE is present.

3. Weak White-Box Threat Model

While the already proposed dedicated ciphers such as SPNBox [Bogdanov et al. 2016] and WEM [Cho et al. 2017] provide excellent protection against key extraction, relying only on their incompressibility or space-hardness to prevent code-lifting is not recommended in practice. Without other obfuscation and device binding techniques, the usage of such ciphers becomes restricted to scenarios where it is viable to store gigabytes of lookup tables dedicated only towards symmetric encryption. Even in such scenarios, the size of such lookup tables presents several challenges in regards to performance. Memory operations must be performed in higher levels, incurring harsh performance penalties. Furthermore, due to its incompressible nature, cache hit ratio necessarily suffers unless the full table can be allocated inside the cache layer.

In industry, measures such as the ones summarized in Section 2.1 are used in conjunction with white-box solutions. Furthermore, designs are kept secret in an attempt to further increase the difficulty of reverse engineering. For such scenarios, incompressible lookup tables of smaller sizes are desirable, which minimize performance loss from lookup tables. Their usage provides strong intrinsic protection against most side-channel attacks, since its design guarantees full diffusion of key material throughout the whole lookup table.

In our model, besides black-box and traditional side channel leaks, only access to (noise-free) computational traces of execution are available to the attacker, without explicit memory position access. A computation trace contains accessed values during the execution of an implementation of a cryptographic algorithm. Each trace $\mathbf{v}$ consists of $(v_1, v_2, \ldots, v_t)$ samples, for all $v_j \in \mathcal{V}$, where $j \in [1, t]$ and $\mathcal{V}$ is a set of possible values that can vary according to the particular execution environment.

Definition 3 (Weak White-Box Threat Model)**.** *An attacker in the Weak White-Box Model has access to a collection of n computational traces (with corresponding plain and cipher texts) and side-channel access to the encryption/decryption program or circuit.*

While the attacker cannot directly read the executable code of the implemented cryptographic algorithm, he can analyze computational traces in order to find correlations which might reveal information of either the secret key used in the encryption or of the lookup tables which were generated using it. It is a similar concept to the one presented by Rivain and Wang [Rivain and Wang 2019], of a passive adversary with access to computational traces.

This model is implicitly used when designing secret software implementations of standard ciphers such as the AES for usage in smartphones and other devices which are not trustworthy. When designing protection measures for these standard ciphers, the implementer is at a disadvantage, since these standard ciphers are not allowed to be changed in any manner. Therefore, protection measures usually depend on masking and encoding techniques which can be vulnerable to DCA-like attacks, since fundamental properties of the cipher have not been changed, such as the amount of key material added to each byte of the state on each round.

An implementation which attempts to be secure in this environment can be considered orders of magnitude slower than black-box implementations of standard ciphers, defeating the purpose of these ciphers in several practical scenarios. Even when protecting only against hardware side-channel attacks, such as when using these cryptographic primitives in a Trusted Execution Environment like the one provided by ARM's TrustZone (see Section 2.2), the performance toll caused by the need of using such protection techniques (or the security vulnerability when foregoing these measures) can be considerable.

However, if we are allowed to propose a new design, incompressible techniques can guarantee a stronger and more efficient protection to both standard hardware side-channel attacks and to enhanced software side-channel attacks. In Section 4 we show a possible design with this model in mind, guaranteeing strong resistance for passive software and hardware side-channel attacks while performing close to black-box implementations of standard ciphers.

3.1. Possible Usage Scenarios

We envision two main venues of usage for dedicated ciphers with low level of space hardness. One where hardware protection allows the construction of effective TEEs, which, while still vulnerable against side channel attacks, are highly effective in preventing code lifting attacks, and scenarios where a combination of hardware and application binding are effective against code lifting attacks.

Trusted Execution Environment (TEE). As seen in Section 2.2, even though a TEE is supposed to ensure protection against most attacks, it is not feasible cost-wise to properly protect it against most hardware side-channel attacks, since ARM's TrustZone lives in the same chip as its main processor. Software protection against side channel attacks in standard cryptographic implementations can be costly, often affecting performance in orders of magnitude. However, as a protection against code-lifting attacks, ARM's TrustZone and other TEE are very effective, as memory access is heavily restricted. Incompressible lookup tables are a strong and efficient protection against most side-channel attacks, as they often ensure that key guesses must use the full guess space to discover even a single byte of the original key. Thus, the design of new cryptographic ciphers which use incompressible lookup tables as a way of ensuring protection against side-channel attacks for usage inside TEEs can be recommended as a way to maintain performance while ensuring protection against side-channel attacks.

Another possible combined usage is in regards to generation of session keys. One can envision a system where the protected zone acts as a compiler of the incompressible tables, which can be used in user space in order to minimize possible performance loss associated with communication and usage of the protected zone.

Combining White Box and Hardware Binding. In Bock et al. [Bock et al. 2019], the combination of white box techniques and hardware binding solutions are shown to be conceptually sound. They focus on its usage in mobile payment applications, using hardware and application binding to ensure protection against code-lifting attacks. They expand this idea in [Alpirez Bock et al. 2020], where they propose indistinguishability for white-box cryptography with hardware-binding as a new security notion which allows one to design white-box applications with security against code-lifting attacks. For hardware binding,

the objective is that the white-box cipher should only be executable on the intended device. Thus, the white-box program can be evaluated when accessed from a specific device, but is unreachable from anywhere else.

When methods for binding cryptographic functions to devices are available, nullifying code-lifting attacks, our threat model becomes even more relevant, since its main concern is with the protection against key extraction.

4. Space-Hard AES

In this section we show a proposal which aims to ensure that a cipher using the same instructions as the standard AES cipher has acceptable space-hardness, and consequently a good level of protection against both software and hardware side-channel attacks.

4.1. Design

The cipher is based on the AES-128, with ten rounds. The main idea is that instead of performing a key addition, one replaces this step by using an incompressible white-box S-Box. The code can be seen in Algorithm 1.

Algorithm 1 Space-Hard AES Enc

Input: A 128-bit block of plaintext and an 8-to-8 bit incompressible S-Box S.
Output: The 128-bit encrypted message.

state $\leftarrow$ plaintext
state $\leftarrow S$(state)
for r from 1 to 9 **do**
  state $\leftarrow$ `SubBytes`(state)
  state $\leftarrow$ `ShiftRows`(state)
  state $\leftarrow$ `MixColumns`(state)
  state $\leftarrow S$(state)
end for
state $\leftarrow$ `SubBytes`(state)
state $\leftarrow$ `ShiftRows`(state)
state $\leftarrow S$(state)
return state

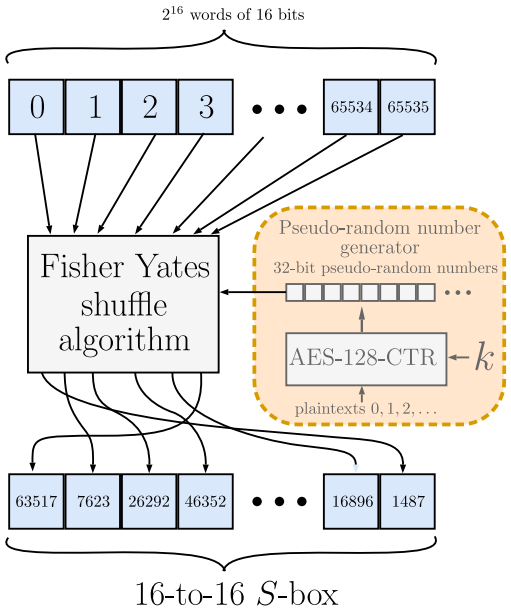

Figure 1. 16-to-16 bit S-box S example generation.

By only using a single incompressible 256 byte lookup table, we ensure that performance is minimally impacted, while still maintaining a strong intrinsic defense against DCA-like attacks. One can generate an incompressible lookup table with multiple approaches. If one wishes to link the security of its incompressible S-Box to well known black-box primitives, the approach detailed by the WEM family of ciphers guarantees such security, at the cost of an extremely expensive black-box decryption. If a black-box instance capable of decoding multiple different keyed compiled instances is needed, the approach taken by the SPNbox family of ciphers provides the best guarantees, even though its security is linked to smaller ciphers without relevant cryptanalysis effort. In this work we consider our incompressible S-Boxes to be generated as described by the WEM

cipher [Cho et al. 2017]. The generation is as follows. In a secure environment, a long sequence of random bits is generated depending on the secret key k, e.g. using an AES based CTR-DRBG; then, the sequence is used as a way to provide random numbers to a shuffler (such as the Fisher Yates algorithm [Knuth 1998]) which permutes a sequence $(0, 1, \ldots, 2^m - 1)$ to form an m-to-m S-box. Figure 1 illustrates the shuffling method.

In the design of this cipher, we assumed our executing environment to be using the ARM architecture, more specifically the ARMv8-A version with the hardware aided AES instructions present in the Cryptographic Extension. If one assumes an ARMv8 architecture, we can compose the generated S-Box with the `SubBytes` operation, allowing the full usage of the Crypto Extension instructions, which can perform a single `MixColumns` operation. However, by design it is also optimal for usage in Intel architectures containing AES-NI instructions. In these machines, the AES-NI operations, unlike its ARMv8 counterparts, perform a full round of encryption, therefore the S-Box must not compose with the `SubBytes` operation.

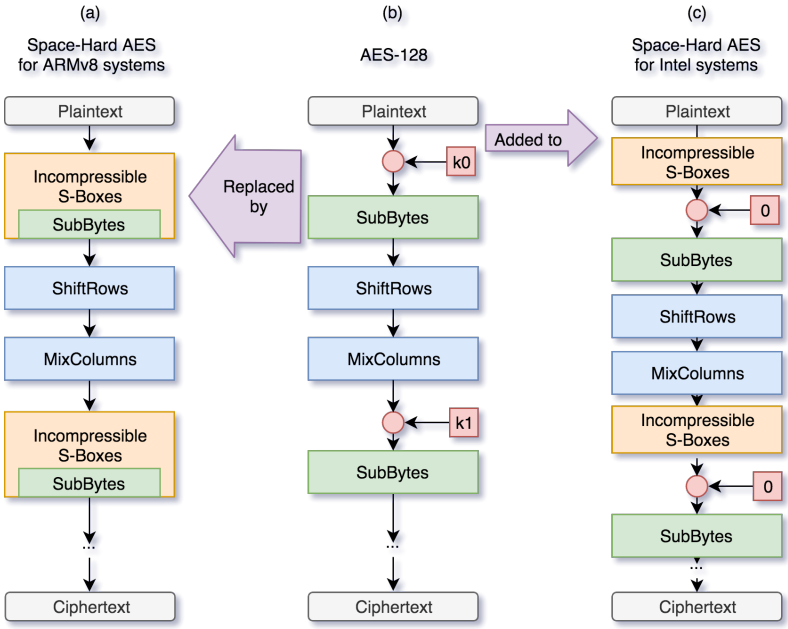

Figure 2. Possible implementation strategies of Space-Hard AES while taking advantage of cryptographic instructions in both ARMv8 and Intel.

When compared to other ciphers shown in this work, we can see that it is supposed to have the best performance, since its diffusion layers are smaller than both WEM and SPNbox ciphers, and its non-linear layer has the same size as the SPNbox-8 cipher. In Figure 2 we present a round of the cipher compared to the AES cipher and a hypothetical Intel version. Note that while here we instantiate a version with a single 8-to-8 lookup table, larger and more numerous tables may be used when desired.

4.2. Security

When regarding its black-box security, its AES design, with the table lookup taking place of the key addition, ensures that its security remains the same as the traditional AES cipher. In regards to the key-extraction security of its incompressible lookup table, it

directly depends on the manner in which it is generated. We do not propose a new method for generation, instead relying on the security of the methods used in either the WEM [Cho et al. 2017] or in the SPNbox family of ciphers [Bogdanov et al. 2016].

DCA attacks and similar side-channel attacks rely on each lookup table depending only on a fraction of the key, which is extracted by computing correlations of key byte guesses and traces of execution. The major difference when comparing to these dedicated white-box ciphers is that each lookup table incorporates at least 128-bit secret key information, therefore implying that the key space for such attacks is at least 2^{128}, making any such attacks unfeasible. Cache and timing attacks also are ineffective at retrieving the embedded key due to the same reasoning.

As for differential fault analysis (DFA), where faults are injected at specific byte positions, modifying its state, again the fact that the attacker has only access to the compiled implementation means that any such technique reduces to a differential attack on the cipher or pseudo-random generator used to generate its lookup tables.

A possible, though yet unreported, code-lifting attack regards a case where DCA-like techniques are used to recover the lookup tables used in the cipher, instead of attempting to recover the secret key. In this scenario it might be possible to design a secret diffusion layer similar to the external encodings present in the AES implementation designs of Chow et al. [Chow et al. 2003] which is able to prevent these attacks. Such scenario might provide other avenues of research in both dynamic diffusion layers and encodings. Poor implementations that fail to be constant time might also lead to such code-lifting attacks, due to leakage of its lookup tables by cache or time side channel attacks.

Finally, regarding its space hardness, we directly use the formula derived by Bogdanov et al. [Bogdanov et al. 2016]: given a size parameter M, the total table size $T = 2^8 \times 8$, the number of accesses per round $t = 16$ and the number of rounds $R = 11$, the probability of successful decryption given a random plaintext is

$$2^{-Z} = \left(\frac{M}{T}\right)^{tR} = \left(\frac{M}{T}\right)^{176},$$

i.e. for an $M = T/2$, its probability of decryption is 2^{-176}, much lower than simply guessing at random. In fact, if we wish for a parameter $Z = -128$, the parameter M might be as high as $T/(2^{0.727})$, which is comparable to the SPNbox8 cipher, with the same total size $T = 256$ bytes.

4.3. Implementation Details

For the implementation of the different ciphers, several techniques were used. Most are also present in [Rodrigues et al. 2019] for the implementation of already proposed dedicated white-box ciphers, such as instruction pipeline optimizations (described as n-way implementations in our results) and memory coverage optimizations to reduce cache misses. We briefly show how to implement the required table lookups in a constant time fashion for Space-Hard AES, and then show details on the proposed cipher implementation.

Constant Time Table Lookups. In order for our proposed ciphers to run in constant time, one must make sure that lookup table accesses are constant time operations. For the space

hard AES proposal with a single table, this can be accomplished efficiently by keeping the whole lookup table in 16 NEON registers, using four `ld1` operations, and then using four `tbl` instructions to execute each table lookup. One way to achieve this is to first apply an XOR operation between the state and masks $[0x40, \ldots, 0x40]$, $[0x08, \ldots, 0x80]$, and $[0xC0, \ldots, 0xC0]$, storing the results in registers `v1, v2` and `v3`. Then, assuming the table is loaded in registers `v16` to `v31`, do the following operations:

```
tbl v0.16b, {v16.16b, v17.16b, v18.16b, v19.16b}, v0.16b
tbl v1.16b, {v20.16b, v21.16b, v22.16b, v23.16b}, v1.16b
tbl v2.16b, {v24.16b, v25.16b, v26.16b, v27.16b}, v2.16b
tbl v3.16b, {v28.16b, v29.16b, v30.16b, v31.16b}, v3.16b .
```

The lookup table access for 16 bytes can then be obtained by adding the registers `v0, v1, v2` and `v3` in $GF((2^8)^{16})$. An alternative is to use a combination of `tbl` or `tbx` instructions to execute each table lookup by subtracting from the state between `tbl`/`tbx`. Such usage is present on the Linux Kernel AES implementation geared towards ARMv8 CPUs without cryptographic extensions. Since this setup requires a large number of registers, a horizontal implementation alleviates this restriction by allowing the reuse of these registers for the constant time finite field operations after each γ layer is completed.

Space-Hard AES versions. Since Space Hard AES relies on many steps of the AES cipher, ARMv8 Cryptography Extention (CE) instructions can be used at large to implement the first. The first version relies on computing the byte substitutions using the `tbl` and `tbx` instructions to replace the `AddRoundKey` layer from AES; then, `aese` with a null key and `aesmc` can be issued to execute the remaining AES operations. The decryption operation can be implemented in the same way, only properly reordering layers and using inverse AES instructions. This implementation can be optimized towards pipelining; to avoid costly memory operations, processing four 128-bit words is optimal in a register pressure standpoint.

A second version doesn't rely on ARMv8 CE instructions, widening the number of platforms in which the cipher can be run. To increase performance, this implementation relies on a combined `SubBytes` with `AddRoundKey` (which, per se, it is another substitution lookup) tables, requiring a single `tbltbx` (and helper instructions such as `sub`) to implement both substitutions. The diffusion layer is implemented akin to the Linux Kernel AES implementation geared towards ARMv8/ARMv7 CPUs without the CE instructions support. To further speedup the non-CE version, an optimal formula [Fujii et al. 2019] to compute the `MixColumns` layer was implemented, encrypting four 128-bit blocks at a time.

5. Experimental Results

For the performance comparisons with our proposals' implementations, a Cortex-A76 was used, present in the Qualcomm®Snapdragon™855 Mobile Hardware Development Kit. Ciphers were cross-compiled to the Android platform using `clang` version 8.0.2 for all cores with flag `-O3` enabled. The performance test was done using the CTR mode of operation to encrypt messages of size 2KiB or 16KiB for 2^{15} iterations in which each iteration takes as input the output of the previous one; the first message was sampled from `/dev/urandom`.

Following the nomenclature of [Rodrigues et al. 2019], there are pipeline-optimized versions (4 or 8-way), cache-optimized versions ("Hway"), and single block

Table 1. Measured performance (in cycles per byte) of different versions of proposed dedicated ciphers in CTR mode of operation in the Cortex-A76 core. The "# Loads" column shows the quantity of loads/table lookups needed to cipher a single 128-bit block.

Implementation	Constant Time	Lookup Tables		Performance (cpb)	
		Size (KiB)	# Loads	2048 B	16 KiB
SpaceHAES-CE-1way (ours)	✓	0.25	176	25.811	25.808
SpaceHAES-Crypto-4way (ours)	✓	0.25	176	9.263	9.251
WEM16-Hway [Rodrigues et al. 2019]	×	13.3	104	102.884	145.147
SPNbox8-LUT-Hway [Rodrigues et al. 2019]	×	0.25	160	17.83	17.63
SPNbox8-CT-Hway [Rodrigues et al. 2019]	✓	0.25	160	42.685	42.622
SPNbox16-8way [Rodrigues et al. 2019]	×	128	80	21.21	21.01

version (equivalent to a "1-way" version). Pipeline-oriented versions were built on the fact that there are 32 NEON registers, hence avoiding register spilling and penalties associated with it. For the SPNbox cipher, we additionally present a version using lookup tables as a way to speed up the θ and γ layers, here referred as the "LUT" version. The `SPNbox-8-CT-Hway` version refers to a constant time white-box implementation comparable to the implementation of our proposed Space-Hard AES cipher. For further details, please refer to [Rodrigues et al. 2019]. The `SpaceHAES-Crypto-4way` refers to the pipelined 4-way implementation which uses the hardware AES instructions, while the `SpaceHAES-CE-1way` implementation refers to the implementation without using the Cryptographic Extension (note that all other implementations use these instructions where possible).

The experiments displayed in Table 1 show that the performance of Space Hard AES can be about two times faster than the state-of-the-art dedicated white-box cipher with comparable space hardness. When comparing only constant time implementations, the advantage is even greater, with it being more than 4 times faster than the SPNBox8 implementation. We also highlight that the implementation without usage of the Cryptographic Extension from ARM is also competitive with the best known implementations of currently proposed dedicated ciphers.

6. Conclusions

In this paper we presented a threat model between the classical white-box and grey-box models, where attackers have access to traces of computation instead of full control of the executing environment. For this context, we proposed a minimal dedicated white-box cipher which guarantees incompressibility and security against key extraction.

In the Space-Hard AES proposal, we add incompressible lookup tables to the design of the AES cipher, which in turn efficiently protects its design from both software and hardware side channel attacks. The proposal provides a comparable level of space-hardness to other proposed dedicated white-box ciphers with tables of similar size, such as SPNbox8, while improving performance and ensuring the possibility of efficient constant-time implementations.

In several contexts where white-box solutions are required, their usage is restricted to key generation for standard block ciphers which then operate via a black-box implementation. While Fouque et al. [Fouque et al. 2016] proposed a white-box generator, its

performance is not comparable to newer dedicated ciphers such as SPNbox or our own proposals, therefore it would be of interest to research whether it is possible to achieve higher performance in key generation by adapting our designs towards this goal.

References

Alpirez Bock, E., Amadori, A., Brzuska, C., and Michiels, W. (2020). On the security goals of white-box cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(2):327–357.

Bellare, M., Kane, D., and Rogaway, P. (2016). Big-key symmetric encryption: Resisting key exfiltration. In *Proceedings, Part I, of the 36th Annual International Cryptology Conference on Advances in Cryptology — CRYPTO 2016 - Volume 9814*, pages 373–402. Springer-Verlag.

Billet, O., Gilbert, H., and Ech-Chatbi, C. (2005). Cryptanalysis of a white box aes implementation. In Handschuh, H. and Hasan, M. A., editors, *Selected Areas in Cryptography*, pages 227–240, Berlin, Heidelberg. Springer Berlin Heidelberg.

Biryukov, A., Bouillaguet, C., and Khovratovich, D. (2014). Cryptographic schemes based on the asasa structure: Black-box, white-box, and public-key (extended abstract). In Sarkar, P. and Iwata, T., editors, *Advances in Cryptology – ASIACRYPT 2014*, pages 63–84, Berlin, Heidelberg. Springer Berlin Heidelberg.

Biryukov, A. and Khovratovich, D. (2015). Decomposition attack on SASASASAS. Cryptology ePrint Archive, Report 2015/646.

Bock, E. A., Brzuska, C., Fischlin, M., Janson, C., and Michiels, W. (2019). Security reductions for white-box key-storage in mobile payments. Cryptology ePrint Archive, Report 2019/1014.

Bogdanov, A. and Isobe, T. (2015). White-box cryptography revisited: Space-hard ciphers. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 1058–1069, New York, NY, USA. ACM.

Bogdanov, A., Isobe, T., and Tischhauser, E. (2016). Towards practical whitebox cryptography: Optimizing efficiency and space hardness. In *Advances in Cryptology – ASIACRYPT 2016*, pages 126–158, Berlin, Heidelberg. Springer Berlin Heidelberg.

Bos, J. W., Hubain, C., Michiels, W., and Teuwen, P. (2016). Differential computation analysis: Hiding your white-box designs is not enough. In *CHES*, volume 9813 of *Lecture Notes in Computer Science*, pages 215–236. Springer.

Bukasa, S. K., Lashermes, R., Bouder, H. L., Lanet, J., and Legay, A. (2017). How trustzone could be bypassed: Side-channel attacks on a modern system-on-chip. In *WISTP*, volume 10741 of *Lecture Notes in Computer Science*, pages 93–109. Springer.

Cho, J., Choi, K. Y., Dinur, I., Dunkelman, O., Keller, N., Moon, D., and Veidberg, A. (2017). Wem: A new family of white-box block ciphers based on the even-mansour construction. In Handschuh, H., editor, *Topics in Cryptology – CT-RSA 2017*, pages 293–308, Cham. Springer International Publishing.

Chow, S., Eisen, P., Johnson, H., and Van Oorschot, P. C. (2003). White-box cryptography and an aes implementation. In Nyberg, K. and Heys, H., editors, *Selected Areas in Cryptography*, pages 250–270, Berlin, Heidelberg. Springer Berlin Heidelberg.

De Mulder, Y., Roelse, P., and Preneel, B. (2013). Cryptanalysis of the xiao – lai white-box aes implementation. In Knudsen, L. R. and Wu, H., editors, *Selected Areas in Cryptography*, pages 34–49, Berlin, Heidelberg. Springer Berlin Heidelberg.

De Mulder, Y., Wyseur, B., and Preneel, B. (2010). Cryptanalysis of a perturbated white-box aes implementation. In Gong, G. and Gupta, K. C., editors, *Progress in Cryptology - INDOCRYPT 2010*, pages 292–310, Berlin, Heidelberg. Springer Berlin Heidelberg.

Fouque, P.-A., Karpman, P., Kirchner, P., and Minaud, B. (2016). Efficient and provable white-box primitives. In Cheon, J. H. and Takagi, T., editors, *Advances in Cryptology – ASIACRYPT 2016*, pages 159–188, Berlin, Heidelberg. Springer Berlin Heidelberg.

Fujii, H., Rodrigues, F. C., and López, J. (2019). Fast AES implementation using armv8 ASIMD without cryptography extension. In *ICISC*, volume 11975 of *Lecture Notes in Computer Science*, pages 84–101. Springer.

Goubin, L., Masereel, J.-M., and Quisquater, M. (2007). Cryptanalysis of white box des implementations. In Adams, C., Miri, A., and Wiener, M., editors, *Selected Areas in Cryptography*, pages 278–295, Berlin, Heidelberg. Springer Berlin Heidelberg.

Gouvêa, C. P. L. and López, J. (2015). Implementing GCM on ARMv8. In *Topics in Cryptology — CT-RSA 2015*, pages 167–180, Cham. Springer International Publishing.

Karroumi, M. (2011). Protecting white-box aes with dual ciphers. In Rhee, K.-H. and Nyang, D., editors, *Information Security and Cryptology - ICISC 2010*, pages 278–291, Berlin, Heidelberg. Springer Berlin Heidelberg.

Knuth, D. E. (1998). *The art of computer programming, Volume II: Seminumerical Algorithms, 3rd Edition*. Addison-Wesley.

Lapid, B. and Wool, A. (2018). Navigating the samsung trustzone and cache-attacks on the keymaster trustlet. In *ESORICS (1)*, volume 11098 of *Lecture Notes in Computer Science*, pages 175–196. Springer.

Lepoint, T., Rivain, M., De Mulder, Y., Roelse, P., and Preneel, B. (2014). Two attacks on a white-box aes implementation. In Lange, T., Lauter, K., and Lisoněk, P., editors, *Selected Areas in Cryptography – SAC 2013*, pages 265–285, Berlin, Heidelberg. Springer Berlin Heidelberg.

Michiels, W., Gorissen, P., and Hollmann, H. D. L. (2009). Cryptanalysis of a generic class of white-box implementations. In Avanzi, R. M., Keliher, L., and Sica, F., editors, *Selected Areas in Cryptography*, pages 414–428, Berlin, Heidelberg. Springer Berlin Heidelberg.

Rivain, M. and Wang, J. (2019). Analysis and improvement of differential computation attacks against internally-encoded white-box implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):225–255.

Rodrigues, F. C., Fujii, H., Zoppi Serpa, A. C., Sider, G., Dahab, R., and López, J. (2019). Fast white-box implementations of dedicated ciphers on the ARMv8 architecture. In *Progress in Cryptology – LATINCRYPT 2019*, pages 341–363, Cham. Springer International Publishing.

Xiao, Y. and Lai, X. (2009). A secure implementation of white-box aes. In *2009 2nd International Conference on Computer Science and its Applications*, pages 1–6.