IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2021, No. 1, pp. 78–129. DOI:10.46586/tosc.v2021.i1.78-129

MOE: Multiplication Operated Encryption with Trojan Resilience

Olivier Bronchain*, Sebastian Faust†, Virginie Lallemand*,‡, Gregor Leander‡, Léo Perrin◇ and François-Xavier Standaert*

* Crypto Group, ICTEAM Institute, UCLouvain, Louvain-la-Neuve, Belgium.
fstandae@uclouvain.be, olivier.bronchain@uclouvain.be
† Chair of Applied Cryptography, TU Darmstadt, Darmstadt, Germany.
sebastian.faust@gmail.com
* Université de Lorraine, CNRS, Inria, LORIA, Nancy, France. virginie.lallemand@loria.fr
‡ Horst Görtz Institute for IT Security, Ruhr-Universität Bochum, Bochum, Germany.
gregor.leander@rub.de
◇ Inria, Paris, France. leo.perrin@inria.fr

Abstract. In order to lower costs, the fabrication of Integrated Circuits (ICs) is increasingly delegated to offshore contract foundries, making them exposed to malicious modifications, known as hardware Trojans. Recent works have demonstrated that a strong form of Trojan-resilience can be obtained from untrusted chips by exploiting secret sharing and Multi-Party Computation (MPC), yet with significant cost overheads. In this paper, we study the possibility of building a symmetric cipher enabling similar guarantees in a more efficient manner. To reach this goal, we exploit a simple round structure mixing a modular multiplication and a multiplication with a binary matrix. Besides being motivated as a new block cipher design for Trojan resilience, our research also exposes the cryptographic properties of the modular multiplication, which is of independent interest.

Keywords: symmetric encryption · modular multiplication · Trojan-resilience.

1 Introduction

Most modern cryptographic systems rely on the fundamental assumption that the hardware on which they are implemented is trustworthy. This assumption is, however, violated when the hardware manufacturer becomes malicious, and can mount attacks at the hardware-level, including hardware Trojans or hardware counterfeiting. Perhaps most devastating are hardware Trojans [TK10, BHBN14, XFJ+16], which are stealthy malicious modification of the integrated circuits – the heart of any electronic hardware. Hardware Trojans may have catastrophic consequences for security, including deliberately weakened cryptographic devices or malfunction of control systems in nuclear power plants.

Unfortunately, the dangers resulting from malicious hardware have drastically escalated due to the inherent distributed nature of modern hardware manufacturing. Since setting up top-notch foundries that can produce integrated circuits at 14nm or beyond is very costly – estimates suggest almost US $15 billion to setup the next generation fabs [Rib14] – the majority of ICs are produced offshore by untrusted foundries. Currently only 13 foundries control more than 90% of the global IC production market, and none of them is located within the EU [Ins14]. In this situation, as little as a single malicious foundry can have major negative impact on a large fraction of modern security systems.

In view of this, various types of countermeasures have been introduced in order to mitigate the threats arising from hardware Trojans. At a high-level these countermeasures

Received: 2020-06-01 Revised: 2020-11-23 Accepted: 2021-01-23 Published: 2021-03-19

can be classified into two main types. The first type are detection-based solutions which aim at spotting the presence of a Trojan through device inspection [ABK+07, AARP10]. Unfortunately, the effectiveness of Trojan inspection significantly decreases with the growing complexity of the IC, and in most cases becomes infeasible for practically deployed devices. The second type of countermeasures are the so-called prevention-based methods which aim at making the insertion or exploitation of a hardware Trojan more difficult. Examples of preventive countermeasures include split manufacturing [IEGT13], or input scrambling [WS11], but up to now both their design and analysis have been ad-hoc and only allows for protection against weak types of Trojans.

Recently, several papers have initiated a more formal approach to designing and studying the security of Trojan countermeasures, see in particular [DFS16, MCS+17, AKM+18]. These works formally define security models, propose novel countermeasures and prove their security against broad and well-defined classes of Trojan attacks. In order to achieve Trojan resilience, their main ingredient is to rely on techniques from Multi-Party Computation (MPC), where the IC internally runs an MPC protocol thereby preventing the adversary from deliberately activating the hardware Trojan. While at a technical level these works use similar ideas (i.e., relying on MPC techniques), the concrete constructions and the level of security that they can achieve are quite different. In particular, while [MCS+17, AKM+18] aim at preventing an adversary from breaking the underlying cryptographic primitive (e.g., by extracting the secret key from the device), in [DFS16] Dziembowski et al. consider a much stronger security definition called *robustness*. The robustness property guarantees that the device under attack behaves correctly (i.e., it still executes its specification) even when run in an adversarial environment. The latter means that the adversary has full input/output control over the device under attack.

In this paper, we focus on the approach given in [DFS16] as it gives a stronger security guarantee under the reasonable condition that the number of times the device is used in real life is bounded.[1] To illustrate the strength of the model of [DFS16] let us consider a simple example, where a device is implementing an authentication protocol for controlling an aircraft. The security property of [MCS+17, AKM+18] guarantees that a potential hardware Trojan inside the device does not allow an external adversary to take over the control of the aircraft (since even in the presence of the Trojan the authentication device retains its security). However, their security notion does not protect against malfunction, i.e., a malicious manufacturer may plant a Trojan that when activated makes the authentication device always fail. Clearly, such malfunctioning can have devastating consequences for our aircraft example. On the contrary, the countermeasure proposed in [DFS16] prevents such denial-of-service attacks. Our example also illustrates that a security guarantee that holds for a bounded number of executions is meaningful in practical settings, as there is a natural bound on how often the aircraft is going to be used.

In [DFS16] the authors present a *general countermeasure* based on a passively secure 3-party computation protocol. A general purpose countermeasure allows to protect arbitrary computations implemented on a device against Trojan attacks. Dziemboski et al. achieve this by relying on general purpose MPC techniques, which unfortunately leads to several drawbacks. First, they require a so-called *testing phase* where prior to using the device, the input/output behaviour of each individual component of the system is intensively tested. For standard block ciphers, this results into rather involved testing, where the entire communication between the parties running a 3-party computation protocol is tested. Second, the increase of the overall circuit size and the computational overheads needed for the Trojan countermeasure of [DFS16] compared to an unprotected device is significant. In this work, we investigate whether these two shortcomings can be addressed. To this end, we focus on one of the most important primitives of cryptography, and design an efficient *Trojan-resilient block cipher*, which combines small computational overheads with

[1] As shown by Dziembowski et al. this is unavoidable for such a strong security guarantee.

minimal requirements on the testing phase.

1.1 Our contributions

Our main contribution is the design and the security analysis of MOE, a new block cipher with an innovative structure that allows for efficient computation on secret shares by mostly relying on linear operations. Obviously, if the cipher only consists of linear operations for the XOR, we cannot hope its construction to be secure, so our design idea is to mix linear operations over various different underlying groups. After in-depth analysis of the possible choices, we picked two groups, that are $(\mathbb{Z}/2\mathbb{Z})^n$ and $(\mathbb{Z}/2^n\mathbb{Z})$. The latter corresponds simply to a modular multiplication, an operation that was used back in the 1990s for the design of IDEA [LM91] and inspired the design of ciphers such as MESH [NRPV04], MMB [DGV93] and WIDEA-n [JM09]. We provide an extended analysis of the cryptographic properties of the modular multiplication by 3 by evaluating its algebraic degree together with its differential and linear properties. We next study how the modular multiplication behaves when associated with the binary matrix multiplication to form one round of encryption. By introducing the notion of change branch number, we are able to compute bounds on the probability of differential characteristics. We also discuss other attacks and evaluate the security of a small scaled version, and conclude that MOE seems resistant against basic attacks.

In addition to this, we show that our design decisions lead to the desired outcome in regard to the implementation of MOE as a Trojan-resilient cipher. More precisely, we consider the methodology introduced in [DFS16] and compare the performances of MOE with the one obtained in [BDFS18] for the AES and Mysterion, a bitslice-oriented cipher. The first notable advantage of our cipher is that it allows a simplified testing phase: only the input/output behaviour of the shared circuits implementing MOE must be tested (which can be done with a single plaintext/ciphertext pair) while when implementing the other ciphers all the intermediate computations must be verified (which requires a large dictionary of plaintext/ciphertext pairs).[2] This improvement represents a significant step in the direction of Trojan-resilient block ciphers that can be deployed and tested on-the-fly, as it avoids the previous situation where one either needs a way to monitor all the communications (which will make the test much more expensive in time) or use a dedicated board with all circuits trusted but one (which allows testing only the output result, but requires to plug/unplug the circuits to test on this dedicated board). The second improvement brought about by our design is a decrease in the communication complexity, resulting in an increased throughput and better robustness guarantees. The gain is significant compared to the AES, a bit less compared to ciphers that are already optimized for multiplicative complexity (for other applications like masking). This last point seems to indicate that in terms of communication complexity, the limits that one can reach are similar for block ciphers designed according to very different principles. Finally, a last drawback of [DFS16] was the need of a 3-party protocol. The structure of MOE, and more precisely the linear nature of its round operations, enables a much simpler 2-party protocol which reduces the overall hardware cost. Similar to [DFS16] our construction requires a small trusted circuitry. The only downside of our result is that this trusted circuitry increases in complexity – albeit only by a small factor. We believe it is an important direction for future work to improve on the design of MOE (or provide alternative block cipher designs) that minimize the size of the trusted master.[3]

We finally emphasize that our design goals are different than the ones used for MPC-friendly or masking-friendly ciphers, see e.g., [GLSV15, ARS+15, GRR+16, DEG+18,

[2] This is because despite the parties of [DFS16] always communicate through the master, these communications are not randomized (which would result in even more prohibitive overheads).

[3] As proven in [DFS16] some trusted circuitry is needed in order to achieve strong Trojan-resilience.

AAB+20, GLR+20]. While the latter mostly focus on minimizing the number of non-linear operations, the use of a small trusted master (responsible for the sharing operations) in our Trojan-resilient circuits implies that we rather focus on minimizing the number of transitions from one group to another. Yet, again, the (pseudo) multiplicative depth of our cipher (defined as the number of such transitions) is close to the actual multiplicative depth in these other ciphers, suggesting that both proposals are close to the minimum required. The latter observation, together with the in-depth investigation of the cryptographic properties of the modular multiplication, is of independent interest.

Structure of the paper. We first describe our new approach to Trojan-resilient block ciphers in Section 2 and compare it with the generic solution of Dziembowski et al. Based on this, we design a new block cipher named MOE that we describe in Section 3. One peculiarity of MOE is the use of modular multiplication as a block cipher ingredient, a choice that we carefully justify in Section 4 by discussing its cryptographic properties. We start by looking at the algebraic degree of this operation and then give an extensive (and to the best of our knowledge, the first) analysis of the differential and linear properties of a special case, that is the multiplication by 3. We then provide a security analysis of our cipher (Section 5), and in particular our analysis of the modular multiplication allows us to prove that no high-probability differential characteristics exist. We conclude by discussing the implementation of MOE on a prototype Printed Circuit Board (PCB) combining four commercial FPGAs and compare the obtained performances with the ones reached with a generic MPC implementation of both the standard AES and a bitslice-oriented cipher.

2 Our approach to Trojan-resilience

In this section we start by recalling the framework of Dziembowski et al. [DFS16], provide a high-level description of our construction and develop a formalization for analyzing its security. Notice that for now we view the underlying round operations of the MOE cipher as a black-box, and do not dive into the details of our concrete instantiation.

2.1 Trojan attack setup

We follow the work of Dziembowski et al. [DFS16] and consider a setup involving three parties: the trusted designer of the device, a trusted tester and a malicious manufacturer. In this setting the designer of the IC sends the IC's specification (in some hardware description language) to the manufacturer who produces the hardware and delivers it for testing and final assembly. The testing is carried out in a trusted environment by a special party called the tester, which checks whether the devices operate according to their intended specification. We only consider black-box testing, i.e., testing if the input/output behavior of the devices match with their specification. Such a setting of outsourcing the manufacturing process is very common and widely used in modern IC production due to the high financial cost for setting up state-of-the-art chip foundries.

While a vast number of different hardware Trojan attacks has been discussed in the literature, we focus on a well-defined (yet still broad) class of possible attacks that informally can be described as "logical attacks". More concretely, recall that a hardware Trojan consists of a *triggering mechanism* and a *payload*. The triggering mechanism activates the hardware Trojan, while the payload describes the malicious behavior that is carried out by the Trojan (e.g., revealing secret keys or malfunctioning of the device). At a very high-level one can distinguish between "physical" and "logical" Trojans. In the first case, the triggering and payload are carried out in some physical way. This may for instance be an activation of the Trojan by running it in an environment of higher temperature, and/or leaking the secrets via physical side-channels. On the other hand, logical Trojans assume that the Trojan triggering is carried out via logical inputs delivered to the device, and

the payload is received by the adversary via a logical output. As in [DFS16], we focus in this work on logical Trojans. Logical Trojans include many natural Trojan attacks such as cheat codes (activation is triggered by a hard-to-guess input) and time bombs (activation is triggered after a certain number of executions) [WS11, DFS16].

Let us now describe the framework of Dziembowski et al. in general terms. First, a single malicious manufacturer produces a set of mini-devices $\mathcal{D}_i^j$, where every triple $\mathcal{D}^j := (\mathcal{D}_1^j, \mathcal{D}_2^j, \mathcal{D}_3^j)$ aims to compute a 3-party protocol and is denoted as a sub-device. These devices supposedly implement the desired functionality and are delivered back to the designer for testing. In the testing the designer checks if the produced mini-devices have the same input/output behavior as the desired functionality that the devices are supposed to implement. After the testing of these sub-devices has been completed successfully, they are assembled together with a so-called *trusted master* to build the final device $\mathcal{D}$. One may think of the trusted master as a coordinator that controls the computation among the different mini-devices and carries out some simple computations. Naturally, we want that these computations are as simple as possible as otherwise the full circuit can directly be implemented using the technology used for building the trusted master. Dziembowski et al. show however that some minimum trusted computations are necessary when we aim to achieve strong Trojan-resilience guarantees (i.e., robustness).

2.2 Trojan countermeasure and cipher design

2.2.1 Multiparty computation as a Trojan countermeasure

Let us consider cheat codes as one particularly dangerous Trojan attack. In a cheat code the Trojan gets activated by running the device on a certain hard-to-guess input (the "cheat code"). Once this input is provided the Trojan delivers its payload and the device starts to malfunction. Cheat codes are dangerous because they are nearly impossible to detect by functional testing. This is due to the fact that they are hard-to-guess by design. To effectively prevent cheat codes we may however use techniques from MPC and secret sharing. Consider a setting with 3 parties denoted by P_1, P_2 and P_3, where each party holds its input x_1, x_2, respectively x_3. A secure MPC protocol for a function f allows $P_1 \dots P_3$ to securely evaluate $f(x_1, x_2, x_3)$ such that nothing is revealed about the parties' individual inputs except for what is implied by the evaluated output $f(x_1, x_2, x_3)$.

The high-level idea of using multiparty computation to protect against cheat codes is the following. Before the adversarially generated input x enters the device it is shared using a secret sharing scheme (e.g., Shamir's secret sharing) into shares x_1, x_2 and x_3. This secret sharing is done on a trusted master device, and hence is done without influence from the malicious circuit manufacturer. Each of the shares x_i is then given to a mini-device $\mathcal{D}_i^j$ that was produced by the manufacturer and supposedly implements the functionality computed by party P_i in the MPC protocol. The devices then emulate a secure function evaluation of f. From the security guarantee of the secret sharing scheme and the multiparty computation protocol, it is guaranteed that the mini-devices do not learn anything about the shared input x. Thus, this approach takes away the adversary's control over the inputs x, thereby preventing targeted activation by a cheat code.

The above description gives the high-level idea of the basic construction outlined in [DFS16]. Dziembowski et al. observe however that the MPC approach alone does not suffice to protect against more powerful Trojan attacks. For instance, multiparty computation by itself cannot protect against time-bombs (i.e., Trojans that get activated after a certain number of executions). This is the case because the Trojan activation is now not triggered via an input but just once a certain threshold of activation is reached. Even worse, standard MPC does not protect against malicious devices undoing the effect of secret sharing by just reconstructing the shares via subliminal channels. This motivates the approach of [DFS16] to combine MPC with testing, where the individual devices are

tested for correct computation, thereby preventing subliminal channels and time-bombing attacks. Let us continue by providing more details on the approach of [DFS16].

Consider a stateful arithmetic circuit Γ describing the desired functionality that we want to outsource to a (possibly) malicious manufacturer $\mathcal{A}$ for production. One may think of Γ as a specification of an AES block cipher, where the state corresponds to the secret key of the AES. To protect against Trojans, instead of letting $\mathcal{A}$ produce a device $\mathcal{D}$ that implements Γ, we first transform Γ into a new algorithm Γ' that is hardened against Trojan attacks. At a high-level, Γ' consists of two components. First, a set of mini-circuits Γ_i^j, and second a specification of the master circuit $\mathcal{M}$. As discussed above, the role of the master circuit is to manage the communication between the mini-circuits and carry out some simple trusted computation (e.g., the secret sharing of the inputs). Every triple of mini-circuits $(\Gamma_1^j, \Gamma_2^j, \Gamma_3^j)$, with $1 \leq j \leq \lambda$ and λ the number of sub-circuits, performs a 3-party computation of the target functionality. In the computation the trusted $\mathcal{M}$ takes the role of carrying out the communication between the parties, and additionally runs some simple trusted pre- and post-processing (essentially secret sharing the inputs and reconstructing the outputs). Looking again at the AES example from above, each of the λ sub-circuits would implement a 3-party computation protocol securely evaluating the AES on shared inputs corresponding to the adversarially chosen plaintext. The set of circuit specifications $\{\Gamma_1^j, \Gamma_2^j, \Gamma_3^j\}_{j\in[\lambda]}$ is then given to the malicious manufacturer $\mathcal{A}$ who produces devices $\mathcal{D}_i^j$ that implement the corresponding mini-circuits Γ_i^j. For completeness, Appendix A recalls the 3-party protocol used in [DFS16].

When the designer receives the devices back from the manufacturer, he first carries out some functional testing of each $\mathcal{D}_i^j$. This is done by a tester $\mathsf{T}_{\mathsf{DSF}}$ which runs each $\mathcal{D}_i^j$ and interacts with it through its interface by providing as inputs some values that $\mathcal{D}_i^j$ would expect when it is run in the real environment. If the input/output behavior of each $\mathcal{D}_i^j$ matches with the corresponding specification of Γ_i^j, then the designer assembles the final device $\mathcal{D}$ by combining the mini-circuits $\mathcal{D}_i^j$ with the trusted master $\mathcal{M}$. Dziembowski et al. then show that the input/output behavior of this final assembled device $\mathcal{D}$ is with overwhelming probability identical to the input/output behavior of the trusted specification Γ for a limited number of executions, even for adversarially chosen inputs. This security notion on which we focus is called *Trojan robustness* in [DFS16].

2.2.2 High-level idea of our efficient Trojan resilient block cipher

Our block cipher design follows a very similar approach as the one described above. The main difference however is that due to our novel design of the round function we do not need to apply general purpose protocols for secure 3-party computation. Indeed, since all the operations inside a round of the block cipher are linear and the trusted master carries out the secret sharing and the reconstruction, we can evaluate the round operations directly on secret shares without any interactions between the components. This is possible because for linear secret sharing schemes, linear operations are easy to compute on shared data. Let us now take a closer look at our high-level design shown in Figure 1.

Let MOE denote the cipher that we will detail in the later sections. For now it will only be important that MOE requires operations which are linear over some group. To this end, we have chosen two distinct groups in which this linearity is described. After a study of the different possibilities which we provide in Appendix B, we settled for the following two types of round operations: the multiplication by a matrix of $GL_n(\mathbb{F}_2)$, and the multiplications by an odd integer α modulo 2^n.[4] To simplify notation we denote these two operations by ⧫ and ●, and will write $* \in \{⧫, ●\}$ as a placeholder for one of them. Moreover, we let share_* denote the secret sharing operation over group operation $*$, and

[4]The decision to apply these operations on the full state (instead of using a construction that could benefit from the wide trail strategy) is explained in Section 4.5.

let $\mathsf{reconstruct}_*$ denote the corresponding reconstruction operation. The specification of

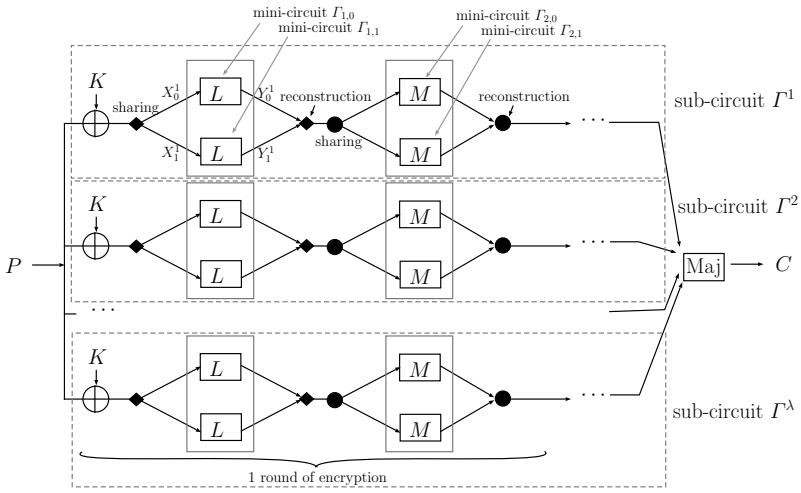


Figure 1: High-level view of the Trojan-resilient specification of our block cipher. The devices L and M are produced by the untrusted manufacturer and are assembled together by a trusted master that takes care of the required sharings and reconstructions. The ciphertext is obtained by computing the majority over the output of the λ sub-circuits.

MOE consists of λ sub-circuits, where each sub-circuit Γ^j has exactly the same structure (hence we will sometimes abuse notation and omit the parameter j). Each sub-circuit Γ consists of 2κ mini-circuits $((\Gamma_{1,0}, \Gamma_{1,1}), \ldots, (\Gamma_{\kappa,0}, \Gamma_{\kappa,1}))$, where each pair of mini-circuits represents a field operation of MOE computed in the shared-domain. Note that these 2κ mini-circuits can be implemented with only 2 physical devices. More precisely, denote by X^i the input to the i-th operation of MOE and with Y^i the corresponding output. Let (X_0^i, X_1^i) be a secret sharing according to the underlying group used by this operation, i.e., $(X_0^i, X_1^i) \leftarrow \mathsf{share}_*(X^i)$. We have $Y_b^i \leftarrow \Gamma_{i,b}(X_b^i)$ with $b \in \{0,1\}$. Notice that due to the linearity of the operation, it holds that $Y^i = \mathsf{reconstruct}_*(Y_0^i, Y_1^i)$. This completes the description of the computation carried out by the (untrusted) mini-circuits $\{(\Gamma_{i,0}, \Gamma_{i,1})\}_i$. The specification of these mini-circuits will be outsourced for production to the malicious manufacturer $\mathcal{A}$, who will return a set of devices $\{(\mathcal{D}_{i,0}, \mathcal{D}_{i,1})\}_i$. The latter will be used together with the trusted master during the assembly of the final device $\mathcal{D}$.

As part of each of the λ sub-devices $\mathcal{D}^j$ (this corresponds essentially to the computation of Γ^j) the master $\mathcal{M}$ will carry out share_* and $\mathsf{reconstruct}_*$ for each of the 2κ operations. Since share_* is a randomized algorithm this means that the master $\mathcal{M}$ has access to a trusted source of randomness. This is a strong assumption, but as explained in [DFS16] this randomness can be generated by the untrusted devices thanks to an efficient Trojan-resilient PRG. Moreover, as in [DFS16] the master computes a final majority of the output produced by the λ sub-devices. Majority is here defined by computing the majority bitwise which, as discussed in [BDFS18] is necessary to keep the cost of the master low, and is possible because after the testing phase the output of all the sub-circuits is equal to the output of the trusted specification. We finally need to handle the key addition. The simplest option for this purpose is to handle it in the trusted master. In this case a small trusted memory is required. To minimize the complexity of the master, we may, however, also carry out the key addition by distributing key shares among the untrusted mini-devices, and let them carry out the key addition. In this case a secure provisioning phase must distribute the shares of the key to the mini-devices during deployment. The master circuitry $\mathcal{M}$ implemented on trusted hardware controlling the mini-devices $\{(\mathcal{D}_{i,0}, \mathcal{D}_{i,1})\}_i$ which were

produced by the malicious manufacturer $\mathcal{A}$, form the final device $\mathcal{D}$. In the following, we will sometimes write $C \leftarrow \mathcal{D}(K, P)$ for running the final device $\mathcal{D}$ with input key K and plaintext P producing ciphertext C. Similarly we will use the notation $C \leftarrow \mathcal{D}^j(K, P)$ for the execution of the j-th sub-device of $\mathcal{D}$. Notice that the latter does not involve the computation of the majority that is part of the trusted master circuitry.

2.2.3 Comparison with the generic proposal of Dziembowski et al.

While the construction of Dziembowski et al. has the important advantage of offering a general purpose countermeasure (i.e., a compiler that can protect any computation against Trojan attacks), our construction has the following main advantages making it particular appealing for real-world implementations.

- *Simplified testing phase:* One fundamental limitation of the generic solution of Dziembowski et al. is that the testing phase must verify the correctness of all the intermediate results of the computation (including their communication). The latter is critical to prevent subliminal channels between the mini-devices. For instance, in [DFS16] if we do not test the necessary communication between the parties running the 3-party computation, then the mini-devices may just exchange their secret shares that they have received from the master. This would enable an adversary to easily exploit cheat codes. The main advantage of our construction is that the testing phase is significantly simpler than the one used in [DFS16]. The main reason for this is that we only need to test correctness of the input/output behavior of the *entire* block cipher instead of testing all individual components produced by the malicious manufacturer. Since the sub-circuits are shared, this further means that the test can be done by storing a single correct plaintext/ciphertext pair.[5]
- *Reduced communication complexity:* Our construction provides Trojan-resilience with a reduced number of communication rounds between the trusted master and the mini-circuits, which is the main factor limiting the implementation throughput.
- *Reduced hardware cost:* Our construction can provide Trojan-resilience with two untrusted mini-circuits per sub-circuit while the construction in [DFS16] requires three mini-circuits per sub-circuit as it relies on general MPC techniques.

2.3 Protecting against Trojans: robustness vs. security

As discussed in the introduction, in this work we aim to achieve the stronger security guarantee of robustness introduced in [DFS16]. Informally, robustness guarantees that after the testing phase is completed successfully, with high probability the final assembled device $\mathcal{D}$ operates according to its specification. This is formalized via a robustness game in Figure 2 that we tailored to the case of a block cipher.

The robustness game considers a setting where the untrusted mini-devices $\{(\mathcal{D}_{i,0}, \mathcal{D}_{i,1})\}_i$ are produced by a malicious manufacturer $\mathcal{A}$. The manufacturer is allowed to arbitrarily change the logical description of these mini-devices. This means that they can implement an arbitrary circuitry $(\tilde{\Gamma}_{i,0}, \tilde{\Gamma}_{i,1})$, which deviates from the intended specification $(\Gamma_{i,0}, \Gamma_{i,1})$. Similar to [DFS16] we only consider Trojans that are triggered via logical inputs and payloads that are delivered through the logical outputs. Let us now take a closer look at the robustness game $\mathsf{ROB}_{\mathsf{MOE}}(\mathsf{T}, \mathcal{A}, \eta, t, \lambda, K)$.

Let $\mathcal{A}$ denote a malicious manufacturer, and T be a trusted testing algorithm that will be defined in more detail below. The purpose of T is to test whether the potential

[5] Technically, this is made possible because a reconstruction/resharing operation is preformed by the trusted master each time we switch from one group to another, which prevents mini-circuits to exchange information. One could possibly apply this strategy to the generic compiler of Faust et al. but it would cause prohibitive overheads to an already expensive solution.

Game $\mathsf{ROB}_{\mathsf{MOE}}(\mathsf{T}, \mathcal{A}, \eta, t, \lambda, K)$:
Let $\{(\mathcal{D}^j_{i,0}, \mathcal{D}^j_{i,1})\}_{i,j} \leftarrow \mathcal{A}(1^k, \mathsf{MOE})$;
If $\mathsf{T}^{\mathcal{D}^1(K,.),\ldots,\mathcal{D}^\lambda(K,.)}(1^k, \lambda, t, K) = \mathsf{false}$, then return 0;
Let $P_1 \leftarrow \mathcal{A}(1^k)$;
For $i = 1$ to η repeat:
  Compute $C_i \leftarrow \mathcal{D}(K, P_i)$;
  Evaluate $C'_i \leftarrow \mathsf{MOE}(K, P_i)$;
  If $C_i \neq C'_i$ then return 1;
  Let $P_{i+1} \leftarrow \mathcal{A}(1^k, C_i)$;
Return 0.

Figure 2: The formal robustness game $\mathsf{ROB}_{\mathsf{MOE}}$.

malicious devices $\{(\mathcal{D}_{i,0}, \mathcal{D}_{i,1})\}_i$ satisfy their corresponding specification. Further, let MOE denote a trusted reference implementation of our cipher and K be a secret key. In addition, the robustness definition is parameterized by the following two parameters: η denotes the number of executions for which we want the final assembled implementation to work correctly after testing, and t is the maximal number of tests that we carry out during the testing phase. In [DFS16] it was proven that we can guarantee the implementation $\mathcal{D}$ to work correctly for η executions, and that we require $\eta \ll t$.

At a high-level the robustness game as shown in Figure 2 proceeds in three phases. In the first phase, the adversary produces the untrusted mini-devices $\{(\mathcal{D}^j_{i,0}, \mathcal{D}^j_{i,1})\}_{i,j}$ which are then assembled together with the trusted master to build the λ implementations $\mathcal{D}^1, \ldots, \mathcal{D}^\lambda$. Notice that each $\mathcal{D}^j$ is built from a set of mini-devices $\{(\mathcal{D}_{i,0}, \mathcal{D}_{i,1})\}_i$ and uses parts of the trusted master $\mathcal{M}$ for secret sharing and reconstruction of the intermediate values. In the testing phase, the tester algorithm T checks the correctness of each implementation $\mathcal{D}^j$ with respect to a reference implementation MOE. The tester T has oracle access to the partial assembled devices $\mathcal{D}^j$ initialized with key K and will be discussed in more details below. In the last phase, the final device $\mathcal{D}$ (see Figure 1, where we essentially add the majority computation as part of the trusted master) is executed η times. In each such run, the adversary $\mathcal{A}$ is allowed to provide inputs for $\mathcal{D}$ in order to, e.g., trigger the hardware Trojan. We say that the adversary $\mathcal{A}$ wins the game $\mathsf{ROB}_{\mathsf{MOE}}$ if the game outputs 1. This happens if $\mathcal{A}$ manages to activate the hardware Trojan during one of the η executions despite the testing phase being completed successfully. The hardware Trojan is considered activated if the device $\mathcal{D}^{\mathsf{MOE}}$ outputs a value that differs from the value that would have been produced by the trusted reference implementation of MOE. Otherwise the game outputs 0 indicating that the adversary lost the game.

2.3.1 Simplified testing via T

One key feature of our Trojan-resilient cipher is that it allows for a simplified testing, where only the cipher's final input/output pairs must be tested.[6] This is in contrast with [DFS16] where all the communication between the mini-devices produced by $\mathcal{A}$ need to be tested for correctness. Our construction achieves this beneficial property due to the fact that there is no direct communication between the mini-devices $\mathcal{D}_{i,0}$ and $\mathcal{D}_{i,1}$. This can be obtained because the round operations are linear over some group and hence they can compute on secret shared values without communication between $\mathcal{D}_{i,0}$ and $\mathcal{D}_{i,1}$. In addition, communication between $(\mathcal{D}_{i,0}, \mathcal{D}_{i,1})$ and $(\mathcal{D}_{i+1,0}, \mathcal{D}_{i+1,1})$ has to go via a fresh secret sharing (i.e., via the trusted master), and hence $(\mathcal{D}_{i,0}, \mathcal{D}_{i,1})$ cannot directly communicate with

[6] As already mentioned, since the inputs to the cipher are always freshly secret shared, it even means that a single plaintext/ciphertext pair is sufficient to test this part of the circuit.

Tester $\mathsf{T}^{\mathcal{D}^1(K,.),\ldots,\mathcal{D}^\lambda(K,.)}(1^k,\lambda,t,K)$:
Let P^* be some plaintext in the plaintext space and $C^* = \mathsf{MOE}(K,P^*)$.
For $j \in [\lambda]$ repeat the following:
  Sample $t_j \leftarrow [t]$ uniformly at random and repeat t_j times:
    Query P^* to the oracle $\mathcal{D}^j(K,.)$ and denote by C the returned value.
    If $C \neq C^*$ return false.
Return true.

Figure 3: The tester T of the MOE construction.

Tester $\mathsf{T}_{\mathsf{DSF}}^{\mathcal{D}^1(.),\ldots,\mathcal{D}^\lambda(.)}(1^k,\lambda,t)$:
Set the initial state of the devices $\{\mathcal{D}^i\}_i$ to $\vec{0}$ % E.g., sets the key of the devices
For $j \in [\lambda]$ repeat the following:
  Sample $t_j \leftarrow [t]$ uniformly at random and repeat t_j times:
    Sample random 2-out-of-2-sharing $\vec{r},\vec{s}$ representing the shared input
    If $\mathsf{View}(\mathcal{D}^j(\vec{r},\vec{s})) \neq \mathsf{View}(\Gamma^j(\vec{r},\vec{s}))$ return false.
Return true.

Figure 4: The tester $\mathsf{T}_{\mathsf{DSF}}$ from [DFS16] slightly adjusted to our notation. $\mathsf{View}(\mathcal{D}^j(\vec{r},\vec{s}))$ and $\mathsf{View}(\Gamma^j(\vec{r},\vec{s}))$ denote the random variable of the input/output behavior of $\mathcal{D}^j$, respectively Γ^j when run on inputs $\vec{r},\vec{s}$ and mini-circuits communicate throught the master.

$(\mathcal{D}_{i+1,0},\mathcal{D}_{i+1,1})$ as well. We formalize our testing procedure via the tester T presented in Figure 3, where we focus on the case when the key is stored on the trusted master. It is easy to extend the testing algorithm to the case when the key is stored on the untrusted devices. For comparison, the tester $\mathsf{T}_{\mathsf{DSF}}$ from [DFS16] is given in Figure 4. The main differences between these two testing algorithms are in the inner part of the for loop. While our simplified tester T only needs to compare whether the plaintext/ciphertext pair matches with input/output produced by an honest evaluation of MOE cipher (i.e., via comparison with the pair (P^*,C^*)), the original tester $\mathsf{T}_{\mathsf{DSF}}$ from [DFS16] is significantly more complicated. In particular, in $\mathsf{T}_{\mathsf{DSF}}$ we require that the entire input/output *communication* of the devices $\mathcal{D}^j$ matches with the corresponding ideal specification Γ^j. To this end, in Figure 4, $\mathsf{T}_{\mathsf{DSF}}$ uses the check $\mathsf{View}(\mathcal{D}^j(\vec{r},\vec{s})) \neq \mathsf{View}(\Gamma^j(\vec{r},\vec{s}))$.

2.3.2 Trojan robustness of MOE

We are now ready to prove the Trojan robustness of our construction MOE, which is summarized in the theorem below.

Theorem 1. *Let $t,\eta,\lambda,k \in \mathbb{N}_{>0}$ with $\eta < t$ be natural numbers. For any malicious manufacturer $\mathcal{A}$, $K \leftarrow \{0,1\}^k$ chosen uniformly at random we have:*

$$\Pr\left[\mathsf{ROB}_{\mathsf{MOE}}(\mathsf{T},\mathcal{A},\eta,t,\lambda,K)=1\right] \leq \left(\frac{4\eta}{t}\right)^{\lceil\lambda/2\rceil},$$

where the randomness is taken over the randomness of the $\mathsf{ROB}_{\mathsf{MOE}}$ game.

Proof. We follow the proof approach of [DFS16]. To this end we first argue that the inputs that the mini-devices $\{\mathcal{D}_{i,b}^j\}_{i\in[\kappa],j\in[\lambda],b\in\{0,1\}}$ receive in the testing phase are identical to what they receive when run as part of the fully assembled device $\mathcal{D}$. More precisely, we denote by $\mathsf{view}_{i,b}^j(K,P)$ the random variable that represents the inputs taken by the mini-device $\mathcal{D}_{i,b}^j$ when $\mathcal{D}$ is run with input plaintext P and secret key K. The randomness

of $\mathsf{view}^j_{i,b}(K,P)$ is taken over the randomness of the secret sharing done by the trusted master. The next lemma follows from the fact that the mini-devices only receives inputs that went through a secret sharing gate using uniform randomness for the sharing function, with $\equiv$ denoting equivalence of the distributions.

Lemma 1. *For any $i \in [\kappa], j \in [\lambda], b \in \{0,1\}$, for any plaintext inputs P, P' and any secret key K, we have $\mathsf{view}^j_{i,b}(K,P) \equiv \mathsf{view}^j_{i,b}(K,P')$.*

This lemma implies that the view of each mini-device is *independent* of the plaintext P. Hence, the distribution of inputs that $\mathcal{D}^j_{i,b}$ receives in the testing phase is indistinguishable from the view that it receives in the online phase. Given the above the proof of the theorem follows the proof of Theorem 1 in [DFS16] using a series of hybrid games.

1. **Game $\mathsf{ROB}^1_{\mathsf{MOE}}(\mathsf{T},\mathcal{A},\eta,t,\lambda,K)$:** This is the same as the robustness game defined in Figure 2 except for the difference that we replace the mini-devices $\mathcal{D}^j_{i,b}$ by some abstract circuit specification $\tilde{\Gamma}^j_{i,b}$ with the same input/output behavior as $\mathcal{D}^j_{i,b}$. Hence, we directly get that the probability that $\mathcal{A}$ wins in this game is identical to the probability that $\mathcal{A}$ wins in $\mathsf{ROB}_{\mathsf{MOE}}(\mathsf{T},\mathcal{A},\eta,t,\lambda,K)$.

2. **Game $\mathsf{ROB}^2_{\mathsf{MOE}}(\mathsf{T},\mathcal{A},\eta,t,\lambda,K)$:** Denote by $\tilde{\Gamma}^j$ the specification of the j-th component consisting of $(\tilde{\Gamma}^j_{1,0},\tilde{\Gamma}^j_{1,1}),\ldots,(\tilde{\Gamma}^j_{\kappa,0},\tilde{\Gamma}^j_{\kappa,1})$ and the corresponding parts of the master circuits $\mathcal{M}$ that controls these mini-circuits. ROB^2 differs from ROB^1 as follows. While in ROB^1 the game outputs 1 when for some of the η iterations $C_i \neq C'_i$, in ROB^2 we output 1 when for more than $\lambda/2$ of the λ components $\tilde{\Gamma}^j$ the ciphertext output by this component differs from the execution of the correct specification Γ^j. Since the output is part of the views, we get that:

$$\Pr[\mathsf{ROB}^2_{\mathsf{MOE}}(\mathsf{T},\mathcal{A},\eta,t,\lambda,K)=1] \geq \Pr[\mathsf{ROB}^1_{\mathsf{MOE}}(\mathsf{T},\mathcal{A},\eta,t,\lambda,K)=1].$$

3. **Game $\mathsf{ROB}^3_{\mathsf{MOE}}(\mathsf{T},\mathcal{A},\eta,t,\lambda,K)$:** In this game we replace the malicious input plaintext P provided by $\mathcal{A}$ with some fixed plaintex P'. By Lemma 1 we get:

$$\Pr[\mathsf{ROB}^3_{\mathsf{MOE}}(\mathsf{T},\mathcal{A},\eta,t,\lambda,K)=1] = \Pr[\mathsf{ROB}^2_{\mathsf{MOE}}(\mathsf{T},\mathcal{A},\eta,t,\lambda,K)=1].$$

Using Lemma 4 in [DFS16] we can show that for $\eta < t$:

$$\Pr[\mathsf{ROB}^3_{\mathsf{MOE}}(\mathsf{T},\mathcal{A},\eta,t,\lambda,K)=1] \leq \left(\frac{4\eta}{t}\right)^{\lceil\lambda/2\rceil},$$

which concludes the proof of the theorem. □

2.3.3 Eliminating the randomness from the master $\mathcal{M}$

The need for trusted randomness in the master may sound contradictory with the requirement of only using a very limited number of trusted gates. Trojan-secure randomness can however be obtained via a Trojan secure PRG as discussed in [DFS16]. To this end, a Trojan secure PRG can be constructed by XORing the output of multiple untrusted PRGs. As long as one of the PRGs works correctly, the result of the XORing is guaranteed to be pseudorandom, and hence is sufficient to be used as randomness for internal protocol computations, i.e., for the sharing of intermediate values before entering the mini-devices. Concretely, for our use case we just XOR the result of $\lceil\lambda/2\rceil$ cryptographic PRGs which achieves a security bound of $\left(\frac{\eta}{t}\right)^{\lceil\lambda/2\rceil}$, similar to the one we target for robustness. Since these PRGs do not need to have randomized inputs, their testing is also simple as we only need to verify their final outputs rather than their intermediate computations.

The main impact of replacing trusted randomness for sharing by randomness generated by a Trojan-resilient PRG is that we move to the computational setting. This means that in Theorem 1 we only consider PPT adversaries and obtain an additional loss in the robustness bound compared to Theorem 1 of negl(k), where k is the security parameter. More precisely, we obtain the following corollary:

Corollary 1. *Let $t, \eta, \lambda, k \in \mathbb{N}_{>0}$ with $\eta < t$ be natural numbers. For any malicious PPT manufacturer $\mathcal{A}$, $K \leftarrow \{0,1\}^k$ chosen uniformly at random we have:*

$$\Pr\left[\mathsf{ROB}_{\mathsf{MOE}}(\mathsf{T}, \mathcal{A}, \eta, t, \lambda, K) = 1\right] \leq \left(\frac{4\eta}{t}\right)^{\lceil \lambda/2 \rceil} + \left(\frac{\eta}{t}\right)^{\lceil \lambda/2 \rceil} + \mathit{negl}(k),$$

where the randomness is taken over the randomness of the $\mathsf{ROB}_{\mathsf{MOE}}$ game.

3 Specification of MOE

3.1 Description of MOE

In this section, we introduce a concrete instance of the approach introduced in Section 2 and propose a new block cipher called MOE. Its name stands for Multiplication Operated Encryption to reflect that all round operations are based on multiplication, namely a multiplication by a binary matrix and a modular multiplication by a constant. In order to mimic the API of the AES, it encrypts blocks of 128 bits using a 128-bit key. The encryption routine consists in iterating 4 times a `step` function made of 6 operations:

$$\mathtt{step}_i = M \circ A_3 \circ \mathsf{K}_{2i+1} \circ M \circ A_3^{-1} \circ \mathsf{K}_{2i} \qquad i \in \{0,1,2,3\}.$$

The different operations, all working on the full 128-bit state, are as described below. A graphical representation of one `step` can be found in Figure 5.

$\mathsf{K}_\mathbf{j}$ is the key addition, i.e., the function $\mathsf{K}_j : x \mapsto x \oplus K \oplus c_j$, where K is the 128-bit master key and the $\{c_j\}_{0 \leq j < 9}$ are 128-bit round constants.

$\mathbf{A_3}$ interprets the 128-bit internal state as the binary representation of an element of $\mathbb{Z}/2^n\mathbb{Z}$ and multiplies it by 3 modulo 2^n.

M interprets the 128-bit internal state as an element of $\mathbb{F}_2^n$ and multiplies it with the 128×128 binary matrix M.

$\mathbf{A_3^{-1}}$ is the compositional inverse of A_3 (the multiplication by the inverse of 3 modulo 2^n).

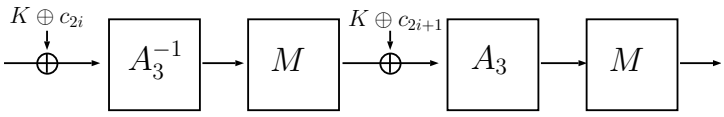


Figure 5: The `step` function number i.

Note that unlike in many ciphers such as the AES, the last call to M is *not* omitted. Another key addition (K_8) finishes the evaluation of the cipher.

To fully specify our proposal, we need to describe the round constants c_j together with the binary matrix M. Since as we will prove later a random matrix M has the necessary properties, we do not impose any specific structure on M beyond its invertibility and we simply generate a random matrix and check if it has full rank. To do so, we suggest to follow the approach proposed by the designers of LowMC [ARS+15] which

uses a self-shrinking generator [MS95] based on the LFSR of the GRAIN cipher [HJMM08]. The bits produced are used to fill the matrix row by row. We check the invertibility of the matrix once it is fully specified. If it is not, we repeat the process using the next keystream bits until it is invertible. Once the matrix is produced, we use the next bits of the keystream to form the nine 128-bit round constants $c_0 \cdots c_8$.

3.2 Security claim

We claim 127 bits of security as long as the amount of plaintext/ciphertext pairs the attacker has is smaller than 2^{64}. The claimed security level of MOE is thus reminiscent of the one for FX constructions like PRINCE [BCG+12] or QARMA [Ava17]. For such ciphers, a generic attack with complexity 2^{k-d} exists when 2^d plaintext/ciphertext pairs are available, so that attacks are expected to become practical as d gets closer to $n/2$. Our claim is stronger in that we claim 127 bits of security for all $d < 64$ rather than $127 - d$. The reason for bounding the data complexity is two-fold. First, the Trojan resilience can only be ensured for a limited number of encryptions, namely the number of tests that have been performed. In this context, it does not make sense to provide full codebook security. Second, limiting the data complexity means we can use fewer rounds to prevent attacks faster than brute-force. It thus improves the performance of our algorithm. Besides, MOE is intended to run on devices with a low throughput (see for example Table 5 in Section 6). Thus, in practice, we do not expect manufacturers to actually enforce this data limitation "manually", it will instead be a side-effect of the speed of the intended platforms.

3.3 Comparison with recent design strategies

Especially in the last few years, we have witnessed an increasing number of innovative designs focusing on efficiency in MPC and FHE settings as well as proof-friendly design strategies to be deployed, e.g., in smart contracts. Those designs on the one hand try to minimize the number of AND operations, and, more deviating from traditional symmetric primitives, are often defined over vector spaces over fields of odd characteristic and in particular over prime fields for odd primes.

Minimizing the number of AND-gates (which can be the number of AND operations per encrypted bit or the AND-depth) is the aim of designs like LowMC [ARS+15] and RASTA [DEG+18]. While RASTA achieves this by making large part of the design nonce-dependent, LowMC uses partial non-linear layers (used previously in Zorro [GGNS13] for efficient masking), and fixed but randomly generated binary linear layers.

Working over larger fields has been brought forward by MiMC [AGR+16], a cipher that uses a block-wide (or half-block wide in the case of the Feistel-variant) S-boxes defined as the cube mapping over the binary extension field. Actually, even though the motivation was very different, MiMC can be seen as an iterated version of the KN-cipher [NK95] which to our knowledge is the first cipher to use a large finite field for its design. MiMC was further generalized to GMiMC, deploying a generalized Feistel structure with the cube mapping as the round function [AGP+19]. Such ciphers all have in common the fact that their round function has a simple univariate representation over the relevant finite field.

The idea of working over non-binary fields was used, e.g., in the cipher Rescue [AAB+20]. Here, S-boxes are defined as power mappings over prime fields and the linear layer deploys MDS matrices over vector spaces over the prime field.

Combining the idea of partial non-linear layers with making use of larger and potentially non-binary fields is the core of the Hades design strategy [GLR+20] (and of its Hash instantiation Poseidon [GKK+19]). Here, the idea is to use some full S-box layer in the first and last rounds to ensure good statistical properties together with a number of partial S-box layers in the middle rounds to ensure diffusion and good algebraic properties.

Recall that the KN-cipher, that was designed to be provable secure against several statistical attacks, was broken with the invention of the interpolation attack [JK97] making use of its simple algebraic structure. This is a fate shared by some of its successors.

It turns out that it is non-trivial to analyse the new designs, in particular the ones defined over non-binary prime fields . Interestingly, some of those ciphers could be broken with algebraic attacks (e.g., Jarvis, Friday [ACG+19]) due to the simplicity of their algebraic structure. For others, it turns out that more care has to be taken with respect to the components, e.g., the MDS matrices used in Hades designs have to fulfill additional properties as described in [BCD+20, KR20].

Our approach has some resemblance to the one used for LowMC in that MOE also relies on a large binary matrix generated randomly. However, unlike LowMC, the non-linear transformations we use are dense and have a very high algebraic degree, meaning that algebraic attacks over $\mathbb{F}_2$ are not a threat. On the other hand, MOE differs significantly from algorithms like MiMC or Rescue: in those ciphers, all operations are defined in the same finite field in such a way as to allow the round function to have a simple univariate representation. It is not the case in MOE as the two main operations are defined over different mathematical structures ($\mathbb{F}_2^n$ and $\mathbb{Z}/2^n\mathbb{Z}$).

4 Justification of our design decisions: cryptographic properties of modular multiplication and general structure

The aim of this section is to evaluate the cryptographic properties of the multiplication by a constant modulo 2^n (in particular when the multiplier is equal to 3) in order to prove that its use in our proposal leads to a cipher that resists basic attacks.

We introduce the notation A_α given as follows:

Definition 1. Let α be some element of the modular ring $\mathbb{Z}/2^n\mathbb{Z}$. We denote A_α the following function:

$$\begin{aligned} A_\alpha : \mathbb{Z}/2^n\mathbb{Z} &\to \mathbb{Z}/2^n\mathbb{Z} \\ x &\mapsto \alpha \times x \end{aligned}$$

Note that A_α is a permutation if and only if α is odd.

We start by surveying the use of modular multiplication in previous designs (Section 4.1). We next discuss several cryptographic properties of A_α, namely the algebraic degree for any odd α (Section 4.2), and, for $\alpha = 3$, the differential and linear properties (Section 4.3 and Section 4.4). Beyond their significance for our design MOE, these analyses are also of independent interest. The only work formally treating of the cryptographic properties of the multiplication modulo 2^n we are aware of is the study of the so-called *S-functions* by Mouha et al. in [MVDP11] which focuses only on their differential properties.

4.1 A Brief History of Modular Multiplication in Cryptography

The use of modular multiplication in symmetric cryptography is not new: already in the 90s, it was used to build symmetric algorithms. Most prominently, the 64-bit block cipher IDEA [LM91] uses multiplications modulo $2^{16} + 1$ to mix subkeys with the internal state. The other operations it uses are the bitwise XOR and the addition modulo 2^{16}. Its authors used operations belonging to different algebraic groups in order for them to be *incompatible*—that is, no pair of operations satisfies a distributive or associative law. This design principle was meant to increase the cryptographic strength. To the authors' credit, no attack against IDEA significantly improving upon brute-force has been found in

the single key model. IDEA inspired various other constructions, namely MESH, MMB, WIDEA-n and MARS [NRPV04, DGV93, JM09, BCD+98]. Modular multiplication also appeared in other (now broken) ciphers, that used it to mix the key with the internal state: Nimbus, MultiSwap and xmx [Mac00, Scr01, MNSV97]). It was also employed in the SHA-3 candidate Shabal [BCCM+08]. An overview of some characteristics of these different algorithms is given in Table 1.

Table 1: Some cryptographic primitives using modular multiplication. n is the block size and k the key size (in bits). MultiSwap also uses a 64-bit initialization vector.

Name	Ref.	n	k	Operand	Modulus	Vulnerabilities
IDEA	[LM91]	64	128	key	$2^{16}+1$	weak keys [DGV94, BNPV02]
MARS	[BCD+98]	128	128–1248	key	2^{32}	∅
MESH	[NRPV04]	64/96/128	128/192	key/state	$2^{16}+1$	∅
MMB	[DGV93]	128	128	constants	$2^{32}-1$	differential attack [WNS09]
WIDEA-m	[JM09]	$64 \times m$	$128 \times m$	key	$2^{16}+1$	diff./lin. under weak keys, free-start collisions [Nak12, MRTV13]
Nimbus	[Mac00]	64	128	key	2^{64}	(xor) differential [Fur02]
MultiSwap	[Scr01]	64	374	key	2^{32}	multiplicative diff. [BCJW02]
xmx	[MNSV97]	flexible	flexible	key	$2\ell+1$	multiplicative diff., complementation property, weak keys [BCJW02]
Shabal	[BCCM+08]	1408	–	constants	2^{32}	∅

Note that the *multiplicative differential* is a variant of differential attack introduced in [BCJW02]. It studies the propagation through the cipher of pairs of the form $(x, \alpha x)$. It allowed attacks against Nimbus, MultiSwap and xmx.

We can learn some lessons from the previous uses of modular multiplication. On one hand, using this operation to mix the key with the internal state may lead to weak keys as in IDEA, WIDEA-n, and xmx. On the other hand, the multiplication with a constant provides interesting cryptographic properties, such as an algebraic degree and non-linearity increase, for a cost which may be as low as a single processor instruction.

4.2 Sum representation and algebraic degree

The computation of $A_\alpha(x)$ can be represented using the binary representations of its operands as the following sum modulo 2^n.

$$\begin{array}{cccccc|c} & x_{n-1} & \dots & x_2 & x_1 & x_0 & \times\alpha_0 \\ \boxplus & x_{n-2} & \dots & x_1 & x_0 & 0 & \times\alpha_1 \\ \boxplus & & \dots & & & & \cdots \\ \boxplus & x_0 & \dots & 0 & 0 & 0 & \times\alpha_{n-1} \end{array}$$

We call it the *sum representation*. This tool is the core of our proof of the next theorem.

Theorem 2. *The maximum degree that A_α can take is equal to $n-1$ and it is reached if α is congruent to 3 modulo 4.*

Proof. By referring to the sum-representation, we can easily see that bit number i of $A_\alpha(x)$ depends linearly on bits x_i and non-linearly (in a broad sense) on x_0 to x_{i-1} so its maximum degree is equal to i. Hence, the most significant bit is at most of degree $n-1$.

We now consider the modular multiplication by an integer α of the form $\alpha = 3+4k$ with $k \in \mathbb{Z}$. We start by treating the case $k=0$ and then move to the general case.

We introduce the function $\mu_i(x)$ that corresponds to the value of the carry of bit number i. Namely, the function $\mu : \mathbb{F}_2^n \to \mathbb{F}_2^n$ is given by: $\mu : x \mapsto (3 \times x \mod 2^n) \oplus x \oplus (2 \times x)$ and in particular we have: $y_i = \mu_i \oplus x_i \oplus x_{i-1}$.

It is easy to see that the coordinates of $\mu(x)$ follow the recurrence formula:

$$\begin{cases} \mu_0(x) = 0, \\ \mu_1(x) = 0, \\ \mu_i(x) = \mathrm{maj}\left(x_{i-1}, x_{i-2}, \mu_{i-1}(x)\right) & \text{if } i > 1 \ , \end{cases}$$

where $\mathrm{maj}(a, b, c) = ab \oplus bc \oplus ac$ denotes the majority function. We observe that the first non-zero carry is $\mu_2(x)$ and is equal to $x_0 x_1$, and in particular $\deg\left(\mu_2(x)\right) = 2$. We also have that:

$$\begin{cases} \deg\left(\mu_0(x)\right) = \deg\left(\mu_1(x)\right) = -\infty, \\ \deg\left(\mu_i(x)\right) = \max\left(2, \deg\left((x_{i-1} + x_{i-2})\mu_{i-1}(x)\right)\right) & \text{if } i > 1 \ . \end{cases}$$

Since $\mu_{i-1}(x)$ does not depend on x_{i-1}, we have that the degree of $(x_{i-1} + x_{i-2})\mu_{i-1}(x)$ is equal to $1 + \deg(\mu_{i-1}(x))$ and we obtain that:

$$\begin{cases} \deg\left(\mu_0(x)\right) = \deg\left(\mu_1(x)\right) = -\infty, \\ \deg\left(\mu_i(x)\right) = \max\left(2, 1 + \deg\left(\mu_{i-1}(x)\right)\right) = i & \text{if } i > 1 \ . \end{cases}$$

This gives the following relations linking the degrees of the bits of A_α:

$$\begin{cases} \deg(y_0(x)) = 1, \\ \deg(y_i(x)) = i, \text{ if } i \geq 1 \ . \end{cases}$$

This proves that the maximal degree is reached for $\alpha = 3$. We remark that y_i contains only one monomial of degree i (the product of x_0 to x_{i-1}) so we can write $y_i(x)$ as $y_i = x_i + x_0 x_1 x_2 ... x_{i-1} + f_i(x_0, x_1, x_2, ..., x_{i-1})$ where f_i is a Boolean function of degree $i - 1$.

Let $n > 1$. We now consider the case $k > 0$. The mapping A_α can be expressed as $A_\alpha : x \mapsto (3 \times x) \boxplus (4k \times x)$. As previously we denote $y = 3 \times x$, and we also use the notation $t = 4k \times x$ and $\ell = A_\alpha(x) = y \boxplus t$. We also let $h = \min_{i>1}\{\alpha_i = 1\}$.

Using these notations, the value of $A_\alpha(x)$ is given by the following modular sum:

$$\begin{array}{cccccc} & y_{n-1} & \cdots & y_2 & y_1 & y_0 \\ \boxplus & t_{n-1} & \cdots & t_2 & t_1 & t_0 \\ \hline & \ell_{n-1} & \cdots & \ell_2 & \ell_1 & \ell_0 \end{array}$$

Similarly to the case $\alpha = 3$, the bit with index i in ℓ is given by $\ell_i = y_i \oplus t_i \oplus \delta_i$ where δ_i denotes the carry and follows the relation:

$$\begin{cases} \delta_i(x) = 0, & \text{if } i \leq h \\ \delta_i(x) = \mathrm{maj}(y_{i-1}, t_{i-1}, \delta_{i-1}(x)), & \text{otherwise} \ . \end{cases}$$

We observe that $\deg(\delta_{h+1}) = \deg(y_h) = h$, and this is our base case. We make the induction hypothesis that for all $j < i$ we have $\deg(\delta_j) \leq j - 1$. To show that the property holds for bit number i, we make the following additional remarks, deduced from the sum-representation:

- $\deg(t_i) \leq i - 2$ since it depends linearly on x_{i-2} and non-linearly on x_0 to x_{i-3},
- $\delta_i(x)$ is a function of x_0 to x_{i-1}.

We look at $\delta_i(x) = y_{i-1}t_{i-1} + y_{i-1}\delta_{i-1}(x) + \delta_{i-1}(x)t_{i-1}$. The last term depends only on x_0 to x_{i-2} so its degree is lower than or equal to $i-1$. We then focus on the first two terms. With the remark made previously, we can rewrite them as:

$$y_{i-1}(t_{i-1} + \delta_{i-1}(x)) = (x_{i-1} + x_0...x_{i-2} + f_{i-1}(x_0, ..., x_{i-2}))\,(\underbrace{t_{i-1}}_{\deg \leq i-3} + \underbrace{\delta_{i-1}(x)}_{\deg \leq i-2}),$$

where f_{i-1} is of degree $i-2$ and where t_{i-1} and $\delta_{i-1}(x)$ depend respectively on $x_0, ..., x_{i-3}$ and $x_0, ..., x_{i-2}$. This new expression makes clear that the first two terms of $\delta_i(x)$ are at most of degree $i-1$ and so is $\delta_i(x)$. Thus, $\ell_i = y_i \oplus t_i \oplus \delta_i$ contains the degree i monomial $x_0x_1x_2...x_{i-1}$ (present in y_i) that is not canceled since the other terms are of smaller degree. This concludes the proof. □

4.3 Differential properties

An algorithm for DDT coefficients. The following theorem allows an efficient computation of any DDT coefficient of the multiplication by 3. The full proof is in Appendix C.1 but we also provide a proof sketch below.

Theorem 3. *Let $A_3(x) = 3 \times x \mod 2^n$ for $n \geq 2$ and let $\mathcal{D}_A^n(a,b)$ be a coefficient of the Difference Distribution Table (DDT) of A_3, i.e. the number of solutions of $A_3(x) \oplus A_3(x \oplus a) = b$. This quantity can be evaluated in time $\mathcal{O}(n)$ using that:*

$$\mathcal{D}_A^n(a,b) = \lambda_{n-1}^0(a,b) + \lambda_{n-1}^1(a,b),$$

where the sequence $\lambda_{n-1}^z(a,b)$ is defined by:

$$\lambda_1^z(a,b) = 2 \times (1 \oplus a_0 \oplus a_1 \oplus b_1) \times (1 \oplus a_0 \oplus b_0)$$

and by the following induction rule for $i \geq 2$, where $d_i(a,b) = 2 \times (b_i \oplus a_i) \boxplus (b_{i-1} \oplus a_{i-1})$:

$$d_i(a,b) = 0 \implies \begin{cases} \lambda_i^0(a,b) = 2\lambda_{i-1}^1(a,b) + \lambda_{i-1}^0(a,b) \times (1 \oplus a_{i-1} \oplus a_{i-2}) \\ \lambda_i^1(a,b) = \lambda_{i-1}^0(a,b) \times (1 \oplus a_{i-1} \oplus a_{i-2}) \end{cases}$$

$$d_i(a,b) = 1 \implies \begin{cases} \lambda_i^0(a,b) = \lambda_{i-1}^0(a,b) + \lambda_{i-1}^1(a,b) \\ \lambda_i^1(a,b) = 0 \end{cases}$$

$$d_i(a,b) = 2 \implies \lambda_i^0(a,b) = \lambda_i^1(a,b) = \lambda_{i-1}^0(a,b) \times (a_{i-1} \oplus a_{i-2})$$

$$d_i(a,b) = 3 \implies \lambda_i^0(a,b) = \lambda_i^1(a,b) = \lambda_{i-1}^0(a,b).$$

Note that $\lambda_{i-1}^0 = \lambda_{i-1}^1 = 2^{-1} \times \mathcal{D}_A^i((a_0...a_{i-1}),(b_0...b_{i-1}))$ if $d_i(a,b) \in \{1,3\}$.

Sketch. We study $\mu : x \mapsto A_3(x) \oplus x \oplus A_2(x)$ which is extended-affine equivalent to A_3 rather than A_3 itself. We denote its i-th coordinate μ_i . The proof works inductively over the index of the coordinate considered, from the lowest weight to the highest weight. At step i, we count the number of solutions of $\mu_i(x+a) + \mu_i(x) = b$ as a function of the number of solutions of $\mu_{i-1}(x+a) + \mu_{i-1}(x) = b$. The key trick in our proof is to separate the solutions of $\mu_i(x+a) + \mu_i(x) = b$ where $\mu_i(x) = 0$ from those where $\mu_i(x) = 1$. We deduce an inductive formula for computing any coefficient in the DDT of μ. We finally use the extended-affine equivalence of μ and A_3 to obtain a similar algorithm for A_3. □

Sierpinski triangles. The induction described in Theorem 3 has a surprising consequence: the indicator function of the DDT of $\eta(x) = A_3(x) \oplus x$ forms a pattern similar to the Sierpinski triangle, as can be seen in Figure 6. We provide more details in Appendix C.2.

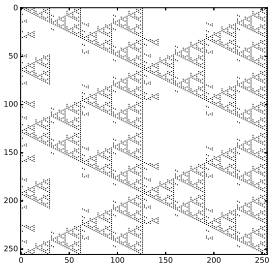


Figure 6: The indicator function of the DDT of $\eta(x) = A_3(x) \oplus x$ for $n = 8$.

Bounding the DDT coefficients. Theorem 3 will also allow us to bound $\mathcal{D}_A^n(a,b)$ using only the value of a. To prove this, we first need the following lemma.

Lemma 2. *The coefficient $\mathcal{D}_A^n(a,b)$ can be bounded by:*

$$\mathcal{D}_A^n(a,b) \leq 2^{1+\sum_{i=1}^{n-1} \nu_i(a,b)} ,$$

where $\nu_i(a,b)$ is equal to either 0 or 1 and is given by:

$$\begin{cases} \nu_1(a,b) = (1 \oplus b_0 \oplus a_0) \times (1 \oplus b_1 \oplus a_1 \oplus a_0) \text{ if } i = 1, \\ \nu_i(a,b) = (1 \oplus b_{i-1} \oplus a_{i-1}) \times (1 \oplus b_i \oplus a_i \oplus a_{i-1} \oplus a_{i-2}) \text{ otherwise} . \end{cases}$$

Sketch. Theorem 3 provides us with a way of computing $\mathcal{D}_A^n(a,b)$ for any a, b. It works by scanning the bits of a and b from lowest weight to highest weight and updating a starting value by either:

1. setting it to zero,
2. leaving it unchanged, or
3. increasing it, in which case it is at most doubled.

A careful study shows that this case 3 only occurs when $\nu_i(a,b) = 1$. The bound follows. □

As before, the details of the proof of Lemma 2 are in Appendix C.3. This bound can be exploited via a simple observation stated in the following lemma.

Lemma 3. *The following implication always holds for $i > 1$:*

$$a_{i-1} \neq a_{i-2} \implies \nu_i(a,b) + \nu_{i+1}(a,b) \leq 1 .$$

Proof. If $\nu_i(a,b) + \nu_{i+1}(a,b) = 2$, then

$$\begin{cases} b_{i-1} \oplus a_{i-1} = 0 \\ b_i \oplus a_i \oplus a_{i-1} \oplus a_{i-2} = 0 \end{cases} \quad \text{and} \quad \begin{cases} b_i \oplus a_i = 0 \\ b_{i+1} \oplus a_{i+1} \oplus a_i \oplus a_{i-1} = 0 . \end{cases}$$

If we add the bottom equation from the left and the top one from the right, we obtain $a_{i-1} \oplus a_{i-2} = 0$. Thus, if $a_{i-1} \neq a_{i-2}$, it is impossible for both systems to be satisfied. □

A naive combination of this lemma with Lemma 3 would give that $\log_2 \mathcal{D}_A^n(a,b) \leq 1 + n - |\{i \leq 1, a_i \neq a_{i-1}\}|$. Indeed, for each position in which $a_i \neq a_{i-1}$, Lemma 2 imposes that the maximum value of the sum of $\nu_i(a,b)$ is decreased by 1. However, this approach does not work out of the box. If $a_{i-2} \neq a_{i-1}$ and $a_{i-1} \neq a_i$, then it is always true that $\nu_i(a,b) + \nu_{i+1}(a,b) \leq 2$ and that $\nu_{i+1}(a,b) + \nu_{i+2}(a,b) \leq 2$ but it is *not* always true that

$\nu_i(a,b) + \nu_{i+1}(a,b) + \nu_{i+2}(a,b) \leq 1$. Thus, bounding the maximum value of the sum by n minus the number of i such that $a_{i-1} \neq a_i$ is wrong.

Nevertheless, we can write a similar bound if we add the condition that if $a_{i-1} \neq a_i$ then we do not take into account whether $a_i \neq a_{i+1}$. We then take into account the quantity $\mathrm{aw}(a)$ defined as:

$$\mathrm{aw}(a) = \sum_{i=1}^{n-1} c_i(a) \ ,$$

where:

$$\begin{cases} c_1(a) = (a_1 \oplus a_0) \\ c_i(a) = \overline{c_{i-1}(a)} \times (a_i \oplus a_{i-1}) \quad \text{if } i > 1 \end{cases}$$

and it then holds that:

$$\mathcal{D}_A^n(a,b) \leq 2^{n+1-\mathrm{aw}(a)}$$

when $\mathcal{D}_A^n(a,b)$ is a coefficient of the DDT of $A_3 : x \mapsto 3 \times x \mod 2^n$ for $n > 2$. Informally, $\mathrm{aw}(a)$ captures the number of changes occurring in a, i.e. the number of times where a run of 0 is interrupted by a 1 and vice-versa. In fact, the induction defining $\mathrm{aw}(a)$ is the exact same used to compute the *arithmetic weight* as introduced in [MG72].

Definition 2 (NAF and Arithmetic Weight)**.** Let a be an element of $\mathbb{Z}/2^n\mathbb{Z}$. The *non-adjacent form (NAF)* of a is the tuple $(e_{n-1}, ..., e_0)$ of elements in $\{-1, 0, 1\}$ such that $a = \sum_{i=0}^{n-1} e_i 2^i$ and such that the number of non-zero coefficients e_i is minimal. It is unique. The number of non-zero coefficients in the NAF of a is called the *arithmetic weight* of a and is denoted $\mathrm{aw}(a)$. For $a \in \mathbb{F}_2^n$, $\mathrm{aw}(a)$ is the arithmetic weight of the integer with a as its binary representation.

We have shown that is possible to bound $\mathcal{D}_A^n(a,b)$ using the arithmetic weight of a. We formalize this result into the following theorem.

Theorem 4 (Main Bound)**.** *Let a be an element in $\mathbb{F}_2^n$ and let $\mathcal{D}_A^n(a,b)$ be a coefficient of the DDT of $A_3 : x \mapsto 3 \times x \mod 2^n$ for $n > 2$. Then, it holds that:*

$$\mathcal{D}_A^n(a,b) \leq 2^{n+1-\mathrm{aw}(a)} \ .$$

In particular, it is possible to bound $\mathcal{D}_A^n(a,b)$ independently from b.

This bound is illustrated in Figure 7(a) where:

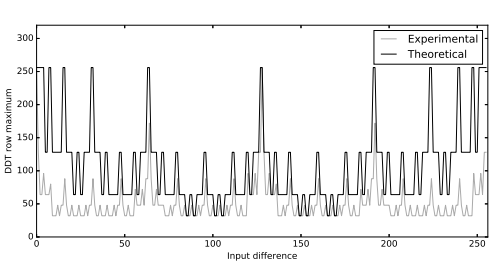
- the first curve was obtained by actually computing the DDT of A_3 to find the maximum coefficient in each row, and
- the second curve was obtained using Theorem 4.

As we can see, the actual maximum is indeed lower than the bound of Theorem 4. Though coarse, this bound is sufficient in practice to serve as the basis for the design of a block cipher as we show in Section 5.1.

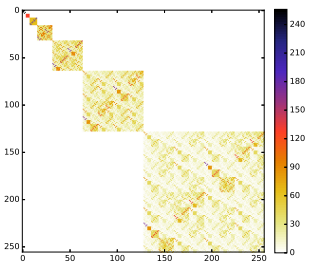
4.4 Linear properties

Theorem 5. *Let (a,b) be a pair of input and output masks. Furthermore, let: i be such that $2^i \leq int(b) < 2^{i+1}$ so that $b = (b_0, ..., b_i, 0, ..., 0)$ and:*

$$L_i : x \mapsto (x_0, ..., x_{i-3}, x_{i-2} + x_{i-1}, 0, .., 0)$$



(a) An experimental illustration of Theorem 4.



(b) The LAT of A_3.

Figure 7: Some properties of A_3 for $n = 8$. Figure (a) is a visual representation of the $\mathrm{LAT}[a][b]$ of A_3, where the x axis corresponds to a and the y axis corresponds to b.

be a linear function. The Walsh coefficient $\mathcal{W}_A(a,b) = \sum_{x\in\mathbb{F}_2^n}(-1)^{\langle a,x\rangle+\langle b,A_3(x)\rangle}$ *is equal to 0 if* $a < 2^i$ *or* $a \geq 2^{i+1}$*. Otherwise, it can be deduced from values of* $\mathcal{W}_A(.,.)$ *for smaller masks using:*

$$\begin{aligned} a' &= L_i(a) \oplus (2^{i-2} + 2^{i-1})b_{i-1} \ , \\ b' &= (b_0, ..., b_{i-1}, 0, ..., 0) \ , \\ \mathcal{W}_A(a,b) &= \frac{\mathcal{W}_A(a',b')}{2} + (-1)^{a_{i-1}+b_{i-1}+1}\frac{\mathcal{W}_A\left(a' \oplus 2^{i-1}, b' \oplus 2^{i-1}\right)}{2} \ . \end{aligned}$$

The proof consists purely in computations given in Appendix C.5.

4.5 On the unsuitability of SPN constructions

Our aim is to design a 128-bit cipher where all operations are linear (in different groups) to ease the implementation of secret sharing inside each round. The most costly part in such an implementation is the secret sharing itself, so we want our cipher to require as few sharing/recombinations as possible while remaining secure. A natural approach would consist in using the wide trail strategy to design a secure Substitution-Permutation Network (SPN). However, we demonstrate here that this approach would require a high number of sharing and recombinations to give a secure cipher. By contrast, our proposal relying on applying both a multiplication by 3 and a binary matrix on the full state of the cipher can be proven resistant to differential attacks with a reasonable complexity.

The wide trail argument was introduced by Daemen and Rijmen who famously used it to design the AES [AES01]. It allows to prove a simple bound on the maximum expected probability of a differential trail covering r rounds of an S-Box-based cipher in two steps. First, we show that the maximum probability of a differential for the S-Box used is upper bounded by a certain quantity:

$$\Pr\left[S(x \oplus a) \oplus S(x) = b\right] \leq u \times 2^{-n} \ ,$$

where u is the *differential uniformity* [Nyb94] of S. Then, we show that any differential trail covering r rounds *activates* at least $a(r)$ S-Boxes and conclude that the expected probability of any single trail covering r rounds is at most $(u2^{-n})^{a(r)}$. When the cipher is a SPN, we can use the branching number b of the linear layer to have that $a(2) = b$.

We could build an n-bit block cipher using multiplication-based S-Boxes operating on m bits, where m divides n. As we allow ourselves arbitrarily complex linear layers, we can have the optimal bound $a(2) = n/m + 1$ by building the linear layer from an MDS code.

As we need to be able to compute the DDT of the S-Boxes considered and since we need that m divides $n = 128$, $m = 16$ is the maximum size of the S-Boxes we consider.

4.5.1 Using multiplications as S-Boxes.

The simplest approach would consist simply in using multiplications by constants modulo 2^m as S-Boxes. The non-linear operation would then correspond to the multiplication by a diagonal matrix of elements in $\mathbb{Z}/2^m\mathbb{Z}$. However, for any $\alpha \in \mathbb{Z}/2^m\mathbb{Z}$, $\Pr\left[\left(\alpha \times (x \oplus 2^{m-1})\right) \oplus (\alpha \times x) = 2^{m-1}\right] = 1$, meaning that the differential uniformity of a multiplication by a constant is always maximum. As a consequence, a wide trail argument cannot work; it would only bound the maximum expected differential probability with 1.

4.5.2 Building S-Boxes with several multiplications.

A simple fix for this issue is to build the S-Boxes by combining two multiplications modulo 2^m interleaved with a multiplication with a binary matrix of size $m \times m$. The multiplication coefficients and the binary matrix could further be optimized to lower the differential uniformity of the whole construction as much as possible. However, while such an approach could be used to build a secure block cipher, it would lead to a higher number of sharing/recombinations than the design strategy we describe in Section 3.

Indeed, as first shown by the attacks targetting SPNs with a "SASAS" structure [BS01, BS10] which were later generalized to more rounds [BKP16], at least 4 S-Box layers are needed to prevent integral attacks. Precisely, the structural attack against SASAS from [BS01, BS10] only needs 2^{2m} plaintext/ciphertext pairs which, for $m \leq 16$, is well under our cap of 2^{64} plaintext/ciphertext pairs.

Furthermore, each such round would contain four secret sharing and recombinations as each round would consist in a key addition, a multiplication in $(\mathbb{Z}/2^m\mathbb{Z})^{n/m}$, a multiplication by a binary matrix of size $n \times n$, another multiplication in $(\mathbb{Z}/2^m\mathbb{Z})^{n/m}$ to finalize the evaluation of the S-Box layer and, finally, another multiplication by a binary matrix of size $n \times n$ to provide diffusion between the S-Boxes.

On the top of this, several more rounds would need to be appended to this construction to take into account the fact that a parallel layer of S-Boxes lends itself to an efficient key recovery over several rounds. Indeed, the key material at the input/output of each S-Box can be brute-forced separately and then recombined in a second step as done, e.g., with the *partial sum technique* introduced in [FKL+01].

Thus, each round requires 4 sharing/recombinations, at least 3 are needed to prevent distinguishers and then at least another 2 such rounds to prevent key recoveries. The total number of sharing/recombinations in each cipher evaluation is thus at least equal to 20. On the other hand, while our proposal requires 2 sharing/recombinations per round and 6 rounds to prevent distinguishers, the full diffusion of its round function implies that 2 additional rounds are sufficient to prevent key recovery. Thus, it needs a total of only 16 sharing/recombinations per encryption.

5 Security analysis of MOE

In this section, we list the results of our attempts at attacking MOE as well as the justification of its security against various attacks. Our best distinguishers are listed in Table 2. Key recovery attacks are made extremely difficult by the strong diffusion of the M layer and the fact that A_3 operates on the full state. We leave the description of such attacks as an open problem. We additionally assessed experimentally the security of small-scale variants of MOE against differential and linear attacks. The results we obtained are consistent with the argument outlined below and give us confidence in the sanity of our design approach. These experiments are described in Section 5.2.

Table 2: Our best distinguishers against round-reduced MOE. Time is measured in number of encryptions, data is measured in number of plaintext/ciphertext pairs; s is any integer such that $2 \leq s < n$.

Description		Complexity		
Type	# rounds	Time	Data	Memory
differential	2	2^{22}	2^{22}	negligible
0-sum	2	2^s	2^s	negligible
impossible diff.	3	2^{128}	2^{128}	negligible

5.1 Proof of security against single-trail differential attacks

In this section, we bound the probability of differential trails in MOE. We show that the majority of binary matrices lead to a cipher that does not have high-probability characteristics after only a few rounds. The starting point of our reasoning is the bound established in Theorem 4: informally, the more changes in the input difference, the lower the probability of the characteristic. Consequently, a lower bound on the number of changes in the vectors in each differential trail implies a lower bound on the probability of all characteristics. This observation made us opt for a `step` function that alternates A_3 and its inverse for the non-linear layer. In this way, the probability that a differential characteristic covers the 3 operations A_3^{-1}, M and A_3 can be bounded by a quantity depending on the arithmetic weights of the output of A_3^{-1} and the input of A_3. Since these two vectors are related by M, we can reformulate our problem as finding the minimum number of changes present in the input and output of the binary matrix M. Given the similarities of this notion with the one of differential branch number, we denote this as the *Change Branch Number (CBN) of* M. We denote by $\mathcal{C}_n^k$ the number of n-bit vectors with an arithmetic weight of k. Its value is given by the following theorem.

Theorem 6. *Let n be a positive integer. The number of n-bit vectors with exactly an arithmetic weight of k is given by:*

$$\mathcal{C}_n^k = \binom{n-1-k}{k} \times 2^{k+1} + \binom{n-1-k}{k-1} \times 2^k$$

Proof. To ease our enumeration, we split the solutions in two sets, depending if a change is present in the last 2 bits or not. If not (first set), it means that exactly k changes start in the $n-2$ first bits of the vector. In case a change is positioned in the last 2 bits (second set), we have $k-1$ changes in the first $n-2$ bits, and the last change of this smaller vector must start at the $n-3$th bit at the latest.[7]

In the following, we denote by P_κ^η the number of possibilities for choosing a valid set of starting indexes of κ changes among η index possibilities.

Let us next consider the first set. To build a solution in this set we start by choosing the positions of the k changes in the $n-2$-bit vector (P_k^{n-2} possible choices) and then fix their values ("01" or "10"). It is easy to see that once these elements are fixed, all the bits between two changes are uniquely determined: indeed, there is only one solution that does not contradict either the total number of changes or their starting positions. However, the bits following the last change are less constrained and can take 2 values: either all-zero or all-one. This implies that the size of the first set is equal to $P_k^{n-2} \times 2^k \times 2$.

We now consider the second set. A change is present in the last 2 positions, so after positioning the other $k-1$ changes only one solution is possible for the other bits. There are P_{k-1}^{n-3} possibilities for positioning the $k-1$ changes, and each of the k changes can take 2 values. Consequently, the size of the second set is $P_{k-1}^{n-3} \times 2^k$.

We finally have to determine the value of P_κ^η. The problem of positioning the start indexes of κ changes in a vector of η bits can be seen as partitioning $\eta - \kappa$ bits with

[7] All along the proof, we focus on the position of the first bit of the pair creating a change.

the following conditions, where the x_i are integers corresponding to the number of bits separating two change starts:

$$\begin{cases} x_0 + x_1 + \cdots + x_\kappa = \eta - \kappa, \\ x_0, x_\kappa \geq 0, \\ x_i > 0 \text{ for } 0 < i < \kappa, \end{cases}$$

where x_0 and x_κ represent the number of bits before the first change and after the last change, respectively, so can be null, whereas the other x_i are at least equal to 1 as a change is made of 2 bits. By subtracting 1 to all the x_i, $0 < i < k$, the conditions can be reformulated as follows:

$$\begin{cases} y_0 + y_1 + \cdots + y_\kappa = \eta - 2\kappa + 1, \\ y_i \geq 0. \end{cases}$$

The number of solutions of this equation is given by a famous combinatoric theorem and is equal to $\binom{\eta-\kappa+1}{\eta-2\kappa+1} = \binom{\eta-\kappa+1}{\kappa}$. This concludes the proof as it shows that the number of n-bit vectors with exactly k changes is:

$$\mathcal{C}_n^k = \binom{n-1-k}{k} \times 2^{k+1} + \binom{n-1-k}{k-1} \times 2^k.$$

□

5.1.1 From the CBN of a random permutation to the one of M

To estimate the expected CBN of a random linear permutation, we first compute the distribution of the CBN of random (non-linear) permutations using Theorem 6. In a second step, we checked experimentally that this distribution is unchanged when we restrict ourselves to random *linear* permutations.

Recall that $\mathrm{aw}(x)$ is the arithmetic weight of $x \in \mathbb{F}_2^n$ and we denote the change branch number of a transformation T as $\mathrm{CBN}(T)$. The CBN of a random permutation S verifies the following:

$$\begin{aligned} \mathbb{P}[\mathrm{CBN}(S) \leq t] =& 1 - \mathbb{P}[\mathrm{CBN}(S) > t] \\ =& 1 - \mathbb{P}[\mathrm{aw}(x) + \mathrm{aw}(S(x)) > t \quad \forall x \neq 0] \\ \approx& 1 - \Big(\prod_{s=1}^{t-1} (\mathbb{P}[\mathrm{aw}(S(x)) > t - s])^{|\{x|\mathrm{aw}(x)=s\}|}\Big) \\ \approx& 1 - \Big(\prod_{s=1}^{t-1} (\mathbb{P}[\mathrm{aw}(x) > t - s])^{|\{x|\mathrm{aw}(x)=s\}|}\Big). \end{aligned} \tag{1}$$

The approximations stems from the assumption that the input and the output of S are independent. The inequality $\mathrm{aw}(x) + \mathrm{aw}(S(x))$ has to hold for all x, so assuming independence, it is the product of the probabilities that it holds for a specific x. Those values are then grouped into x values of a given $\mathrm{aw}(x) = x$. The expression of the probability that the change branch number is equal to t is easily deduced. We experimentally checked this result. Table 3 shows the distribution of the CBN of 500 matrices picked at random from $\mathrm{GL}(24, \mathbb{F}_2)$ along with the expected distribution deduced from Formula 1. The two distributions are very close. We found similar results for values of n up to 28.

Consequently, Formula 1 gives an accurate estimate of the CBN one can expect for a random matrix of $\mathrm{GL}(n, \mathbb{F}_2)$. Further, we observe that the distribution reduces to two values when we increase n, as illustrated in Figure 8. For instance, for $n = 64$, only the

Table 3: Comparison of the CBN distribution deduced from Formula 1 with the experimental distribution obtained with 500 random matrices of $\mathrm{GL}(24, \mathbb{F}_2)$.

CBN	1	2	3	4	5	6	7
# experiment	0	1	8	175	315	1	0
# predicted	0.0	0.2	9.5	161.6	327.5	1.1	0.00

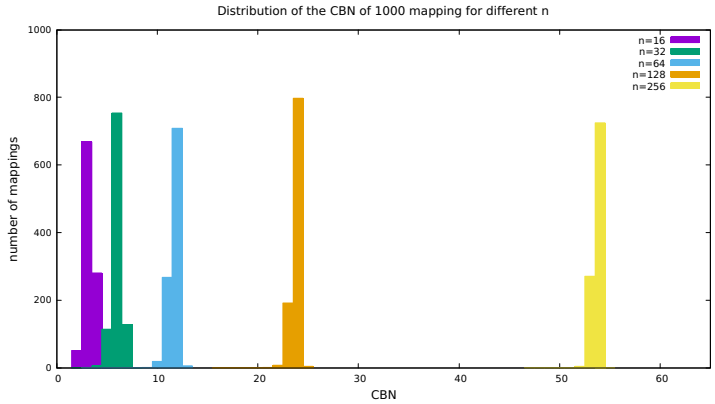


Figure 8: Distribution of the CBN of 1000 mappings for different n obtained with Formula 1.

values of the CBN equal to 11 and 12 have a meaningful probability. For $n = 128$, which is the case we are interested in, we have a change branch number equal to 24 roughly 80% of the time and a change branch number equal to 23 for the remaining 20%. Overall, most mappings share the same (or very close) branch number and that for high values of n the distribution reduces to 2 values. Note that the CBN is upper bounded by the minimal distance. For $n = 128$, the Gilbert-Varshamov bound gives that a random linear code has with high probability a minimum distance of 31, while we found that the CBN will be of 24: our results are consistent with this bound.

5.1.2 From the CBN to a bound on differential characteristics

By combining this CBN bound with Theorem 4, we obtain a bound on the probability of any differential characteristic on one `step` of MOE.

We denote by δ_1, δ_2 and δ_3 the differences at the input of A_3^{-1}, M and A_3 respectively and δ_4 the difference at the output of A_3:

$$\begin{aligned}\Pr\left[\delta_1 \xrightarrow{A_3^{-1}} \delta_2 \xrightarrow{M} \delta_3 \xrightarrow{A_3} \delta_4\right] &= \Pr\left[\delta_2 \xrightarrow{A_3} \delta_1\right] \times \mathbb{1}_{(\delta_3 = M\times\delta_2)} \times \Pr\left[\delta_3 \xrightarrow{A_3} \delta_4\right] \\ &\leq 2^{1-\mathrm{aw}(\delta_2)} \times 2^{1-\mathrm{aw}(\delta_3)} \\ &\leq 2^{2-(\mathrm{aw}(\delta_2)+\mathrm{aw}(\delta_3))} \leq 2^{2-\mathrm{CBN}(M)} \leq 2^{-22}\end{aligned}$$

By iterating the 4 operations A_3^{-1}, M, A_3 and M three times, we obtain a succession of operations for which we can prove that there are no differential characteristics with probability greater than 2^{-66}. Given the cap on the data complexity we consider, this is sufficient to prevent the existence of distinguishers based on differential characteristics. It appears that a reasonable choice would be to fix the number of non-linear layers to 8:

6 to be safe against differential characteristics, plus 2 as a security margin.[8] Recall that M is dense so that inverting 1 round, even if only partially, will require substantial key guessing, which makes us believe that our choice is sensible.

Furthermore, while we *proved* that there exists no exploitable differential characteristics covering 6 rounds of MOE, whether there are in fact such characteristics covering only 4 rounds is an interesting open problem. Indeed, in light of our experiments in Section 5.2, such characteristic may not even exist.

So-called *Multiplicative differentials* [BCJW02] have been used to easily find differential trails in some ciphers using modular multiplications. As we prevent the existence of *any* high probability differential trail, we are in particular safe from those found in this way.

5.2 Experimental results on small scale variants of MOE

Unfortunately, we were not able to find a clean argument proving the security of our algorithm against linear attacks as we did for differential attacks. However, due to the fact that all of the operations of MOE operate on the full state, it is very easy to design variants of MOE operating on smaller blocks.

As detailed below, we seized this opportunity and conducted experiments on variants of our cipher with block sizes from 8 to 16. In short, we experimentally computed the maximum coefficient in the DDT and in the LAT of MOE^n (n being the block size) for different keys, matrices M and round constants. Note that while cryptographers usually work on differential *trails*—it is what our security argument against differential attacks relies on as well— this approach deals with differentials and linear approximations directly. To put it differently, such experimental results are not directly based on the study of patterns propagating throughout the rounds. Therefore, it has the advantage of taking into account the possible clustering of differential or linear trails. In what follows, we describe our approach in more details.

Recall that we limit the data complexity of attacks we want to prevent to $2^{n/2}$. The number of plaintext/ciphertext pairs needed to mount a differential distinguisher for a block cipher instance E_k is essentially:

$$D_{\text{differential}} = \frac{1}{\max_{a\neq 0}(\Pr\left[E_k(x\oplus a)\oplus E_k(x)=b\right])}\,, \tag{2}$$

and, for a linear attack:

$$D_{\text{linear}} = \frac{1}{2\left(\max_{a\neq 0,b}\left(\Pr\left[\langle a,x\rangle = \langle b,E_k(x)\rangle\right]-1/2\right)\right)^2}\,. \tag{3}$$

Both probabilities are taken over all possible keys. In the differential case, we have:

$$\begin{aligned}\max_{a\neq 0,b}(\Pr\left[E_k(x\oplus a)\oplus E_k(x)=b\right]) &= 2^{-n}\max_{a\neq 0,b}\left(\sum_{k\in\mathbb{F}_2^n}\frac{\mathrm{DDT}_{E_k}[a,b]}{2^n}\right)\\ &\leq 2^{-n}\max_{a\neq 0,b}\left(\max_{k\in\mathbb{F}_2^n}(\mathrm{DDT}_{E_k}[a,b])\right)\\ &\leq 2^{-n}\max_{k\in\mathbb{F}_2^n}\left(\max_{a\neq 0,b}(\mathrm{DDT}_{E_k}[a,b])\right)\,,\end{aligned}$$

so that we can estimate a bound on the maximum of this probability by computing the maximum coefficient in the DDT of several keyed instances $\mathsf{MOE}^n_{k_i}$ for $k_i \in \mathcal{K}, \mathcal{K}\subset\mathbb{F}_2^n$, and

[8] This choice is also reasonable if the CBN of M is "only" equal to 23 (according to Formula 1 this occurs for 20% of random mappings) since 6 non-linear layers ensure the absence of differential characteristics with probability greater than 2^{-63}.

looking at the maximum of these quantities which we denote q_d:

$$q_d(\mathcal{K}) = \max_{k_i \in \mathcal{K}} \left(\max_{a \neq 0, b} \left(\text{DDT}(\mathsf{MOE}^n_{k_i})[a, b] \right) \right) ,$$

where $\max_{a\neq 0,b}(\Pr[E_k(x \oplus a) \oplus E_k(x) = b]) \leq 2^{-n} q_d(\mathbb{F}_2^n)$. As we cannot consider all possible keys, we assume $q_d(\mathbb{F}_2^n) \approx q_d(\mathcal{K})$ for the subset $\mathcal{K}$ we experimentally consider.

We want to have $D_{\text{differential}} > 2^{n/2}$. We deduce from Equation (2) that it is equivalent to having the inequality $\max_{a\neq 0,b}(\Pr[E_k(x \oplus a) \oplus E_k(x) = b]) < 2^{-n/2}$. Using what we just established, it holds that:

$$\max_{a \neq 0, b} (\Pr[E_k(x \oplus a) \oplus E_k(x) = b]) \leq 2^{-n} q_d(\mathbb{F}_2^n) \approx 2^{-n} q_d(\mathcal{K}) .$$

In order to prevent the existence of differential distinguisher using less than $2^{n/2}$ plaintext/ciphertext pairs, it is thus sufficient to have $2^{-n} q_d(\mathcal{K}) < 2^{-n/2}$ or, equivalently;

$$\log_2 (q_d(\mathcal{K})) \frac{2}{n} < 1 .$$

Similarly, for the linear case, we look at the maximum coefficient in the LAT of several instances $\mathsf{MOE}^n_{k_i}$ and then keep the maximum of those quantities. We denote the result q_ℓ:

$$q_\ell(\mathcal{K}) = \max_{k_i \in \mathcal{K}} \left(\max_{a \neq 0, b} \left(\mathcal{W}_{\mathsf{MOE}^n_{k_i}} (a, b) \right) \right) ,$$

where $\max_{a\neq 0,b}(\Pr[\langle a, x\rangle = \langle b, E_k(x)\rangle] - 1/2) \leq 2^{-n-1} q_\ell(\mathbb{F}_2^n)$ and we assume $q_\ell(\mathbb{F}_2^n) \approx q_\ell(\mathcal{K})$ for the subset $\mathcal{K}$ we experimentally consider.

Again, our aim is to have $D_{\text{linear}} > 2^{n/2}$. We deduce from Equation (3) that it is equivalent to having:

$$2 \left(\max_{a \neq 0, b} \left(\Pr[\langle a, x\rangle = \langle b, E_k(x)\rangle] - 1/2 \right) \right)^2 < 2^{-n/2} ,$$

a condition which we re-write:

$$\max_{a \neq 0, b} \left(\Pr[\langle a, x\rangle = \langle b, E_k(x)\rangle] - 1/2 \right) < 2^{-n/4-1/2} .$$

We have established that an estimate of the maximum of this probability can be bounded with $2^{-n-1} q_\ell(\mathbb{F}_2^n) \approx 2^{-n-1} q_\ell(\mathcal{K})$, it is therefore sufficient to have:

$$2^{-n-1} q_\ell(\mathcal{K}) < 2^{-n/4-1/2},$$

which is equivalent to:

$$\log_2 (q_\ell(\mathcal{K})) \frac{4}{3n} < 1.$$

To experimentally assess the security of MOE^n for smaller values of n, we thus proceed as follows. For each n, we looked at 20 different matrices M_i and sets of round constants and, for each such instance, at a set of 20 different master keys denoted $\mathcal{K}_i$. We computed the maximum coefficient in the DDT and the LAT of the corresponding permutations and deduced $q_d(\mathcal{K}_i)$ and $q_\ell(\mathcal{K}_i)$. We then plotted the average, minimum and maximum of $\log_2 (q_\ell(\mathcal{K}_i)) \frac{4}{3n}$ and $\log_2 (q_d(\mathcal{K}_i)) \frac{2}{n}$ in Figures 9 and 10 respectively.

As we can see, for 3-round MOE^n, some linear and differential distinguishers with a data complexity under $2^{n/2}$ may exist as neither $\log_2 (q_\ell(\mathcal{K})) \frac{4}{3n}$ nor $\log_2 (q_d(\mathcal{K})) \frac{2}{n}$ are consistently under 1 for the values of n considered. Furthermore, they seem to stabilize just above this number as n increases. Still, we can deduce that any differential or linear

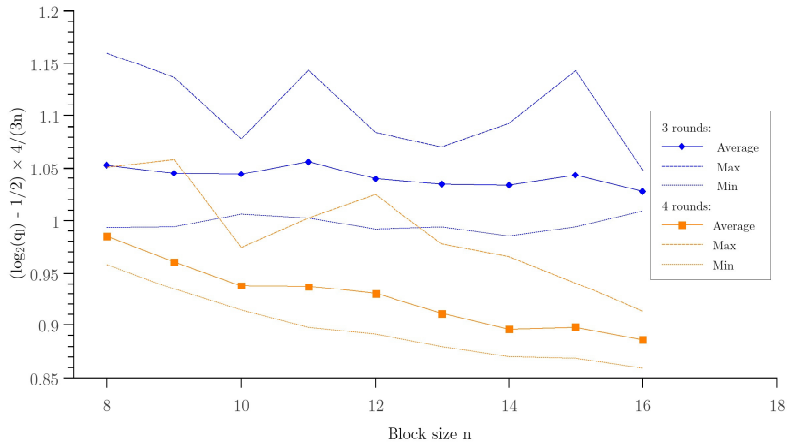


Figure 9: Experimental resilience of MOE^n against linear attacks for small block sizes n.

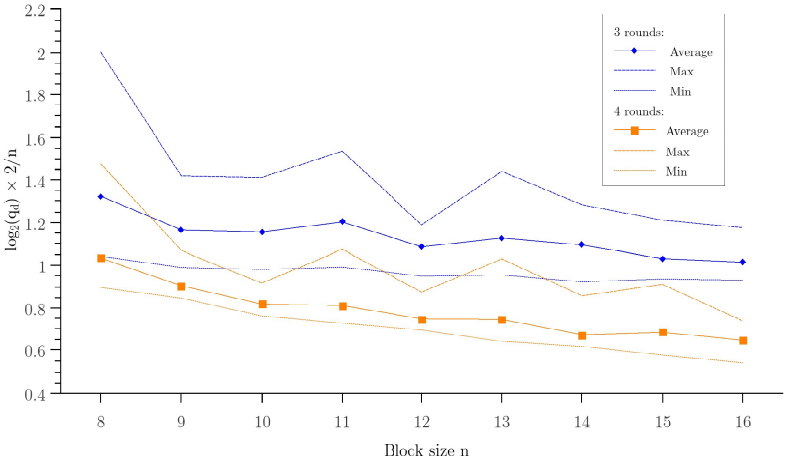


Figure 10: Experimental resilience of MOE^n against differential attacks for small n.

distinguisher covering 3 rounds would need an amount of data close to $2^{n/2}$. We stress that such distinguishers could not be improved using a cluster of differential (or linear) trails as their effect is already taken into account by these experiments.

We also see that no linear or differential distinguisher will cover 4 rounds with a data complexity under $2^{n/2}$. Indeed, the two quantities we consider are mostly under 1 for the small block sizes n we experimentally investigated. Further, they decrease as n increases. We can thus expect that a differential or linear distinguisher covering 4 rounds needs much more than $2^{n/2}$ plaintext/ciphertext pairs. This result is consistent with the bound we have put on the differential probability of 6-round MOE when $n = 128$ as we have shown that no differential trail covers 6 rounds with a probability higher than 2^{-66}. In fact, it seems like our bound is rather loose and 4 rounds may actually be sufficient.

5.3 Other attacks

While our proof and experimental results cover both differential and linear attacks, other types of attacks exist. We summarize our corresponding results below.

5.3.1 Variants of differential and linear Attacks

As evidenced by our experiments, the clustering of differential or linear trails are not a threat to MOE. The diffusion is both very fast (due to the M layer), and does not interact in any particular way with the non-linear operation. As a consequence, this experimental observation is not surprising. These properties of M are also behind the security of MOE against truncated differential attacks.

5.3.2 Impossible differentials

Impossible differential cryptanalysis [Knu98] is a natural attack strategy given that our non-linear layer admits a probability one differential characteristic and that the cipher uses few rounds compared to most algorithms in the literature. In addition, this technique was proven efficient to break up to four and a half rounds of the multiplication-using cipher IDEA [BBS99], a result that was for long the best attack.

In the following, we present an impossible differential that covers the 7 transformations $M, A_3, M, A_3^{-1}, M, A_3, M$, that is 3 rounds out of the 8 that define the cipher. We will see that turning this differential into an efficient attack seems complicated, which shows that impossible differential attacks are not a concern for MOE.

As a starting point, remark that for any choice of α (odd) and for $\alpha = 3$ in particular, a 1-bit difference standing in the most significant bit gives the same difference after A_α. By extending this characteristic through the linear layer, we obtain a differential that covers the 3 steps M, A_3 and M and goes from $M^{-1}(0\cdots01)$ to $M(0\cdots01)$. To build the impossible differential from this, we connect two such differentials by a non-linear step A_3^{-1}. Given the structure of the DDT of A_α (see Appendix C.4) it is easy to see that more than 3/4 of the transitions are impossible. Consequently, with a high probability, the differential going from $\Delta_X = M^{-1}(0\cdots01)$ to $\Delta_Y = M(0\cdots01)$ over the steps $M, A_3, M, A_3^{-1}, M, A_3, M$ is impossible. Note here that the construction of the impossible differential holds for any choice of M and regardless of the non-linear layer, as soon as this layer has probability one (non-trivial) characteristics. However, the non-linear layer impacts the probability that the transition from a probability one characteristic to the other is impossible. A good point (for the attacker) is that our previous discussion on the DDT (see Section 4.3) implies that checking if the transition is impossible can be done efficiently.

A possible idea to use this differential to mount an attack would be to consider a reduced version of MOE with 4 non-linear steps, with the previous impossible differential positioned at the beginning (see Figure 11).

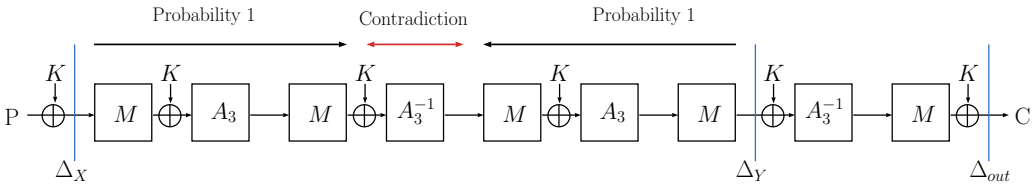


Figure 11: Impossible differential on a reduced version of MOE with 4 non-linear steps.

The idea would then be to ask for the encryption of pairs of messages with the input difference $\Delta_X = M^{-1}(0\cdots01)$. We could test a key by inverting the last linear and non-linear layer from the ciphertext and check if the difference is equal to $\Delta_Y = M(0\cdots01)$. The good point (again for the attacker) is that the definition of A_3 allows to make guesses with a reduced cost: indeed, τ consecutive output bits of A_3 can be computed with the knowledge of $\tau + 1$ consecutive input bits and one bit of carry (this property can be seen on the sum representation of A_3). If this property offers the possibility to make partial

independent key guesses and to combine them later, the total time complexity would still be prohibitive. Since we don't have a strong filter on the ciphertext difference, we would have to roughly examine all the pairs. Each of them would allow to cancel one key. Taking into account the data limitation of 2^{64} plaintext/ciphertext pairs, we conclude that the impossible differential is hard to exploit. Indeed, even as a simpler 3-round distinguisher it would require the full code-book (and thus the time needed to compute it) and a negligible amount of memory: for each pair, if the impossible output difference is observed then the permutation observed cannot be 3-round MOE. It is thus impossible to exploit given the limitation we impose on the data complexity.

5.3.3 Invariant subspaces and 0-sums

Multiplications by constants in $\mathbb{Z}/2^n\mathbb{Z}$ exhibit many invariant subspaces in the sense of [LAAZ11] (see below). However, the permutation M very thoroughly disrupts this pattern and it is thus impossible to use such spaces for an invariant subspace attack. It is nevertheless possible to exploit this property along with the low algebraic degree of the low weight bits of A_3^{-1} to obtain a wide array of 0-sum distinguishers against 2-round MOE (where the last call to M has been removed).

Let us show how such distinguishers would work. Let $s \leq n$ be an integer, let $a \in \mathbb{F}_2^{n-s}$ be a constant, and let $\mathcal{S}_s(a) = \{x||a, x \in \mathbb{F}_2^s\}$ be an affine subspace of $\mathbb{F}_2^n$ of dimension s where the s bits of highest weight take all possible values and where the $n-s$ bits of lowest weight are set to a. Such sets have the following properties:

- $\mathsf{K}_j(\mathcal{S}_s(a)) = \mathcal{S}_s(a')$, where a' is the $n-s$ bits of lowest weight of $a \oplus K \oplus c_j$,
- $A_3(\mathcal{S}_s(a)) = \mathcal{S}_s(a')$ where a' is the $n-s$ bits of lowest weight of $A_3(a)$, and
- $A_3^{-1}(\mathcal{S}_s(a)) = \mathcal{S}_s(a')$ where a' is the $n-s$ bits of lowest weight of $A_3^{-1}(a)$.

Using the terminology first introduced in [LAAZ11], these sets are *invariant subspaces*. However, such structures are completely broken by the $\mathbb{F}_2$-linear layer M and are thus impossible to use to build an invariant subspace attack.

Nevertheless, they yield a powerful 0-sum distinguisher for 2-round MOE, i.e., for $R = \mathsf{K}_2 \circ M \circ A_3 \circ \mathsf{K}_1 \circ M \circ A_3^{-1} \circ \mathsf{K}_0$. Indeed, as we have established in the proof of Theorem 2, the bit at position i in the output of A_α is a function of degree i. This holds in particular for A_3^{-1}. The sum of a function of algebraic degree d over any affine space of dimension strictly greater than d is equal to 0. As the image by M of $\mathcal{S}_s(a)$ is an affine space of dimension s, it holds that:

$$\begin{aligned} 0 &= \bigoplus_{P \in \mathcal{S}_s(a)} \left\langle e_i, \left(M^{-1}(\mathsf{K}_2) \circ A_3 \circ \mathsf{K}_1 \circ M \circ A_3^{-1} \circ \mathsf{K}_0\right)(P) \right\rangle \\ &= \bigoplus_{P \in \mathcal{S}_s(a)} \left\langle e_i, (M^{-1} \circ R)(P) \right\rangle, \end{aligned}$$

where $i < s$ and $(e_0, ..., e_{n-1})$ is the canonical basis of $\mathbb{F}_2^n$. As a consequence, the permutation R corresponding to MOE reduced to 2 rounds exhibits a wide-array of 0-sum distinguishers where the sum over an affine space of dimension s yields a 0-sum over $s-1$ bits. Due to good diffusion of M, it is hard to turn this distinguisher into an attack faster than brute-force. Still, these observations lead to the existence of simple 0-sum distinguishers needing 2^s chosen plaintexts, a time corresponding to the corresponding encryption and a negligible amount of memory as only the sum needs to be stored.

5.3.4 Slide attacks

This type of attack, introduced in [BW99], targets ciphers using identical round functions. Given that different round constants are added during each round, MOE resists slide

attacks. Besides, even in the absence of round constants, distinguishing a slid pair covering M, A_3^{-1}, M, A is hard because all operations operate on the full state. The resulting key recovery would also have a high complexity.

5.3.5 Notes on the key-schedule

The key-schedule we use is trivial as it simply adds the round key every time along with some round constants. This leads to a *complementation-like property* that can speed up an exhaustive search of the key by a factor of 2, much like in the DES. It is the reason why we only claim 127 bits of security against brute-force. Recall also that we do not make any security claim in the related-key setting. We detail this property below.

Consider the encryption of a plaintext P with a key K. We compare this result with the encryption of $(P \oplus M(\Delta))$ under the key $(K \oplus \Delta \oplus M(\Delta))$ where Δ represents the difference with only one bit set to 1, positioned in the MSB. As can be seen in Figure 12, the difference between the two executions is equal to Δ at the input of the first non-linear layer. Since any modular multiplication sends Δ to Δ with probability one, the difference remains constant after this step. The obtained characteristic is iterative on the 3 steps of key addition, linear layer and modular multiplication, and we can see that it spreads with probability one through the whole cipher. Consequently, regardless of the number of rounds, the difference between the two executions can be predicted with probability 1:

$$\mathsf{MOE}_K(P) = \mathsf{MOE}_{K \oplus M(\Delta) \oplus \Delta}(P \oplus M(\Delta)) \oplus \Delta \ .$$

As for the DES, we can use this property to speed-up a brute force attack. The attacker starts by asking for the ciphertexts C and C' corresponding to a random plaintext P and to $P' = (P \oplus M(\Delta))$. She then exhaustively considers one key out of each pair $(K, K \oplus \Delta \oplus M(\Delta))$ and encrypts P with it. If the obtained ciphertext is equal to C, she concludes that her guess is correct. In case she obtains $C' \oplus \Delta$, she concludes that the key used in the cipher differs from her guess of $(\Delta \oplus M(\Delta))$. If none of these relations holds, she continues browsing the keys. The speed-up factor is of one half. Note that this attack works for any choice of α used in the modular multiplication, for any choice of M and for any number of rounds. However, we could easily fix this by choosing a key schedule, or by using different M matrices in each round.

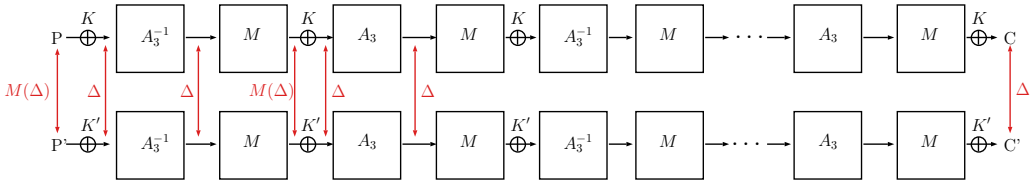


Figure 12: Complementation-like property of MOE: if we denote $\Delta = (0 \cdots 01)$, the differential characteristic depicted here holds with probability 1.

5.3.6 Algebraic attacks

As discussed in Section 3.3, multiple ciphers have been recently designed to be used in specific settings where operation over fields of large degree are desirable. Many of these algorithms have been targeted with efficient algebraic attacks that leveraged the low univariate degree or the simple algebraic structure of the round function over the field used. We expect MOE to be safe from such attacks. Indeed, it relies on operations that are defined over different structures, namely $\mathbb{F}_2^n$ and $\mathbb{Z}/2^n\mathbb{Z}$. Thus, its round function does

not have a simple representation using polynomials with coefficients in either structure (an argument, e.g., found in [BIP+18]). Such polynomials would be both dense and with a high degree, and thus would be unusable to mount an attack against MOE.

6 Performance evaluation

We conclude the paper with an investigation of the performances reached by MOE in a Trojan-resilient setting, based on prototype Printed Circuit Board developed in [BDFS18] implementing traditional block ciphers based on the Trojan-resilient compiler of [DFS16]. In that work, the authors implemented the AES Rijndael and the bitslice cipher Mysterion [JSV17] thanks to the MPC protocol in [AFL+16] and performed an analysis of the performances in two steps. First, they confirmed the reduction of the trusted area that such Trojan-resilient circuits allow; second, they quantified the throughput that can be reached for both ciphers, and its impact on the robustness bounds. In this section, we follow the same steps for the analysis of MOE, and use the same hardware as [BDFS18].

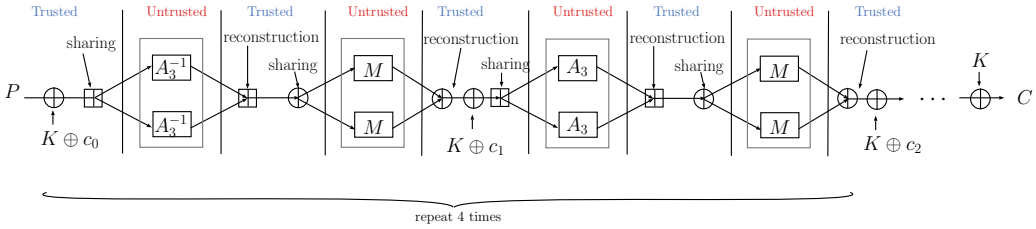


Figure 13: General idea of the Trojan-resilient implementation. To reduce the trusted area, the key additions can be performed in a shared manner. To reach better performance, one can compute the A_3 function in the trusted part.

6.1 Trusted area requirements

The key hypothesis of the previous Trojan-resilient architecture is that the trusted area is small (and at least, significantly smaller than the one of the functionality to implement). The rationale behind this hypothesis is that it should be easier to apply heuristic methods for hardware Trojan detection on the trusted master than on a complex circuit. Concretely, this reduction of the trusted area is achieved in two main directions. Firstly, an unprotected implementation needs trusted memories which takes a large proportion of the design area. For example, storing the key and the state in a low-area PRESENT implementation represents half of it [BKL+07]. Our hardware Trojan architecture gets rid of this requirement by storing shares in untrusted components. Secondly, the logic used for the computations (i.e., S-boxes, matrix multiplications, etc.) is also outsourced to untrusted components leaving only sharing logic in the trusted design. This has the advantage of making the trusted area (per sub-circuit) independent of the target computation (which, for example, would come in handy if a cipher suite had to be implemented in a trusted manner).

Compared to the proposal of Dziembowski et al., the main variation in our case is that the trusted part has to deal with two types of secret sharing operations, namely additions in $\mathbb{F}_2$ and $\mathbb{Z}/2^n\mathbb{Z}$. Considering a bit-serial interface (which is optimal w.r.t. the size of the master [BDFS18]), the first operation requires a single XOR gate, while the second operation requires a serial adder (composed of a full adder and a register for the carry). As a result, the trusted area per sub-circuit which is worth 16[GE] when only Boolean sharing is considered increases to 26[GE] in our case. One interesting additional feature of

MOE is that the constant multiplication A_3 can also be performed by the trusted party, hence saving a few communication rounds for a minimal additional area cost. Since this operation can be executed based on an addition, the serial adder used for the sharing over $\mathbb{Z}/2^n\mathbb{Z}$ can be recycled to compute A_3, given that an additional register is used to shift the serial bits, leading to a total trusted area of 31[GE] per sub-circuit.

For the rest, the majority vote can be implemented identically to the proposal in [BDFS18] since for a given robustness bound, our Trojan resilient cipher requires the same number of sub-circuits as the generic MPC solution.

Considering the variant of our architecture where the key addition is performed by the mini-circuits, and the trusted party is responsible of the secret sharing/reconstruction and A_3 operations together with the majority vote among the sub-circuits, it leads to the values in Table 6. Recalling that a state-of-the-art implementation of the AES requires 2400[GE] [MPL+11], our trusted area in MOE is still one order of magnitude smaller.

We mention that the cost of the multiplications in the untrusted FPGAs is not a primary performance metric in our case. Indeed, since each separate mini-circuit only has to implement a multiplication with a random Boolean matrix or with 3 in $\mathbb{Z}/2^n\mathbb{Z}$ they only consume a small percentage of low-cost commercial FPGAs.

6.2 Throughput and Robustness Bounds

Data throughput is the most relevant metric to compare block ciphers in a Trojan-resilient setting: increased throughput enables both to speed up the testing phase (potentially improving the robustness bounds) and to encrypt data at a higher rate during the online phase. For two algorithms running on the same physical support (and in particular, relying on the same communication performances), the data throughput mainly depends on the communication complexity and the majority vote circuit [BDFS18]. Hereunder, the performances of MOE are compared to the ones of AES and Mysterion.

We first evaluate the communication complexity, which is the key factor influencing the time spent in data transfer for a single encryption. For AES and Mysterion, the communication complexity comes from the multiplicative depth of the algorithm to implement. Thanks to the protocol of [AFL+16] used in [BDFS18], each multiplication requires a single field element to be transferred between two untrusted parties (without latency). Additionally to the multiplication, the initial sharing and final reconstruction of the plaintext and ciphertext also add one communication round each.

In MOE, the communication complexity is mostly proportional to the number of times one has to switch from $\mathbb{F}_2$ to $\mathbb{Z}/2^n\mathbb{Z}$. Each time this happens, the shares are transferred from the untrusted parts to the trusted ones, opened and reshared before being transmitted to the untrusted parts again. The latter can be performed just as the sharing described in [BDFS18], which is executed in a single cycle. It leads to a communication complexity directly proportional to the number of bits to re-share. The overheads due to the initial and final sharing are similar to the ones of AES and Mysterion.

Table 4: Block cipher's communication complexities.

	# of Com. rounds	# of bit transfers per round	# of bit transfers per enc.
AES	12	320	3,456
Mysterion	14	128	1,792
MOE	13	128	1,664

Table 4 contains the communication complexity of each of the investigated block ciphers (including plaintext sharing and ciphertext reconstruction). Taking advantage of the optimizations in [MBPV05], the execution of the AES S-Boxes requires 320 bits to be

exchanged between untrusted parties while Mysterion and MOE only require 128 bits to be transferred per round. Additionally counting the initial sharing and final reconstruction, it leads to a total of $3,456$ bits to be exchanged on the communication bus for the AES. Thanks to its efficient bitslice representation (with a multiplicative depth of one AND gate per round), this number is reduced to $1,792$ bits for Mysterion. It is further reduced to $1,664$ for MOE (thanks to the possibility to perform the operation A_3 in the master, which saves a few additional communication rounds).

We follow with evaluations of the data throughput, which not only depend on the communication complexity but also on the ability to perform a majority vote in the trusted party. The latter depends on the number of sub-circuits λ (the evaluations in [BDFS18] set it at $940/\lambda$[Mbps] using the high-speed communication interfaces of the FPGAs). Yet, since this majority vote is only performed once, when producing the ciphertext, its negative impact on the performances is amortized when increasing λ, as illustrated in Table 5. For $\lambda = 1$, the majority vote can be skipped making the encryption throughput directly related to the communication complexity from Table 4. For $\lambda > 1$, the reduced communication complexity impact is decreased because of the larger time needed for the majority vote.

Table 5: Influence of λ on the encryption throughput ([Mbps]).

λ	1	2	4	8	16
AES	55.0	51.4	46.3	38.7	29.1
Mysterion	103.0	92.6	77.4	58.2	39.0
MOE	115.3	98.7	81.6	60.6	39.9

Putting things together, we can compare the robustness bounds that can be achieved with the different ciphers. Namely, by fixing the number of online executions and the time devoted to the testing phase, we can compute the number of sub-circuits λ required to reach a certain robustness bound (say a probability of hardware Trojan attack lower than 2^{-80}), and therefore the size of the trusted master needed for this purpose.[9] Table 6 contains the results for testing phases of one and seven days. It mostly confirms theoretical expectations and illustrates the lower number of sub-circuits required by MOE compared to the AES, at the cost of an increased trusted area. Gains over Mysterion are more limited due to its throughputs that is quite comparable to MOE.

Note that thanks to its simplification, we can expect that the testing of MOE is performed exactly at these throughputs. By contrast, it may not be the case of AES and Mysterion. As mentioned in introduction, if all the intermediate values need to be communicated to the tester, it may become the bottleneck of the testing phase. And if a dedicated board is used for the testing (with all mini-circuits but one trusted), it still requires the ability to plug/unplug each tested chip on/from this testing hardware. So in general, we believe the simplified testing phase of MOE makes a significant step in the direction of Trojan-resilient block ciphers that can be deployed and tested on-the-fly. The cost reductions allowed by MOE are further highlighted by comparing the total number of mini-circuits ℓ in the Trojan-resilient circuits, since one sub-circuit is made out of three mini-circuits in the generic solution, and only two with MOE.

Note that despite using similar operations, ARX-based block ciphers would also compare negatively with MOE in a Trojan-resilient setting, because of the larger number of transitions between fields it requires (which is the limiting factor for the throughput).

[9] We do not use the bound of Theorem 1 and rather compute $\Pr[\mathsf{ROB} = 1] = \sum_{i=\lceil \lambda/2 \rceil}^{\lambda} \binom{\lambda}{i} \cdot (\frac{\eta}{t})^i \cdot (1 - \frac{\eta}{t})^{\lambda - i}$ directly, which leads to tighter results [DFS16].

Table 6: Robustness guaranttes for different testing periods and trusted area requirements for Trojan-Resilient implementations of the AES, Mysterion and MOE.

Testing [days]	**Online enc.** [bits]	**AES**				**Mysterion**				MOE			
		λ	ℓ	**ROB.**	**Area** [GEs]	λ	ℓ	**ROB.**	**Area** [GEs]	λ	ℓ	**ROB.**	**Area** [GEs]
1	10^3	5	15	2^{-92}	152	5	15	2^{-94}	152	5	10	2^{-94}	222
	10^6	7	21	2^{-82}	198	7	21	2^{-84}	198	7	14	2^{-85}	298
	10^9	17	51	2^{-87}	430	15	45	2^{-81}	383	15	30	2^{-82}	606
7	10^3	5	15	2^{-100}	152	5	15	2^{-102}	152	5	10	2^{-103}	222
	10^6	7	21	2^{-93}	198	7	21	2^{-95}	198	7	14	2^{-96}	298
	10^9	13	39	2^{-89}	337	11	33	2^{-80}	291	11	22	2^{-80}	452

7 Conclusion

In previous works, Trojan-resilience has been approached by adapting current ciphers to fit this specific requirement. In this paper, we have shown that better performances can be obtained if we design an algorithm from the ground up for this purpose. The gains are substantial: a Trojan-resilient implementation of MOE can be up to 2 times faster than one of the AES, and its testing is greatly simplified. Furthermore, MOE is the first cipher of its kind, namely one where all operations are linear. While our aim was to design a cipher optimized for a Trojan-resilient implementation, we can expect the solution we came up with to have applications beyond this use-case. For example, its trivial implementation with secret sharing may be useful in the context of multi-party computation and seems also appealing in the context of masking. Another possible research direction could be in using the scalability of the structure of MOE to design block ciphers with other block sizes, sponge permutations or other primitives which would share MOE's easy secret-shared implementability. Finally, we leave as an open problem the further investigation of the linear properties of the multiplication by 3 modulo 2^n.

Acknowledgments

François-Xavier Standaert is a senior research associate of the Belgian Fund for Scientific Research (FNRS-F.R.S.). This work has been funded in parts by the ERC project 724725 (acronym SWORD). Sebastian Faust was partly funded by the German Research Foundation (DFG) through the Emmy Noether Program FA 1320/1-1. This work was initiated while Virginie Lallemand was with the Horst Görtz Institute for IT Security at the Ruhr-Universität Bochum and was funded by the DFG through LE 3372/4-1. Gregor Leander's work is partially funded by the DFG, under Germany's Excellence Strategy - EXC 2092 CASA - 390781972. Finally, we thank the ToSC reviewers for their comments, as well as Pierre Karpman and Gaëtan Leurent for pointing out a flaw in an earlier version of our security arguments.

References

[AAB+20] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. Design of symmetric-key primitives for advanced cryptographic protocols. *IACR Trans. Symmetric Cryptol.*, 2020(3):1–45, 2020.

[AARP10] Jim Aarestad, Dhruva Acharyya, Reza M. Rad, and Jim Plusquellic. Detecting trojans through leakage current analysis using multiple supply pad i_{ddq} s. *IEEE Trans. Information Forensics and Security*, 5(4):893–904, 2010.

[ABK+07] Dakshi Agrawal, Selçuk Baktir, Deniz Karakoyunlu, Pankaj Rohatgi, and Berk Sunar. Trojan detection using IC fingerprinting. In *2007 IEEE Symposium on Security and Privacy*, pages 296–310. IEEE Computer Society Press, May 2007.

[ACG+19] Martin R. Albrecht, Carlos Cid, Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, and Markus Schofnegger. Algebraic cryptanalysis of STARK-friendly designs: Application to MARVELlous and MiMC. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 371–397. Springer, Heidelberg, December 2019.

[AES01] Advanced Encryption Standard (AES). National Institute of Standards and Technology (NIST), FIPS PUB 197, U.S. Department of Commerce, November 2001.

[AFL+16] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In Weippl et al. [WKK+16], pages 805–817.

[AGP+19] Martin R. Albrecht, Lorenzo Grassi, Léo Perrin, Sebastian Ramacher, Christian Rechberger, Dragos Rotaru, Arnab Roy, and Markus Schofnegger. Feistel structures for MPC, and more. In Kazue Sako, Steve Schneider, and Peter Y. A. Ryan, editors, *ESORICS 2019, Part II*, volume 11736 of *LNCS*, pages 151–171. Springer, Heidelberg, September 2019.

[AGR+16] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 191–219. Springer, Heidelberg, December 2016.

[AKM+18] Giuseppe Ateniese, Aggelos Kiayias, Bernardo Magri, Yiannis Tselekounis, and Daniele Venturi. Secure outsourcing of cryptographic circuits manufacturing. In Joonsang Baek, Willy Susilo, and Jongkil Kim, editors, *ProvSec 2018*, volume 11192 of *LNCS*, pages 75–93. Springer, Heidelberg, October 2018.

[ARS+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 430–454. Springer, Heidelberg, April 2015.

[Ava17] Roberto Avanzi. The QARMA block cipher family. *IACR Trans. Symm. Cryptol.*, 2017(1):4–44, 2017.

[BBS99] Eli Biham, Alex Biryukov, and Adi Shamir. Miss in the middle attacks on IDEA and Khufu. In Knudsen [Knu99], pages 124–138.

[BCCM+08] Emmanuel Bresson, Anne Canteaut, Benoıt Chevallier-Mames, Christophe Clavier, Thomas Fuhr, Aline Gouget, Thomas Icart, Jean-François Misarsky, Marıa Naya-Plasencia, Pascal Paillier, et al. Shabal, a submission to NIST's cryptographic hash algorithm competition. Submission to NIST; available at http://www.shabal.com/wp-content/uploads/Shabal.pdf, 2008.

[BCD+98] Carolynn Burwick, Don Coppersmith, Edward D'Avignon, Rosario Gennaro, Shai Halevi, Charanjit Jutla, Stephen M. Matyas Jr., Luke O'Connor, Mohammad Peyravian, David Safford, and Nevenko Zunic. MARS - a candidate cipher for AES. Submission to the AES competition, available at http://cryptosoft.de/docs/Mars.pdf, June 1998.

[BCD+20] Tim Beyne, Anne Canteaut, Itai Dinur, Maria Eichlseder, Gregor Leander, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, Yu Sasaki, Yosuke Todo, and Friedrich Wiemer. Out of oddity - new cryptanalytic techniques against symmetric primitives optimized for integrity proof systems. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 299–328. Springer, Heidelberg, August 2020.

[BCG+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knežević, 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, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 208–225. Springer, Heidelberg, December 2012.

[BCJW02] Nikita Borisov, Monica Chew, Robert Johnson, and David Wagner. Multiplicative differentials. In Joan Daemen and Vincent Rijmen, editors, *FSE 2002*, volume 2365 of *LNCS*, pages 17–33. Springer, Heidelberg, February 2002.

[BDFS18] Olivier Bronchain, Louis Dassy, Sebastian Faust, and François-Xavier Standaert. Implementing trojan-resilient hardware from (mostly) untrusted components designed by colluding manufacturers. In Chip-Hong Chang, Ulrich Rührmair, Daniel E. Holcomb, and Jorge Guajardo, editors, *ASHES@CCS 2018*, pages 1–10. ACM, 2018. Available at https://perso.uclouvain.be/fstandae/PUBLIS/208.pdf.

[BHBN14] Swarup Bhunia, Michael S. Hsiao, Mainak Banga, and Seetharam Narasimhan. Hardware trojan attacks: Threat analysis and countermeasures. *Proceedings of the IEEE*, 102(8):1229–1247, 2014.

[Bih97] Eli Biham, editor. *FSE'97*, volume 1267 of *LNCS*. Springer, Heidelberg, January 1997.

[BIP+18] Dan Boneh, Yuval Ishai, Alain Passelègue, Amit Sahai, and David J. Wu. Exploring crypto dark matter: New simple PRF candidates and their applications. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 699–729. Springer, Heidelberg, November 2018.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES 2007*, volume 4727 of *LNCS*, pages 450–466. Springer, Heidelberg, September 2007.

[BKP16] Alex Biryukov, Dmitry Khovratovich, and Léo Perrin. Multiset-algebraic cryptanalysis of reduced Kuznyechik, Khazad, and secret SPNs. *IACR Trans. Symm. Cryptol.*, 2016(2):226–247, 2016. http://tosc.iacr.org/index.php/ToSC/article/view/572.

[BNPV02] Alex Biryukov, Jorge Nakahara Jr., Bart Preneel, and Joos Vandewalle. New weak-key classes of IDEA. In Robert H. Deng, Sihan Qing, Feng Bao, and Jianying Zhou, editors, *ICICS 02*, volume 2513 of *LNCS*, pages 315–326. Springer, Heidelberg, December 2002.

[BS01] Alex Biryukov and Adi Shamir. Structural cryptanalysis of SASAS. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 394–405. Springer, Heidelberg, May 2001.

[BS10] Alex Biryukov and Adi Shamir. Structural cryptanalysis of SASAS. *Journal of Cryptology*, 23(4):505–518, October 2010.

[BW99] Alex Biryukov and David Wagner. Slide attacks. In Knudsen [Knu99], pages 245–259.

[DEG+18] Christoph Dobraunig, Maria Eichlseder, Lorenzo Grassi, Virginie Lallemand, Gregor Leander, Eik List, Florian Mendel, and Christian Rechberger. Rasta: A cipher with low ANDdepth and few ANDs per bit. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 662–692. Springer, Heidelberg, August 2018.

[DFS16] Stefan Dziembowski, Sebastian Faust, and Francois-Xavier Standaert. Private circuits iii: Hardware trojan-resilience via testing amplification. Cryptology ePrint Archive, Report 2016/1004, 2016. https://eprint.iacr.org/2016/1004.

[DGV93] Joan Daemen, Rene Govaerts, and Joos Vandewalle. Block ciphers based on modular arithmetic. In *Proc. of the 3rd Symposium on the State and Progress of Research in Cryptography, W. Wolfowicz, Ed., Fondazione Ugo Bordoni*, pages 80–89, 1993.

[DGV94] Joan Daemen, René Govaerts, and Joos Vandewalle. Weak keys for IDEA. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 224–231. Springer, Heidelberg, August 1994.

[FKL+01] Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David Wagner, and Doug Whiting. Improved cryptanalysis of Rijndael. In Bruce Schneier, editor, *FSE 2000*, volume 1978 of *LNCS*, pages 213–230. Springer, Heidelberg, April 2001.

[Fur02] Vladimir Furman. Differential cryptanalysis of Nimbus. In Mitsuru Matsui, editor, *FSE 2001*, volume 2355 of *LNCS*, pages 187–195. Springer, Heidelberg, April 2002.

[GGNS13] Benoît Gérard, Vincent Grosso, María Naya-Plasencia, and François-Xavier Standaert. Block ciphers that are easier to mask: How far can we go? In Guido Bertoni and Jean-Sébastien Coron, editors, *CHES 2013*, volume 8086 of *LNCS*, pages 383–399. Springer, Heidelberg, August 2013.

[GKK+19] Lorenzo Grassi, Daniel Kales, Dmitry Khovratovich, Arnab Roy, Christian Rechberger, and Markus Schofnegger. Starkad and Poseidon: New hash functions for zero knowledge proof systems. Cryptology ePrint Archive, Report 2019/458, 2019. https://eprint.iacr.org/2019/458.

[GLR+20] Lorenzo Grassi, Reinhard Lüftenegger, Christian Rechberger, Dragos Rotaru, and Markus Schofnegger. On a generalization of substitution-permutation networks: The HADES design strategy. In Anne Canteaut and Yuval Ishai,

editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 674–704. Springer, Heidelberg, May 2020.

[GLSV15] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varici. LS-designs: Bitslice encryption for efficient masked software implementations. In Carlos Cid and Christian Rechberger, editors, *FSE 2014*, volume 8540 of *LNCS*, pages 18–37. Springer, Heidelberg, March 2015.

[GRR+16] Lorenzo Grassi, Christian Rechberger, Dragos Rotaru, Peter Scholl, and Nigel P. Smart. MPC-friendly symmetric key primitives. In Weippl et al. [WKK+16], pages 430–443.

[HJMM08] Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. The grain family of stream ciphers. In Matthew J. B. Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 179–190. Springer, 2008.

[HR07] Christopher J. Hillar and Darren L. Rhea. Automorphisms of finite abelian groups. *The American Mathematical Monthly*, 114(10):917–923, 2007.

[IEGT13] Frank Imeson, Ariq Emtenan, Siddharth Garg, and Mahesh V. Tripunitara. Securing computer hardware using 3d integrated circuit (IC) technology and split manufacturing for obfuscation. In Samuel T. King, editor, *Proceedings of the 22th USENIX Security Symposium, Washington, DC, USA, August 14-16, 2013*, pages 495–510. USENIX Association, 2013.

[Ins14] IC Insight. Ic insights research bulletin. http://www.icinsights.com/data/articles/documents/640.pdf, 2014.

[JK97] Thomas Jakobsen and Lars R. Knudsen. The interpolation attack on block ciphers. In Biham [Bih97], pages 28–40.

[JM09] Pascal Junod and Marco Macchetti. Revisiting the IDEA philosophy. In Orr Dunkelman, editor, *FSE 2009*, volume 5665 of *LNCS*, pages 277–295. Springer, Heidelberg, February 2009.

[JSV17] Anthony Journault, François-Xavier Standaert, and Kerem Varici. Improving the security and efficiency of block ciphers based on ls-designs. *Des. Codes Cryptography*, 82(1-2):495–509, 2017.

[Knu98] Lars Knudsen. DEAL-a 128-bit block cipher. *complexity*, 258(2):216, 1998.

[Knu99] Lars R. Knudsen, editor. *FSE'99*, volume 1636 of *LNCS*. Springer, Heidelberg, March 1999.

[KR20] Nathan Keller and Asaf Rosemarin. Mind the middle layer: The HADES design strategy revisited. *IACR Cryptol. ePrint Arch.*, 2020:179, 2020.

[LAAZ11] Gregor Leander, Mohamed Ahmed Abdelraheem, Hoda AlKhzaimi, and Erik Zenner. A cryptanalysis of PRINTcipher: The invariant subspace attack. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 206–221. Springer, Heidelberg, August 2011.

[LM91] Xuejia Lai and James L. Massey. A proposal for a new block encryption standard. In Ivan Damgård, editor, *EUROCRYPT'90*, volume 473 of *LNCS*, pages 389–404. Springer, Heidelberg, May 1991.

[Mac00] Alexis Warner Machado. The Nimbus Cipher. First Open NESSIE Workshop, see also https://www.cosic.esat.kuleuven.be/nessie/workshop/submissions/nimbus.zip, 2000.

[MBPV05] Nele Mentens, Lejla Batina, Bart Preneel, and Ingrid Verbauwhede. A systematic evaluation of compact hardware implementations for the Rijndael S-box. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 323–333. Springer, Heidelberg, February 2005.

[MCS+17] Vasilios Mavroudis, Andrea Cerulli, Petr Svenda, Dan Cvrcek, Dusan Klinec, and George Danezis. A touch of evil: High-assurance cryptographic hardware from untrusted components. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1583–1600. ACM Press, October / November 2017.

[MG72] James L. Massey and Oscar N. García. Error-correcting codes in computer arithmetic. In Julius T. Tou, editor, *Advances in Information Systems Science: Volume 4*, pages 273–326. Springer US, Boston, MA, 1972.

[MNSV97] David M'Raïhi, David Naccache, Jacques Stern, and Serge Vaudenay. XMX: A firmware-oriented block cipher based on modular multiplications. In Biham [Bih97], pages 166–171.

[MPL+11] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the limits: A very compact and a threshold implementation of AES. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 69–88. Springer, Heidelberg, May 2011.

[MRTV13] Florian Mendel, Vincent Rijmen, Deniz Toz, and Kerem Varici. Collisions for the WIDEA-8 compression function. In Ed Dawson, editor, *CT-RSA 2013*, volume 7779 of *LNCS*, pages 162–173. Springer, Heidelberg, February / March 2013.

[MS95] Willi Meier and Othmar Staffelbach. The self-shrinking generator. In Alfredo De Santis, editor, *EUROCRYPT'94*, volume 950 of *LNCS*, pages 205–214. Springer, Heidelberg, May 1995.

[MVDP11] Nicky Mouha, Vesselin Velichkov, Christophe De Cannière, and Bart Preneel. The differential analysis of S-functions. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *SAC 2010*, volume 6544 of *LNCS*, pages 36–56. Springer, Heidelberg, August 2011.

[Nak12] Jorge Nakahara Jra. Differential and linear attacks on the full WIDEA-n block ciphers (under weak keys). In Josef Pieprzyk, Ahmad-Reza Sadeghi, and Mark Manulis, editors, *CANS 12*, volume 7712 of *LNCS*, pages 56–71. Springer, Heidelberg, December 2012.

[NK95] Kaisa Nyberg and Lars R. Knudsen. Provable security against a differential attack. *Journal of Cryptology*, 8(1):27–37, December 1995.

[NRPV04] Jorge Nakahara Jr., Vincent Rijmen, Bart Preneel, and Joos Vandewalle. The MESH block ciphers. In Kijoon Chae and Moti Yung, editors, *WISA 03*, volume 2908 of *LNCS*, pages 458–473. Springer, Heidelberg, August 2004.

[Nyb94] Kaisa Nyberg. Differentially uniform mappings for cryptography. In Tor Helleseth, editor, *EUROCRYPT'93*, volume 765 of *LNCS*, pages 55–64. Springer, Heidelberg, May 1994.

[Rib14] John Ribeiro. Samsung investing $14.7 billion in new chip fabrication facility. http://www.pcworld.com/article/2691992/samsung-to-invest-147-billion-in-new-fab.html, 2014. PCWorld.

[Scr01] Beale Screamer. "Microsoft's Digital Rights Management Scheme – technical details", 2001.

[TK10] Mohammad Tehranipoor and Farinaz Koushanfar. A survey of hardware trojan taxonomy and detection. *IEEE Design & Test of Computers*, 27(1):10–25, 2010.

[WKK+16] Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors. *ACM CCS 2016*. ACM Press, October 2016.

[WNS09] Meiqin Wang, Jorge Nakahara Jr., and Yue Sun. Cryptanalysis of the full MMB block cipher. In Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *SAC 2009*, volume 5867 of *LNCS*, pages 231–248. Springer, Heidelberg, August 2009.

[WS11] Adam Waksman and Simha Sethumadhavan. Silencing hardware backdoors. In *2011 IEEE Symposium on Security and Privacy*, pages 49–63. IEEE Computer Society Press, May 2011.

[XFJ+16] Kan Xiao, Domenic Forte, Yier Jin, Ramesh Karri, Swarup Bhunia, and Mark Mohammad Tehranipoor. Hardware trojans: Lessons learned after one decade of research. *ACM Trans. Design Autom. Electr. Syst.*, 22(1):6:1–6:23, 2016.

A Trojan robust construction from [DFS16]

For completeness we recall the construction of [DFS16] based on a passively secure 3-party computation protocol. The formal construction is given in Figure 14 and taken from [DFS16]. The construction works as a general compiler that can protect any computation against Trojan attacks. To this end, the compiler takes a specification of an algorithm modeled as an arithmetic circuit Γ as input, and outputs a protected circuit Γ'. At a high-level Γ' consists of multiple mini-circuits $(\Gamma_1, \Gamma_2, \Gamma_3)$ that take the role of the parties in the passively secure 3-party computation protocol. Since the compiler works gate-by-gate, Figure 14 presents transformations for each single arithmetic operation in Γ. As usual, linear operations are almost for free and do not require any interaction between the mini-circuits Γ_i. On the other hand the non-linear operations are costly and in particular require interaction between the mini-circuits (the communication is also illustrated in Figure 15). The high communication complexity of the non-linear gates is also the reason for the high costs of the tester $\mathsf{T}_{\mathsf{DSF}}$ as essentially the entire communication between the mini-circuits $(\Gamma_0, \Gamma_1, \Gamma_2)$ has to be tested for correctness.

B General theorems about the possible candidate groups

The first important thing we need to figure out is how to implement the linear operations, that is, what are the computations made by the untrusted mini-circuits. This problem can be reformulated as defining a finite Abelian group and selecting one of its automorphisms. We hereafter refer to the paper *Automorphisms of Abelian Groups* [HR07] of Hillar and Darren to figure out the possible options for implementing the linear operations.

The first well-know result that we use is the following:

Evaluating the gates g of Γ by $(\Gamma_0, \Gamma_1, \Gamma_2)$

1. *Transformation for field addition, i.e., $\vec{a} \widehat{\oplus} \vec{b} = \vec{c}$:* Γ_1 holds the shares (a_1, b_1) that either were received from $\mathcal{M}$ via an in command, or resulted as an output from a previous gate. Similarly, Γ_2 holds (a_2, b_2). Given these inputs Γ_1 computes $c_1 = a_1 \oplus b_1$ and Γ_2 computes $c_2 = a_2 \oplus b_2$.
2. *Transformation for multiplication, i.e., $\vec{a} \widehat{\odot} \vec{b} = \vec{c}$:* This involves the mini-circuits $(\Gamma_0, \Gamma_1, \Gamma_2)$ and the master circuit to forward commands between the circuits Γ_i. To keep the description simple, we will not explicitly describe the computation carried of by $\mathcal{M}$ as it only forwards commands. Initially, Γ_1 holds (a_1, b_1) and Γ_2 has (a_2, b_2). They proceed as follows:
   (a) Run jointly $(u, v) \leftarrow \mathsf{MultShares}(a_1, b_2)$ and $(u', v') \leftarrow \mathsf{MultShares}(b_1, a_2)$ (see description below).
   (b) *Mini-circuit Γ_1:* Compute $c_1 = a_1 \odot b_1 \oplus u \oplus u'$ and output c_1.
   (c) *Mini-circuit Γ_2:* Compute $c_2 = a_2 \odot b_2 \oplus v \oplus v'$ and output c_2.

Sub-circuit $(u, v) \leftarrow \mathsf{MultShares}(x, y)$

Initially, Γ_1 holds x and Γ_2 holds y. At the end Γ_1 holds u and Γ_2 has v such that $v = x \odot y \oplus u$ and $u \in \mathbb{F}$ defined below.

1. *Mini-circuit $\Gamma_0[\vec{w}]$:* Γ_0 has memory cells to store the internal state of the PRG $\vec{w}$. Notice that Γ_0 uses the contents of its memory cells $\vec{w}$ and computes $(\vec{w}, (u_1, u_2, u_3, u_4)) = \mathsf{prg}(\vec{w})$, where the output $\vec{w}$ represents the secret output. It then computes $u = u_3 \oplus u_4 \ominus u_1 \odot u_2$ and sends $((\mathsf{send}, 1), (u, u_2, u_3))$ and $((\mathsf{send}, 2), (u_1, u_4))$ to $\mathcal{M}$.
2. *Mini-circuit Γ_2:* on receiving $((\mathsf{send}, 2), (u_1, u_4))$ from $\mathcal{M}$, compute $z = y \oplus u_1$ and send $((\mathsf{send}, 1), z)$ to $\mathcal{M}$.
3. *Mini-circuit Γ_1:* on receiving $((\mathsf{send}, 1), (u, u_2, u_3, z))$ from $\mathcal{M}$, compute $e = (z \odot x) \oplus u_3$ and $f = x \oplus u_2$. Send $((\mathsf{send}, 2), (e, f))$ to $\mathcal{M}$.
4. *Mini-circuit Γ_2:* on receiving $((\mathsf{send}, 2), (e, f))$ from $\mathcal{M}$, compute $v = u_4 \oplus e \ominus f \odot u_1$.

Figure 14: The computation of the gates by the mini-circuits $(\Gamma_0, \Gamma_1, \Gamma_2)$. All operations are field operations in the underlying field $\mathbb{F}$. The MultShares circuit is called in the field multiplication operation, where the latter is also shown in Figure 15.

Theorem 7. *([HR07]) A finite Abelian group is isomorphic to a product of groups of the form:*

$$H_p = \mathbb{Z}/p^{e_1}\mathbb{Z} \times \cdots \times \mathbb{Z}/p^{e_t}\mathbb{Z}$$

The next theorem defines the endomorphism of H_p:

Theorem 8. *([HR07]) An endomorphism of H_p corresponds to the multiplication by a matrix $A \in R_p$ on a vector of integer representatives, followed by a reduction by the standard quotient mapping* [10]*, where R_p is defined as:*

$$R_p = \{(a_{ij} \in \mathbb{Z}^{t \times t} : p^{e_i - e_j} | a_{ij} \text{ for all } i \text{ and } j \text{ satisfying } 1 \leq j \leq i \leq n\}).$$

Finally, the automorphisms are characterized as follows:

Theorem 9. *([HR07]) An endomorphism of H_p is an automorphism if and only if $A (mod\ p) \in GL_t(\mathbb{F}_p)$.*

[10] i.e. coefficient i of the output vector is mapped to its corresponding class element in $\mathbb{Z}/p^{e_i}\mathbb{Z}$.

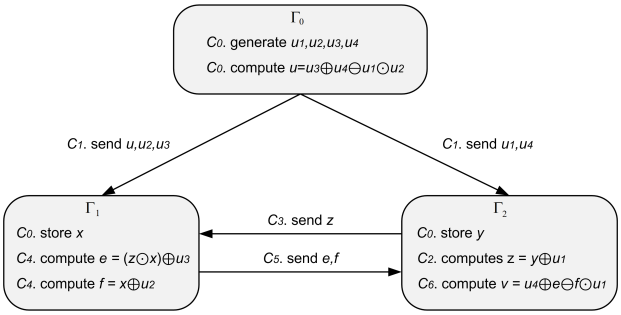


Figure 15: MultShares with threee mini-circuits.

In the remaining of the paper, we focus on the two following extreme cases for $p = 2$:

- $\forall i \in \{1, \cdots, t\}$, $e_i = 1$. In this case, the automorphism group of H_2 corresponds to the set of invertible matrices $GL_t(\mathbb{F}_2)$.
- $H_2 = \mathbb{Z}/2^n\mathbb{Z}$. The automorphisms of H_2 are the multiplications by an integer α so that $\alpha \mod 2 = 1$. These are the modular multiplications by an odd integer.

C Differential and linear properties of the multiplication by 3 modulo a power of 2

C.1 Proof of Theorem 3 (DDT Formula)

We consider the function $\eta : x \mapsto (3 \times x \mod 2^n) \oplus x$ which maps $\mathbb{F}_2^n$ to itself. It is extended affine equivalent to the multiplication by 3. When η operates on n bits, we use the following notation for its DDT coefficients:

$$\mathcal{D}_\eta^n(a, b) = \#\{x, \eta(x + a) + \eta(x) = b\} .$$

The coordinates of η are functions $\eta_i(x)$ such that:

$$\begin{cases} \eta_0(x) = 0, \\ \eta_1(x) = x_0, \\ \eta_i(x) = (x_{i-1} + x_{i-2})(1 + \eta_{i-1}(x)) \end{cases} .$$

The last line is obtained using the facts that $(x, y, z) \mapsto xy + (x + y)z$ is the majority function and that $\eta_i(x) = x_{i-1} + \mathrm{maj}(x_{i-1}, x_{i-2}, \eta_{i-1}(x) + x_{i-2})$. Note that η_i does not depend on x_j for $j \geq i$, meaning that η_i has all of its inputs in $\mathbb{F}_2^i$.
The following notations are used throughout this proof:

- We let $\mathcal{D}_\eta^n(a, b)$ be the DDT coefficients of η when it operates on n bits.
- We let $\mathcal{S}_i^z$ be defined for $z \in \mathbb{F}_2$ and $i \leq n - 1$ as follows:

$$\mathcal{S}_i^z = \left\{x = (x_0, ..., x_{i-1}) \in \mathbb{F}_2^i \mid \forall j \leq i, \eta_j(x + a) + \eta_j(x) = b_j \text{ and } \eta_i(x) = z\right\} .$$

- We further denote $\lambda_i^z = |\mathcal{S}_i^z|$ so that, in particular, $\mathcal{D}_\eta^n(a, b) = \lambda_{n-1}^0 + \lambda_{n-1}^1$.
- We let T be the function truncating the highest weight bit:

$$T : \begin{cases} \mathbb{F}_2^n \to \mathbb{F}_2^{n-1} \\ (x_0, ..., x_{n-1}) \mapsto (x_0, ..., x_{n-2}) \end{cases} .$$

Lemma 4. *The size λ_{n-1}^z of $\mathcal{S}_{n-1}^z$ can be obtained by applying the following rules:*

$$b_{n-1}=0, b_{n-2}=0 \implies \begin{cases}\lambda_{n-1}^0 = 2\lambda_{n-2}^1 + \lambda_{n-2}^0 \times (1 \oplus a_{n-2} \oplus a_{n-3}) \\ \lambda_{n-1}^1 = \lambda_{n-2}^0 \times (1 \oplus a_{n-2} \oplus a_{n-3})\end{cases}$$

$$b_{n-1}=0, b_{n-2}=1 \implies \begin{cases}\lambda_{n-1}^0 = \lambda_{n-2}^0 + \lambda_{n-2}^1 \\ \lambda_{n-1}^1 = 0\end{cases}$$

$$b_{n-1}=1, b_{n-2}=0 \implies \lambda_{n-1}^0 = \lambda_{n-1}^1 = \lambda_{n-2}^0 \times (a_{n-2} \oplus a_{n-3})$$

$$b_{n-1}=1, b_{n-2}=1 \implies \lambda_{n-1}^0 = \lambda_{n-1}^1 = \lambda_{n-2}^0 .$$

Proof. First of all, we observe that if $b_i = 1$ then it must hold that $|\mathcal{S}_i^0| = |\mathcal{S}_i^1|$ because if $x \in \mathcal{S}_i^z$ then $x + a \in \mathcal{S}_i^{1\oplus z}$. Suppose that $|\mathcal{S}_{n-2}^0|$ and $|\mathcal{S}_{n-2}^1|$ are known. We compute $|\mathcal{S}_{n-1}^0|$ and $|\mathcal{S}_{n-1}^1|$ by separating the case $b_{n-1} = 0$ and the case $b_{n-1} = 1$. In each case, we look at which values of $\eta_{n-1}(x)$ actually allow the right output difference to occur and link their number to $\mathcal{S}_{n-2}^z$:

- $\mathbf{b_{n-1} = 1}$. If $\eta_{n-1}(x + a) + \eta_{n-1}(x) = 1$, then we must be in one of these two cases:

$$\begin{cases}\eta_{n-1}(x) = 0 \\ \eta_{n-1}(x + a) = 1\end{cases} \quad \text{or} \quad \begin{cases}\eta_{n-1}(x) = 1 \\ \eta_{n-1}(x + a) = 0 .\end{cases}$$

  Using that $\eta_{n-1}(x) = (x_{n-2}+x_{n-3})(1+\eta_{n-2}(x))$, we deduce that, with $a_{n-2}+a_{n-3} = \alpha$, one of the following two systems must hold:

$$\begin{cases}\eta_{n-2}(x) = 1 \text{ or } x_{n-2} + x_{n-3} = 0 \\ \eta_{n-2}(x + a) = 0 \\ x_{n-2} + x_{n-3} + \alpha = 1,\end{cases} \tag{4}$$

  or:

$$\begin{cases}\eta_{n-2}(x) = 0 \\ x_{n-2} + x_{n-3} = 1 \\ \eta_{n-2}(x + a) = 1 \text{ or } x_{n-2} + x_{n-3} + \alpha = 0 .\end{cases} \tag{5}$$

  The remainder depends on whether $b_{n-2} = 0$ or $b_{n-2} = 1$.

  - $\mathbf{b_{n-2} = 0}$. Let $(x_0, .., x_{n-3}) \in \mathcal{S}_{n-2}^0$, so that $\eta_{n-2}(x + a) = \eta_{n-2}(x) = 0$. Then it is necessary and sufficient that both $x_{n-2} + x_{n-3} = 0$ and $\alpha = 1$ for the system in Equation (4) to have solutions. Similarly, it is necessary and sufficient that $\alpha = 1$ and $x_{n-2} + x_{n-3} = 1$ for $(x_0, ..., x_{n-2})$ to be a solution of the system in Equation (5). Thus, x_{n-2} can be chosen freely to build a solution of $\eta_{n-1}(x+a)+\eta_{n-1}(x) = b_{n-1}$ if and only if $\alpha = 1$, i.e., if and only if $a_{n-2} \neq a_{n-3}$. Suppose now that $(x_0, .., x_{n-3}) \in \mathcal{S}_{n-2}^1$, so that $\eta_{n-2}(x) = \eta_{n-2}(x + a) = 1$. Then none of the systems may have a solution. We conclude that:

$$b_{n-1}=1, b_{n-2}=0 \implies |\mathcal{S}_{n-1}^0| = |\mathcal{S}_{n-1}^1| = |\mathcal{S}_{n-2}^0| \times (a_{n-2} \oplus a_{n-3}), \tag{6}$$

    where there is no simple link between $|\mathcal{S}_{n-2}^0|$ and $\mathcal{D}_\eta^{n-1}(T(a), T(b))$ as $b_{n-2} = 0$.
  - $\mathbf{b_{n-2} = 1}$. In this case, $\eta_{n-2}(x + a) = \eta_{n-2}(x) + 1$ for any x in $\mathcal{S}_{n-2}^0 \cup \mathcal{S}_{n-2}^1$. If $x \in \mathcal{S}_{n-2}^0$, then the system in Equation (4) has no solution and the one in Equation (5) has a unique solution. Similarly, if $x \in \mathcal{S}_{n-2}^1$, then the system in Equation (5) has no solution and the one in Equation (4) has a unique

one. Thus, for each $x \in \mathcal{S}^0_{n-2} \cup \mathcal{S}^1_{n-2}$ we can build a unique x_{n-2} such that $(x_0, ..., x_{n-2}) \in \mathcal{S}^0_{n-1} \cup \mathcal{S}^1_{n-1}$. We conclude that the following holds:

$$b_{n-1} = 1, b_{n-2} = 1 \implies |\mathcal{S}^0_{n-1}| = |\mathcal{S}^1_{n-1}| = |\mathcal{S}^0_{n-2}| \ , \tag{7}$$

where $|\mathcal{S}^0_{n-2}| = |\mathcal{S}^1_{n-2}|$, so that:

$$b_{n-1} = 1, b_{n-2} = 1 \implies \mathcal{D}^n_\eta(a, b) = \mathcal{D}^{n-1}_\eta(T(a), T(b)) \ .$$

- $\mathbf{b_{n-1} = 0}$. If $\eta_{n-1}(x + a) + \eta_{n-1}(x) = 0$, then we must be in one of these two cases:

$$\begin{cases} \eta_{n-1}(x) = 0 \\ \eta_{n-1}(x + a) = 0 \end{cases} \quad \text{or} \quad \begin{cases} \eta_{n-1}(x) = 1 \\ \eta_{n-1}(x + a) = 1 \ . \end{cases}$$

Using that $\eta_{n-1}(x) = (x_{n-2}+x_{n-3})(1+\eta_{n-2}(x))$, we deduce that, with $a_{n-2}+a_{n-3} = \alpha$, one of the following two systems must hold:

$$\begin{cases} \eta_{n-2}(x) = 1 \text{ or } x_{n-2} + x_{n-3} = 0 \\ \eta_{n-2}(x + a) = 1 \text{ or } x_{n-2} + x_{n-3} + \alpha = 0 \end{cases} \quad \text{or} \quad \begin{cases} \eta_{n-2}(x) = 0 \\ x_{n-2} + x_{n-3} = 1 \\ \eta_{n-2}(x + a) = 0 \\ x_{n-2} + x_{n-3} + \alpha = 1. \end{cases}$$

The remainder depends on whether $b_{n-2} = 0$ or $b_{n-2} = 1$. Unlike in the previous case, the two systems are no longer symmetric. The number of solutions of the system on the right will give $\mathcal{S}^0_{n-1}$ while the number of solutions of the system the left correspond to $\mathcal{S}^1_{n-1}$.

  - $\mathbf{b_{n-2} = 0}$. In this case, $\eta_{n-2}(x+a) = \eta_{n-2}(x)$. The left system has a solution if:
    * $(x_0, ..x_{,n-3}) \in \mathcal{S}^1_{n-2}$, in which case we can choose x_{n-2} freely, or
    * $(x_0, ..x_{,n-3}) \in \mathcal{S}^0_{n-2}$ and $x_{n-2} + x_{n-3} = x_{n-2} + x_{n-3} + \alpha = 0$, which has a unique solution if and only if $\alpha = 0$.

    Thus, we obtain the following implication:

$$b_{n-1} = 0, b_{n-2} = 0 \implies |\mathcal{S}^0_{n-1}| = 2|\mathcal{S}^1_{n-2}| + |\mathcal{S}^0_{n-2}| \times (a_{n-2} \oplus a_{n-3} \oplus 1) \ .$$

    The right system has a unique solution if and only if $x \in \mathcal{S}^0_{n-2}$ and $\alpha = 0$, so:

$$b_{n-1} = 0, b_{n-2} = 0 \implies |\mathcal{S}^1_{n-1}| = |\mathcal{S}^0_{n-2}| \times (a_{n-2} \oplus a_{n-3} \oplus 1) \ .$$

    As a consequence, if $a_{n-2} = a_{n-3}$ and if $b_{n-1} = b_{n-2} = 0$, then $\mathcal{D}^n_\eta(a, b) = 2 \times \mathcal{D}^{n-}_\eta(a, b)$.
  - $\mathbf{b_{n-2} = 1}$. In this case, $\eta_{n-2}(x + a) = 1 + \eta_{n-2}(x)$ which implies that the right system has no solutions. We deduce that:

$$b_{n-1} = 0, b_{n-2} = 1 \implies \mathcal{S}^1_{n-1} = \emptyset \ .$$

    On the other hand, the left equation always has exactly one solution. If $x \in \mathcal{S}^0_{n-2}$ then the second equation is satisfied and setting $x_{n-2} = x_{n-3}$ satisfies the first. Identically, if $x \in \mathcal{S}^1_{n-2}$ then the first equation is satisfied and the second is satisfied as well if and only if $x_{n-2} = x_{n-3} + \alpha$. Thus, the following holds:

$$b_{n-1} = 0, b_{n-2} = 1 \implies |\mathcal{S}^0_{n-1}| = |\mathcal{S}^0_{n-2}| + |\mathcal{S}^1_{n-2}| \ .$$

The last two implications show in particular that $\mathcal{D}^n_\eta(a, b)$ is equal to $\mathcal{D}^{n-1}_\eta(T(a), T(b))$.

□

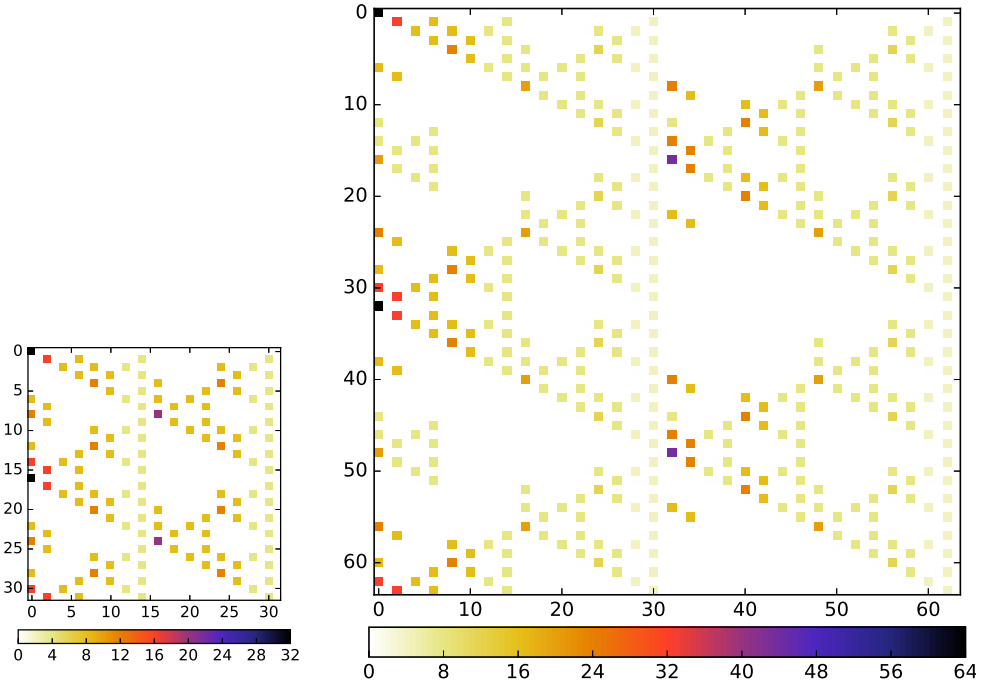
Theorem 3 is obtained directly from Lemma 4 using the extended-affine equivalence between η and A_3 which imposes that:

$$\mathcal{D}^n_A(a, b) = \mathcal{D}^n_\eta(a, a \oplus b) \ .$$

C.2 On the Sierpinski triangle pattern of the DDT of η

The Sierpinski triangle is a fractal that is named after Wacław Sierpiński, who described it in 1915. It can be seen as a full equilateral triangle with smaller triangle-shaped parts removed. Among the various ways to build it, one consists in shrinking and duplicating shapes. We starts with a full equilateral triangle, and shrink it to make it two times smaller. We take 3 copies of this smaller triangle and position them so that they touch in their corners. This newly formed figure is our new starting point: we shrink it, make 3 copies of it and assemble it, and so on.

When looking at the indicator function of the DDT of η (Figure 6 for instance), we can observe a pattern that is very similar to a Sierpinski triangle, with some *additional parts*. What's more, the pattern of the DDT for a given value of n looks like a refined version of the pattern obtained for $n-1$, reminiscent of a version with more details that could have been obtained by another iteration of the shrinking and duplicating algorithm (see Figure 16 for an illustration of this phenomenon).



(a) $n = 5$. **(b)** $n = 6$.

Figure 16: Difference distribution table of $\eta : x \mapsto A_3(x) \oplus x$ for different n.

We just give a very informal idea of this relation:

- From Lemma 4, we have that every coefficient $\mathcal{D}_\eta^n(a, b)$ is a function of the coefficient of $\mathcal{D}_\eta^{n-1}(T(a), T(b))$ in the sense that if the function is non null (so if there is a motif in the considered area of the DDT at n), then it looks like the one we have at $n-1$. Morally, this is the same as shrinking the motif to use it to build next iteration (like in the Sierpinski triangle construction): the motif in the DDT at $n-1$ is copied in the DDT at n, but this one is twice bigger. Figure 17 depicts the *maximum* pattern we would obtain if all the motifs of $n-1$ would be present.
- The situation is made difficult by the *additional parts* (A and B in Figure 17) that

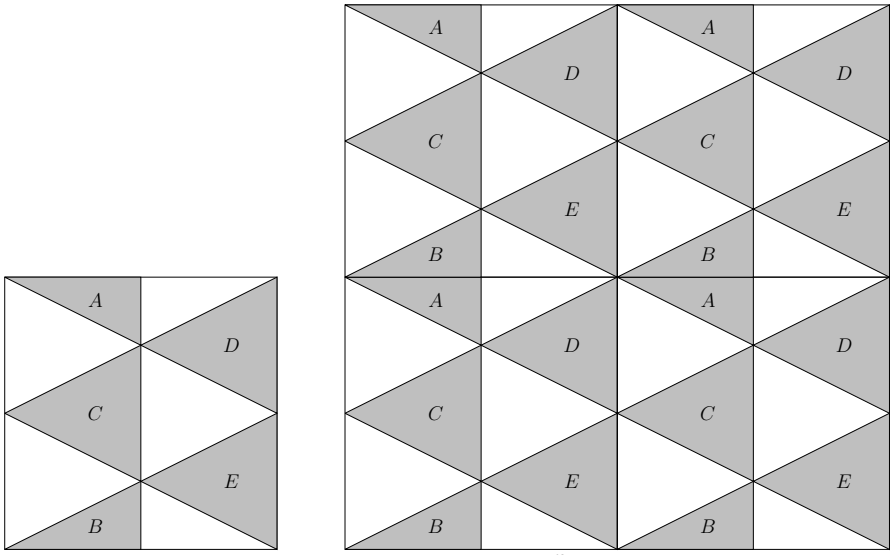


Figure 17: If we assume that the pattern of $\mathcal{D}^{\eta}_{n-1}$ is similar to the one on the left, then from Lemma 4 we can deduce that the *maximum* pattern of $\mathcal{D}^{\eta}_{n}$ is the one on the right.

mess with the pattern, and the obtained DDT does not look like a refined version of the smaller instance. Hopefully, as seen in Lemma 4, some parts are erased (namely a part of C in the left half, A and C in the right half), and thanks to that we obtain something that meets our expectations (see Figure 18).

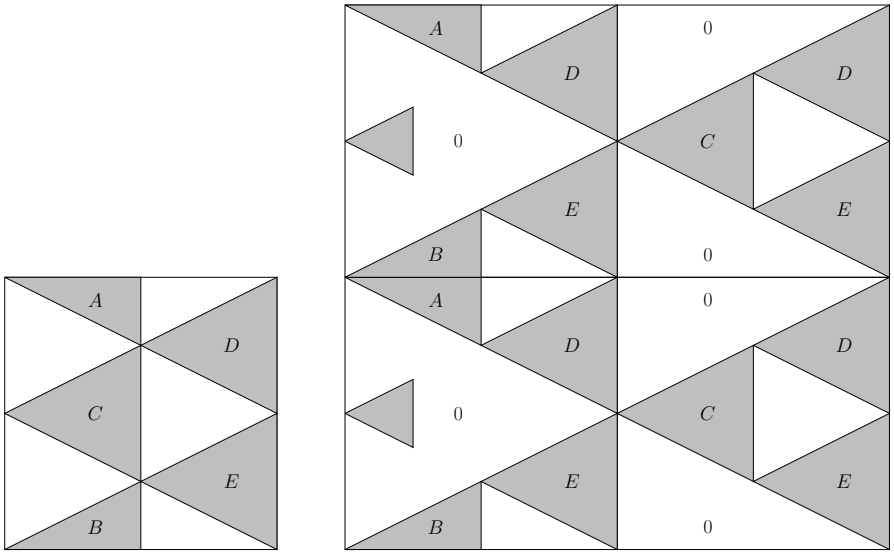


Figure 18: When we take into account the null parts, the pattern at n looks very similar to the one at $n-1$. The fractal structure is visible.

C.3 Proof of Lemma 2 (DDT bound)

Lemma 5. *All the following propositions are true for any a, b in $\mathbb{F}_2^n$ and any $i \geq 2$:*

1. $\lambda_i^0 \geq \lambda_i^1$,
2. *if $b_i \neq a_i$, then $\lambda_i^0 = \lambda_i^1$, and*

3. it holds that $\lambda_i^0 > \lambda_i^1$ if and only if all the following conditions are satisfied:

$$\begin{cases} \lambda_{i-1}^0 + \lambda_{i-1}^1 > 0 \\ b_i = a_i \\ b_{i-1} \neq a_{i-1} \quad or \quad \left(b_{i-1} = a_{i-1},\ b_{i-2} = a_{i-2} = a_{i-3}\right) . \end{cases}$$

Proof. If $b_i \neq a_i$ then $x \in \lambda_i^0$ is equivalent to $x \oplus a \in \lambda_i^1$, meaning that the first two propositions hold in this case. If $b_i = a_i$ then $\delta_i(a,b) \in \{0,1\}$. Let us consider those cases separately:

- If $\delta_i(a,b) = 1$, then $\lambda_i^1 = 0 < \lambda_i^0 = \lambda_{i-1}^0 + \lambda_{i-1}^1$, meaning that the strict inequality holds if $b_i = a_i$ and $b_{i-1} \neq a_{i-1}$.
- Otherwise, if $\delta_i(a,b) = 0$, then $\lambda_i^0 - \lambda_i^1 = 2 \times \lambda_{i-1}^1$, meaning that the general inequality also holds in this last case. The strict inequality holds if and only if $\lambda_{i-1}^1 > 0$. Since $\delta_i(a,b) = 0$, it holds that $\delta_{i-1}(a,b) \in \{0,1\}$. In this context, if $b_{i-2} \neq a_{i-2}$ then $\lambda_{i-1}^1 = 0$, meaning that the strict inequality does not hold. However, if $b_{i-2} = a_{i-2}$, then $\lambda_{i-1}^1 > 0$ is equivalent to $\lambda_{i-2}^0 \times (1 \oplus a_{i-2} \oplus a_{i-3}) > 0$, which is true if and only if $\lambda_{i-2}^0 > 0$ (which is indeed implied by $\lambda_{i-1}^0 + \lambda_{i-1}^1 > 0$) and if $a_{i-2} = a_{i-3}$.

We deduce the lemma. □

The following theorem tells us under which condition the sequence $\lambda_i^0 + \lambda_i^1$ increases.

Theorem 10. *Let $i \geq 2$ be some integer. Then the inequality:*

$$\lambda_i^0 + \lambda_i^1 > \lambda_{i-1}^0 + \lambda_{i-1}^1$$

holds iiif the conditions in Equation (8) and Equation (9) are satisfied, that is if:

$$a_{i-1} = b_{i-1}, \tag{8}$$

and if:

$$\begin{cases} a_i = b_i \\ a_{i-1} = a_{i-2}, \end{cases} \quad or \quad \begin{cases} a_i \neq b_i \\ a_{i-1} \neq a_{i-2} \\ b_{i-2} \neq a_{i-2} \ or \ b_{i-2} = a_{i-2}, b_{i-3} = a_{i-3} = a_{i-4}. \end{cases} . \tag{9}$$

Proof. If $\delta_i(a,b) = 1$ or $\delta_i(a,b) = 3$, then $\lambda_i^0 + \lambda_i^1 = \lambda_{i-1}^0 + \lambda_{i-1}^1$. Thus, in order for the sequence to increase, it is necessary that $\delta_i(a,b) \in \{0,2\}$ which is true if and only if $a_{i-1} = b_{i-1}$:

$$b_{i-1} \neq a_{i-1} \implies \left(\lambda_i^0 + \lambda_i^1 \leq \lambda_{i-1}^0 + \lambda_{i-1}^1\right) .$$

We then consider the cases $\delta_i(a,b) = 0$ and $\delta_i(a,b) = 2$ separately.

- $\delta_{\mathbf{i}}(\mathbf{a},\mathbf{b}) = \mathbf{0}$. As $\lambda_i^1 \leq \lambda_i^0$ (Lemma 5), $2\lambda_i^1 \leq \lambda_{i-1}^0 + \lambda_{i-1}^1$, so the case $a_{i-1} \neq a_{i-2}$ cannot lead to an increase. However, if $a_{i-1} = a_{i-2}$ then we have that $\lambda_i^0 + \lambda_i^1 = 2(\lambda_{i-1}^0 + \lambda_{i-1}^1) > (\lambda_{i-1}^0 + \lambda_{i-1}^1)$. Thus, if $\delta_i(a,b) = 0$, there is a strict increase in $\lambda_i^0 + \lambda_i^1$ if and only if $a_{i-1} = a_{i-2}$, so that:

$$a_{i-1} = b_{i-1},\ a_i = b_i \implies \left(\lambda_i^0 + \lambda_i^1 > \lambda_{i-1}^0 + \lambda_{i-1}^1 \Leftrightarrow a_{i-1} = a_{i-2}\right)$$

- $\delta_{\mathbf{i}}(\mathbf{a},\mathbf{b}) = \mathbf{2}$. In this case, $\lambda_i^0 + \lambda_i^1 = 2 \times \lambda_{i-1}^0 \times (a_{i-1} \oplus a_{i-2})$. Thus, the sequence is strictly increasing if and only if $a_{i-1} \neq a_{i-2}$ and $\lambda_{i-1}^0 > \lambda_{i-1}^1$. Using Lemma 5, we obtain that this is the case if and only if $b_{i-1} = a_{i-1}$ (which is already assumed to be true) and if either $b_{i-2} \neq a_{i-2}$ or $b_{i-2} = a_{i-2}, b_{i-3} = a_{i-3} = a_{i-4}$. Thus, if $\delta_i(a,b) = 2$, the sequence strictly increases if and only if $a_{i-1} \neq a_{i-2}$ and either $b_{i-2} \neq a_{i-2}$ or $b_{i-2} = a_{i-2}, b_{i-3} = a_{i-3} = a_{i-4}$.

The theorem follows. □

This very general theorem is a bit complicated to use in practice. The following corollary gives a simpler — though slightly coarser — interpretation.

Corollary 2. *All values of a, b such that $\lambda_i^0 + \lambda_i^1 > \lambda_{i-1}^0 + \lambda_{i-1}^1$ are such that:*

$$a_{i-1} = b_{i-1} \text{ and } a_i + a_{i-1} + a_{i-2} = b_i \ .$$

As a consequence, the coefficient $\mathcal{D}_A^n(a,b)$ can be bounded by:

$$\mathcal{D}_A^n(a,b) \leq 2^{1+\sum_{i=1}^{n-1} \nu_i(a,b)} \ ,$$

where:

$$\nu_i(a,b) = (1 \oplus b_{i-1} \oplus a_{i-1}) \times (1 \oplus b_i \oplus a_i \oplus a_{i-1} \oplus a_{i-2}),$$

and $\nu_1(a,b) = (1 \oplus b_0 \oplus a_0) \times (1 \oplus b_1 \oplus a_1 \oplus a_0)$.

Proof. We prove the theorem by induction over i using the following induction hypothesis for any $i \geq 1$:

$$\lambda_i^0 + \lambda_i^1 \leq 2^{1+\sum_{j=1}^{i} \nu_j(a,b)}.$$

Let us first consider $i = 1$. In this case, as given in Theorem 3, $\lambda_1^z = 2$ if and only if $a_0 \oplus a_1 \oplus b_1 = 0$ and $a_0 \oplus b_0 = 0$; it is equal to 0 otherwise. Thus, the total number of solutions is initialized with:

- 0 if $\nu_1(a,b) = 0$, in which case the bound always holds, or
- $\lambda_1^0 + \lambda_1^1 = 4 = 2^{1+\nu_1(a,b)}$ if $\nu_1(a,b) = 1$.

The hypothesis thus holds for $i = 1$. We now assume that the induction holds for some $i \geq 1$. According to Theorem 10, if $\lambda_i^0 + \lambda_i^1 > \lambda_{i-1}^0 + \lambda_{i-1}^1$ then $a_{i-1} = b_{i-1}$ and one of the following two conditions must hold:

- $a_i = b_i$ and $a_{i-1} = a_{i-2}$, or
- $a_i \neq b_i$ and $a_{i-1} \neq a_{i-2}$. We ignore the remainder of the condition in this case as we are only interested in *necessary* conditions.

As a consequence, if $\nu_i (=) 0$ then at least one of the conditions for $\lambda_i^0 + \lambda_i^1 > \lambda_{i-1}^0 + \lambda_{i-1}^1$ to be true fails to hold. As a consequence, $\lambda_i^0 + \lambda_i^1 = \lambda_{i-1}^0 + \lambda_{i-1}^1$ and the induction hypothesis still holds. On the other hand, $\lambda_i^0 + \lambda_i^1 \geq 2(\lambda_{i-1}^0 + \lambda_{i-1}^1)$ is always true, so that the induction hypothesis also holds in this case.

In the end, it is true that $\lambda_i^1 + \lambda_i^0 \leq 2^{1+\sum_{j=1}^{i} \nu_j(a,b)}$ for all $i \geq 1$ and in particular for $i = n-1$. Since $\mathcal{D}_A^n(a,b) = \lambda_{n-1}^1 + \lambda_{n-1}^0$, we obtain the lemma.

□

C.4 On the shape of the DDT

In this section, we give a rough estimate of the proportion of impossible difference transitions of the non-linear layer A_3, and show that many of them can be easily characterized by using an affine equivalent function.[11] We denote by R the function reversing the order of the bits in the binary decomposition, namely:

$$\begin{array}{rcl} R : \mathbb{F}_2^n & \to & \mathbb{F}_2^n \\ x = x_{n-1}, \cdots, x_2, x_1, x_0 & \mapsto & x_0, x_1, x_2, \cdots, x_{n-1} \end{array}$$

[11] Note that our reasoning can easily be generalized to A_α in general.

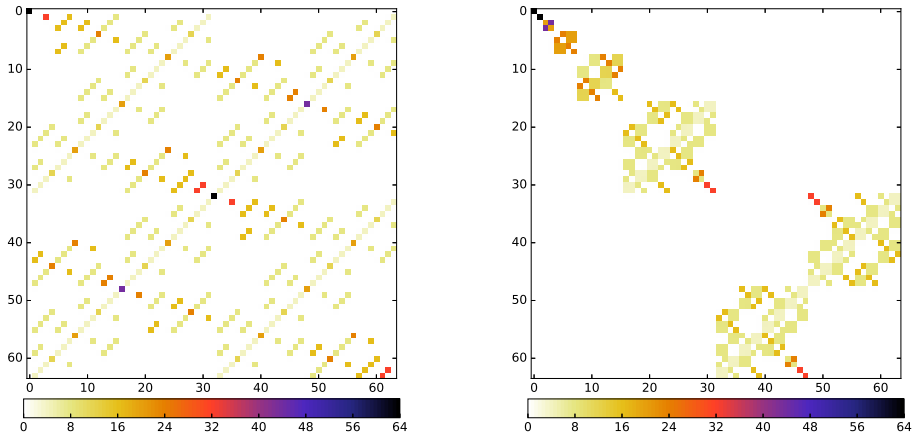


Figure 19: DDT of $A_3 : x \mapsto 3x$ (left) and of $x \mapsto R(3 \times R(x))$ (right), for $n = 6$.

By applying this function to both the input and output of A_3, we obtain a function that is affine equivalent and which DDT looks more organized: as depicted on Figure 19, it shows wide ranges of null coefficients grouped into squares.

The patterns we observe on the example of Figure 19 seem to indicate that for any n we have at least a fraction of $\frac{1}{8} + 2\sum_{i=1}^{n}(\frac{1}{4})^i$ impossible difference transitions. In the following, we show that they can be easily understood and that we can deduce a same amount of null coefficients in the DDT of A_3.

We denote by y the output of $R(A_3(R(x)))$. In the sum representation, this gives:

	x_0	...	x_{n-3}	x_{n-2}	x_{n-1}	$\times\alpha_0 = 1$
$\boxplus$	x_1	...	x_{n-2}	x_{n-1}	0	$\times\alpha_1 = 1$
$=$	y_0	...	y_{n-3}	y_{n-2}	y_{n-1}	

Which makes clear that the output bit number i of $R(A_3(R(x)))$ is linear in x_i and non-linear (in a broad sense) in x_j for $j > i$, property that we rewrite as:

$$y_i = x_i \oplus C_i(x_{i+1}, \cdots, x_n). \tag{10}$$

The following observations can be visualized on Figure 20 (for $n = 6$) but are valid for any value of n. The two hatched squares numbered 1 on Figure 20 each cover one fourth of the DDT and correspond to two types of impossible transitions: the down left one illustrates that a transition from a (truncated) input difference with the MSB set to 1 never leads to an output difference with the MSB equal to 0 while the right top square expresses the opposite situation (a difference with a null MSB never leads to an output difference with the MSB set to 1). This impossibility is clear from the expression of the MSB of the output: $y_{n-1} = x_{n-1}$. Consequently, half of the transitions are impossible. The other squares can be explained in a similar way just by referring to Equation (10):

- For two inputs that are equal on the first $i - 1$ MSB and different at position i, Equation (10) implies that the output difference in bit i is equal to 1 while the output difference in the bits of higher indices must be zero. Consequently, all the transitions from such an input difference to an output difference with the i first MSB equal to 0 are impossible. In Figure 20, this corresponds to the leftmost squares.

- Likewise, two inputs that do not differ on the i first MSB cannot lead to an output difference where the $i-1$ first MSB are null while MSB i is set to 1. This case is the symmetric of the previous one and defines squares of the same size.

For i varying from 1 to n, this defines a fraction of impossible transitions that is equal to $2\sum_{i=1}^{n}(\frac{1}{4})^i$. To conclude and obtain the announced lower bond we remark that an additional fraction of $\frac{1}{8}$ of the transitions are impossible. They are deduced from the linear equation $y_{n-2} = x_{n-2} \oplus x_{n-1}$ when $x_{n-1} = y_{n-1} = 1$ (squares denoted "a" on Figure 20). Given that A_3 is affine equivalent to $R(A_\alpha(R(x)))$, these observations translates into the same number of impossible differentials for A_3, with expressions that can be deduced.

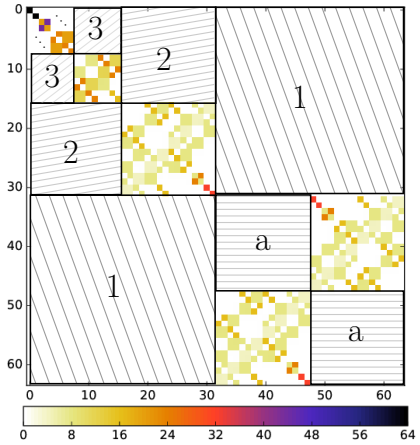


Figure 20: The DDT of $x \mapsto R(3 \times R(x))$ in $\mathbb{Z}/64\mathbb{Z}$. The hatched squares define impossible transitions that can be easily explained.

C.5 Proof of Theorem 5 (LAT formula)

C.5.1 The case of μ

We consider the function $\mu : x \mapsto (3 \times x \mod 2^n) \oplus x \oplus (2 \times x)$ which maps $\mathbb{F}_2^n$ to itself. It is extended affine equivalent to the multiplication by 3. Its coordinates are functions $\mu_i(x)$ such that:

$$\begin{cases} \mu_0(x) = 0, \\ \mu_1(x) = 0, \\ \mu_i(x) = x_{i-1}x_{i-2} + (x_{i-1} + x_{i-2})\mu_{i-1}(x), \end{cases}$$

where $(x, y, z) \mapsto xy + (x + y)z$ is the majority function. Note that μ_i does not depend on x_j for $j \geq i$, meaning that μ_i has all of its inputs in $\mathbb{F}_2^i$.

Let i be the position of the highest 1 in n. Then $\mathcal{W}_\mu(a, b) = 0$ unless $a_j = 0$ for all

$j \geq i$ as μ_i does not depend on x_j for $j \geq i$. We can write $\mathcal{W}_\mu(a)\, b$ as:

$$\begin{aligned}\mathcal{W}_\mu(a,b) &= \sum_{x\in\mathbb{F}_2^n} (-1)^{\langle a,x\rangle+\langle b,\mu(x)\rangle} \\ &= 2^{n-i}\sum_{y\in\mathbb{F}_2^i} (-1)^{x_0a_0+\ldots+a_{i-1}x_{i-1}\ +\ b_0\mu_0(x)+\ldots+b_{i-1}\mu_{i-1}(x)+\mu_i(x)} \\ &= 2^{n-i}\sum_{y\in\mathbb{F}_2^i} (-1)^{x_0a_0+\ldots+a_{i-1}x_{i-1}\ +\ b_0\mu_0(x)+\ldots+b_{i-1}\mu_{i-1}(x)+\left(x_{i-1}x_{i-2}+(x_{i-1}+x_{i-2})\mu_{i-1}(x)\right)} .\end{aligned}$$

We write $\mathbb{F}_2^i$ as the following direct sum:

$$\mathbb{F}_2^i \ = \ \{(z_0,z_1,...,z_{i-2},z_{i-2})\forall z\in\mathbb{F}_2^{i-1}\} \ \cup \ \{(z_0,z_1,...,z_{i-2},z_{i-2}+1)\forall z\in\mathbb{F}_2^{i-1}\} .$$

Furthermore, we remark that $\mu_{i-1}(x_0,...,x_{i-1})$ does depend on x_{i-1}. Thus, $\mathcal{W}_\mu(a,b)$ can be expressed as:

$$\begin{aligned}&2^{n-i}\sum_{z\in\mathbb{F}_2^{i-1}} (-1)^{z_0a_0+\ldots+a_{i-2}z_{i-2}+a_{i-1}z_{i-2}\ +\ b_0\mu_0(z)+\ldots+b_{i-1}\mu_{i-1}(z)+\left(z_{i-2}\right)} \\ &+2^{n-i}\sum_{z\in\mathbb{F}_2^{i-1}} (-1)^{z_0a_0+\ldots+a_{i-2}z_{i-2}+a_{i-1}(z_{i-2}+1)\ +\ b_0\mu_0(z)+\ldots+b_{i-1}\mu_{i-1}(z)+\left(\mu_{i-1}(z)\right)} ,\end{aligned}$$

which is equal to:

$$\begin{aligned}&2^{n-i}\sum_{z\in\mathbb{F}_2^{i-1}} (-1)^{z_0a_0+\ldots+a_{i-3}z_{i-3}+(a_{i-2}+a_{i-1}+1)z_{i-2}\ +\ b_0\mu_0(z)+\ldots+b_{i-1}\mu_{i-1}(z)} \\ &+2^{n-i}(-1)^{a_{i-1}}\sum_{z\in\mathbb{F}_2^{i-1}} (-1)^{z_0a_0+\ldots+(a_{i-2}+a_{i-1})z_{i-2}\ +\ b_0\mu_0(z)+\ldots+b_{i-2}\mu_{i-2}(z)+(b_{i-1}+1)\mu_{i-1}(z)} .\end{aligned}$$

We deduce the following lemma.

Lemma 6. *Recall that $L_i(a) = (a_0,...,a_{i-3},a_{i-2}+a_{i-1},0,..,0)$ and $T_i(a) = (a_0,...,a_{i-1},0,...,0)$ are linear functions. Then, for $a < 2^i$ and $b < 2^{i+1}$:*

$$\mathcal{W}_\mu(a,b) = \frac{\mathcal{W}_\mu\left(L(a)\oplus 2^{i-2}, T_i(b)\right)}{2} \ + \ (-1)^{a_{i-1}}\frac{\mathcal{W}_\mu\left(L(a), T_i(b)\oplus 2^{i-1}\right)}{2} .$$

C.5.2 Deducing an induction for the multiplication

We have found a recursive formula for the Walsh spectrum of $\mu : (3\times x)\oplus x\oplus(2\times x) = A(x)\oplus u(x)$, where $u(x) = x\oplus(2\times x)$ is a linear function. The Walsh coefficients of $A : x \mapsto 3\times x \mod 2^n$ are given by:

$$\begin{aligned}\mathcal{W}_A(a,b) &= \sum_{x\in\mathbb{F}_2^n} (-1)^{\langle a,x\rangle+\langle b,A(x)\rangle} \\ &= \sum_{x\in\mathbb{F}_2^n} (-1)^{\langle a,x\rangle+\langle b,\mu(x)\oplus u(x)\rangle} \\ &= \sum_{x\in\mathbb{F}_2^n} (-1)^{\langle a,x\rangle+\langle b,u(x)\rangle+\langle b,\mu(x)\rangle} \\ &= \sum_{x\in\mathbb{F}_2^n} (-1)^{\left\langle a+u^T(b),x\right\rangle\langle b,\mu(x)\oplus u(x)\rangle} \\ &= \mathcal{W}_\mu\left(a+u^T(b),b\right),\end{aligned}$$

where u^T is obtained by transposing the matrix representation of u, so that $u^T(x) = x \oplus (x/2)$. As before, we define i as the position of the 1 of highest weight in b. Then:

$$\begin{aligned}
&\mathcal{W}_A(a,b)\\
&= \mathcal{W}_\mu\left(a \oplus u^T(b), b\right)\\
&= \frac{\mathcal{W}_\mu\left(L\big(a \oplus u^T(b)\big) \oplus 2^{i-2}, T_i(b)\right)}{2}\\
&\quad + (-1)^{a_{i-1}+b_{i-1}+b_i} \frac{\mathcal{W}_\mu\left(L\big(a \oplus u^T(b)\big), T_i(b) \oplus 2^{i-1}\right)}{2}\\
&= \frac{\mathcal{W}_A\left(L\big(a \oplus u^T(b)\big) \oplus 2^{i-2} \oplus u^T(T_i(b)), T_i(b)\right)}{2}\\
&\quad + (-1)^{a_{i-1}+b_{i-1}+b_i} \frac{\mathcal{W}_A\left(L\big(a \oplus u^T(b)\big) \oplus u^T(T_i(b) \oplus 2^{i-1}), T_i(b) \oplus 2^{i-1}\right)}{2},
\end{aligned}$$

where $L(a \oplus u^T(b))$ is well defined because $a_i = b_i = 1$, so that the 1 of highest weight in $a \oplus u^T(b)$ is at most at position $i-1$. We simplify $L\big(a \oplus u^T(b)\big) \oplus u^T(T_i(b))$ into:

$$\begin{aligned}
L\big(a \oplus u^T(b)\big) \oplus u^T(T_i(b)) &= L(T_i(a)) \oplus L\big(T_i(u^T(b))\big) \oplus u^T(T_i(b))\\
&= L(T_i(a)) \oplus L\big(T_i(b) + b/2)\big) \oplus (T_i(b) + T_i(b)/2).
\end{aligned}$$

For $j \leq i-3$, $\langle e_j, L(T_i(b) + b/2)\rangle = b_j + b_{j+1} = \langle e_j, T_i(b) + T_i(b)/2\rangle$ and, for $j \geq i$, $\langle e_j, L(T_i(b) + b/2)\rangle = 0 = \langle e_j, T_i(b) + T_i(b)/2\rangle$. Thus, we only need to consider $i-2$ and $i-1$ as follows:

- For $i-2$: $\big\langle e_{i-2}, L\big(T_i(b) + b/2\big)\big\rangle = (b_{i-2} + b_{i-1}) + (b_{i-1} + b_i) = b_{i-2} + b_i$ and $\langle e_{i-2}, T_i(b) + T_i(b)/2\rangle = b_{i-2} + b_{i-1}$, so that their sum is equal to $b_i + b_{i-1} = 1 + b_{i-1}$.
- For $i-1$: $\big\langle e_{i-1}, L\big(T_i(b) + b/2\big)\big\rangle = 0$ and $\langle e_{i-1}, T_i(b) + T_i(b)/2\rangle = b_{i-1} + 0$, so that their sum is equal to b_{i-1}.

Furthermore, $u^T(2^{i-1}) = 2^{i-1} + 2^{i-2}$. We deduce Theorem 5, namely that $\mathcal{W}_A(a,b)$ can be computed as the following sum:

$$\begin{aligned}
&\frac{\mathcal{W}_A\left(L(T_i(a)) \oplus (2^{i-2} + 2^{i-1})b_{i-1}, T_i(b)\right)}{2} \quad +\\
&\quad (-1)^{a_{i-1}+b_{i-1}+1} \frac{\mathcal{W}_A\left(L(T_i(a)) \oplus (2^{i-2} + 2^{i-1})b_{i-1} \oplus 2^{i-1}, T_i(b) \oplus 2^{i-1}\right)}{2}.
\end{aligned}$$

Theorem 5 explains some visual patterns that we observe in the LAT of A_3 in Figure 7b. First, the non-zero coefficients are in n different squares with indices i, each defined as the set of points with coordinates (a,b) where $2^i \leq a < 2^{i+1}$ and $2^i \leq b < 2^{i+1}$. These correspond to the condition that $\mathcal{W}_A(a,b) = 0$ unless $2^i \leq a < 2^{i+1}$ when $2^i \leq b < 2^{i+1}$. Another implication of the theorem is that for a and b between 2^i and 2^{i+1} we have $|W_A(a,b)| = |W_A(a + 2^{i-1}, b + 2^{i-1})|$. This can be visualized on Figure 7b: if we cut into 4 equal squares any of the non-zero squares of the figure, the top left one is equal to the down right one, and the top right one has the same pattern as the down left one.