Toubkal: A Flexible and Efficient Hardware Isolation Module for Secure Lightweight Devices

Abderrahmane Sensaoui, David Hely, Oum-El-Kheir Aktouf

Toubkal: A Flexible and Efficient Hardware Isolation Module for Secure Lightweight Devices

Abderrahmane Sensaoui, David Hely, Oum-EL-Kheir Aktouf
Univ. Grenoble Alpes, Grenoble INP,LCIS*
**Institute of Engineering Univ. Grenoble Alpes*
Valence, France
{name.surname}@lcis.grenoble-inp.fr

Regular Paper

***Abstract*—Toubkal is a new hardware architecture which provides secure, efficient and flexible hardware isolation. It is a modular system that offers strong separation of different hardware modules within a system. Lightweight devices use mainly a Memory Protection Unit (MPU) to protect the memory and create an isolation architecture. However, the MPU offers only a memory control access for the software running on the system. This scheme does not prevent other hardware components from accessing system memories. Toubkal aims to enhance these MPU architectures by adding a new hardware layer to create different access environments for different hardware components.**

Toubkal has been designed in such a way that it can easily be adapted to the system needs in terms of security, safety and performances. It does not require any change in the existing hardware modules. We present a detailed description of the architecture, then we compare and discuss run-time, area overhead as well as security limitations using different policies and options. The first experimental hardware module increases between 0.08% and 8.5% a single core Rocket Chip cells area.

***Index Terms*—Isolation, Hardware Security, Trusted Computing**

I. Introduction

With the rise of lightweight devices use in healthcare, autonomous vehicles and smart factories, protection of lightweight devices must be strengthened while offering good performances. Memory isolation could be a good solution to add more resilience to an embedded system. Memory isolation is based on the principal of least privilege and privilege separation. It can create different execution contexts, so each program can be executed safely in its context. The industry proposes solutions for lightweight devices to achieve such security protection. One of the most known is the Arm MPU [1].

A lightweight device is a low-cost device that does not need complex memory management. Therefore, it lacks a Memory Management Unit (MMU), and its memory model is referred to as a flat memory model because its memory appears as one contiguous address space.

Modern lightweight embedded systems have become more complex and contain different hardware components with access to a system memories. The Arm MPU and the likes [2]–[4] are optional hardware components in some cores used to protect the memory in lightweight devices. These MPUs are less complex than the MMUs and have security and safety limitations. First, the privileged modes have access to the whole memory mapping which can be critical to the system security and dependability. Many researchers [5]–[7] have shown flaws in Operating Systems (OS) and how an attacker can escalate the privileges, and so, they can access all secrets stored in memories, change the MPU configuration, etc. Second, they only offer controlled memory accesses to the Control Processing Unit (CPU). Unfortunately, the CPU is not the only hardware component connected to memories. In this paper, we refer such components as Masters. Other Masters, such as the Direct Memory Access (DMA) peripheral, can access memory too. Neither MPUs nor MMUs can protect the memory from such accesses. Therefore, an additional layer of isolation is needed.

Interesting solutions [8]–[13] have been proposed by some academic papers and patents to add a special MMU for peripherals such as DMA. The so called In Out MMUs (IOMMU) or System MMU (SMMU) offer solutions similar to the traditional MMU. They add address translation and permissions for IO components.

However, these solutions are subject to a major limitation in this paper context. They are destined to high performance computer systems and require lots of cells area and memories as they use page tables and caches. Moreover, the IOMMUs and SMMUs are interfaced with one IO component (such as DMA). Therefore, if there are two IO components of which accesses must be controlled, the device will need two IOMMUs or SMMUs.

Another limitation they suffer from in this context, is the latency introduced into the devices. According to [14], they impose an extra performance penalty. This penalty is caused by the frequent mapping and un-mapping calls to create translation entries in the IO component address space.

Contrary to high performance systems, lightweight devices have strong power consumption and size constraints, and most of their applications are time critical. The use of any cited solution will add very high overheads, power consumption, cells area and run-time.

Others [11], [15]–[17] offer a DMA transfer filter where they define a safe region in the memory controller for the

DMA. The DMA can only access the defined region. However, this solution does not offer a real and complete separation. It limits the DMA access to a contiguous memory space.

While the need of a strong spacial separation becomes clear, we observe this level of isolation has been neglected in modern lightweight devices. We present Toubkal, a novel hardware module which overcomes previous work limitations to create low-cost and efficient spacial separation between Masters. Toubkal is an experimental parameterizable block with multiple-ready policies. This hardware block is responsible of controlling memory accesses of different IO components within the device to create a strong isolation between these components. Toubkal is compatible with existing Arm and RISC-V based Instruction Set Architectures (ISA) with an Advanced High-performance Bus (AHB) without changing the existing Masters.

The contributions of this paper are summarized as follows:

- We present an overview of MPUs limitations and the motivations behind designing Toubkal.
- We present Toubkal design and implementation. We implement Toubkal in Chisel3 [18] and offer a flexible design with multiple ready options to generate the right block for a given device.
- We evaluate Toubkal run-time performances, cells area and its own resiliency. We compare the impact of each option on Toubkal design and system security.

The rest of the paper is structured as follows. In section II, we give an example to motivate our solution and discuss limitations of the existing MPUs. Then, in section III we present Toubkal, a description of its major components, the way it communicates with the software, and different parameters and policies included in the package. We evaluate the overheads of Toubkal in section IV, execution time, cells area and security, and we draw a comparison of the impact of each parameter and policy on Toubkal design and behavior. In section V, we discuss related work. Finally, in section VI, we conclude this paper.

II. Toubkal Use case and Motivations

This section presents a brief Use Case and the requirements for an ideal protection system. Note that the use of Toubkal is not limited to this example. Consider a device with three Masters: a CPU, a DMA and a Cryptography Engine (CE). The CE can access the SRAM to read/write values for intermediate cryptographic operations. While usually the Keys are protected, an attacker can take advantage of a vulnerability like a buffer overflow or uses the DMA [19], [20] to access the intermediate cryptographic results. This way, the attacker can find the values of the keys from intermediate operations. Toubkal prevents the CPU and the DMA from reading the concerned memory region while the CE is processing its cryptographic operations.

To implement the protection system in such a way, Toubkal requires to be:

- ***Flexible:*** Toubkal has to be flexible. Flexibility in this paper means that the solution has to offer different strategies and options to achieve hardware separation. In the previous example, designers can opt for different policies depending on the existing tools and device constraints. Some would prefer to forbid all the time the region for all other masters. Others would prefer to forbid the access temporarily and then clean and re-use the same region for other operations by other Masters because of memory area constraints. For example, we can imagine a CPU writes in that region a plain text. This plain text will be read by the CE. Toubkal then forbids the access of the CPU and DMA to this region while the CE is cyphering the data, and then in the same region, the CE cleans intermediate values and writes the resulting encrypted data. Finally, Toubkal grants access again to the CPU to read the result.
- ***Efficient:*** Toubkal must be transparent but also small. Transparent, because Toubkal will catch all communications between the concerned Masters and memories, therefore the latency inside Toubkal must be negligible, and does not affect device timing. Small, because we are targeting principally lightweight embedded systems.
- ***Resilient:*** A protection domain guarantees a certain protection. Therefore, it must be itself resilient. Toubkal has to be configured at some point. Different approaches could be in place. Some approaches guarantee more resiliency than others, like for example hardware-only solutions are usually more resilient than hardware-software or software-only solutions. The chosen approach has to take in consideration the chosen policy and make sure not to weaken Toubkal.

Previous works [8]–[13], [15], [16] fail in *flexibility* and *efficiency* requirements. They add high overheads in terms of power consumption, hardware cells area and run-time execution and they control access to a specific IO component.

Previous MPUs such as ARM MPU, Intel EA-MPU, Rocket Chip Physical Memory Protection (PMP) and Mondrian are limited, and fail in one or more of these requirements. All the cited MPUs are interfaced with the core. This results in controlling memory accesses of the core alone. All other memory accesses are not controlled.

Another limitation of these MPUs is that the OSes, even if not trusted, they run in privileged mode, and thus have access to the whole memory space even MPUs memory mapped registers. [5], [6] depict weaknesses in OSes. Some of these vulnerabilities lead to privilege escalation. Therefore, an attacker can access any memory address, or, sometimes, completely turn off the protection of the MPU.

The aim of Toubkal is not to replace these MPUs but to enhance them. It offers a new layer of security and can work with any cited MPU. It is expected to overcome the above limitations while guaranteeing good performances and acceptable hardware cost.

Figure 1 illustrates a visual of the memory protection. Each column represents a protection domain. Here the protection domain is a given Master. Each Master can be prohibited or not from accessing memory regions. There can be different

policies to manage protection domains. Therefore, an ideal system would allow flexibility for both designers to generate the hardware block with the right size and policy, and developers to use it easily statically or dynamically.

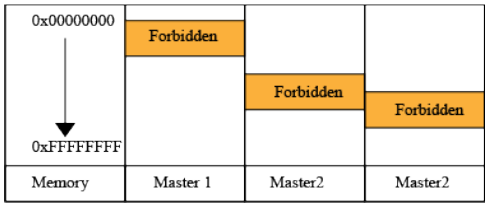

Fig. 1. A visual of how a single contiguous memory space is segmented between different Masters.

Toubkal is a parameterizable block written in Chisel3. It generates a block design in Verilog ready to include in a given system design. This block is responsible of creating a spacial separation between Masters. The size of Toubkal depends on the needs and policies. Designers can choose between three policies, *static*, *semi-static* and *dynamic* policies. They can also choose, among many other parameters, the address space (32-bits or 64-bits etc.) and the granularity of memory regions (size of each region is a power of two or a multiple of 32 bytes).

The next section presents in detail the design of Toubkal. We describe Toubkal major components and the different options and policies to choose to generate the tailored design.

III. Toubkal Design

The major challenge of Toubkal is to develop a flexible design and to keep it small with a negligible run-time overhead. This section starts with describing our initial analysis of the design, then presents our proof-of-concept implementation.

A. Toubkal Requirements

Toubkal needs to control all memory accesses of the concerned Masters. Therefore, the position choice of Toubkal is crucial. Communication between Masters and Memories is done via a communication bus. In this paper, we focus on systems with an Advanced High-performance Bus(AHB). The AHB [21] is a bus that connects hardware components with high bandwidth speed needs. Such components are Masters, Interconnect interfaces and Slaves. For example, CPUs, DMAs and Memories (external or internal) are connected with an AHB. The AHB looks like the perfect spot to catch communication between Masters and memories. Although, working in this spot requires Toubkal to control the access instantly, within the clock cycle. The design has to avoid any sequential processing.

The second requirement is the flexibility of the design. We suppose that depending on the application field, size and policies may vary. This is why Toubkal was designed as parameterizable design. In the design's top, designers choose the compatible configuration such as the address width, the granularity and the policy. Then they can generate the corresponding Verilog code to embed it in their design. To do so, we write our block in Chisel3, which is a new hardware description language that supports advanced hardware design. Chisel is embedded in Scala programming language and takes advantage of it to offer a high level hardware description language with concepts like object-oriented programming, polymorphism and functional programming.

The third requirement is resiliency. This point depends on the chosen policy. For a static policy, there is no need for Toubkal to include a software part. However, for a semi-static or dynamic policy, Toubkal must include a micro-code to manage configurations. This introduces a new attack surface to Toubkal. Section IV discusses in details this aspect.

B. Toubkal Major Components

Toubkal is composed mainly of the Master Look-aside Buffer (MLB) the uCode block, in addition to some logic for the look-up and match, security checks and Toubkal configuration (see figure 2). In this work, Toubkal is interfaced with the AHB but it can be positioned in any other position that catches communication between Masters and memories. The abstraction layer aims to facilitate the integration of Toubkal in different systems and positions.

Toubkal is composed of uCode which is responsible of memory mapping all Toubkal registers and configuring Toubkal. uCode can be configured from a software called micro-code. The MLB is a set of registers to store regions configurations or look-up&match some address. This paper presents and discusses two different implementations of the MLB. uCode is connected to MLB registers to store regions configurations. And then, there is the abstraction layer. This layer is customizable and aims to facilitate the integration of Toubkal in different systems and spots.

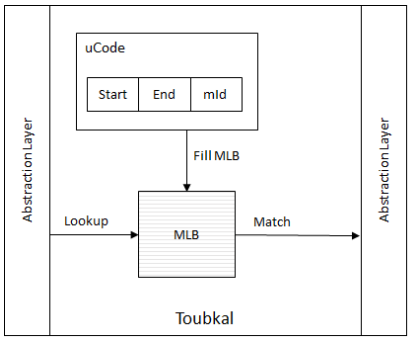

Fig. 2. Major components of hardware block of Toubkal: The uCode module contains information to recognize the micro-code which is responsible of configuring Toubkal. This module is linked to the MLB. The MLB is a set of registers to store regions configurations. The abstraction layer is customizable and aims to facilitate the integration of Toubkal in different systems and at different spots.

C. Master Look-aside Buffer

The MLB is designed to store memory regions configurations, it looks-up&matches addresses. To store configurations, Toubkal uses registers, which are costly, but guarantee rapid access. Toubkal proposes different ways to store configurations. As shown in figure 3, designers can choose the number

of memory region slots, the granularity, the address width, the number of concerned Masters and the method to attribute slots to Masters.

Configuration Method 1: UMLB

Master Id	Start Address	Size/End	Lock	Valid
0				
1				
2				
log2Ceil(N) bit	27 bits	5/27 bits	1 bit	1 bit

Configuration Method 2: SMLB

Master Id	Start Address	Size/End	Lock
0b001			
0b101			
0b110			
N bit	27 bits	5/27 bits	1 bit

Master 2 Master 0 Master 1

Fig. 3. MLB slots: The figure presents the different ways to store configurations in the MLB for a 32 bits address space. N represents the number of masters. The first method, called UMLB (U for Unique) stores one configuration per Master per slot, while the second method, called SMLB (S for Shared), stores one configuration per slot but for multiple masters. In the SMLB, there is no need for a valid bit as each Master has a corresponding bit in the first column from the left. The value 0 means the configuration is not valid for that master and vice versa. The third column from the left refers to size in case of coarse-grained granularity, and to end address in case of fine-grained one.

The first important decision to make in Toubkal is to choose between configuring memory regions **white-list** or memory regions **black-list**. This choice is strategic and depends on which case would lead to fewer slots to configure. We added this option as we believe for isolation at this level, contrary to MPUs and MMUs, it is more interesting to create a black-list rather than a white-list. It is more likely to prohibit a memory region rather than allowing it. Indeed, the way of thinking at this level is different compared to the application level. Thus, in the rest of the paper we will be more talking about prohibiting memory regions to Masters rather than allowing them.

Toubkal offers two ways to organize slots (see figure 3). The first method is called Unique MLB (*UMLB*) because each slot is unique to a Master, and the second method is called Shared MLB (*SMLB*) because a slot can be shared between multiple Masters. The *UMLB* consists in storing in each slot a memory region configuration for a specific Master. While the *SMLB* consists in storing in each slot a memory region configuration for multiple Masters. Each way has its advantages and its drawbacks, and each is more suitable to specific use cases. While the *UMLB* requires less logic to match or insert the Master Id, the *SMLB* can require fewer slots (therefore fewer cells area) in case some Masters share some common permissions. In the SMLB, there is a bit for each Master. When the bit is up, it means that the configuration is valid for the corresponding Master and vice-versa.

Concerning the number of slots, Toubkal accepts up to 16 slots. Because of the high cost of registers, we limited the number of slots to 16 slots. Designers can choose between 4, 8, 12 or 16 slots. The size of Toubkal grows proportionally to this number. Section IV-B shows how the area is impacted by the number of slots.

The other parameter is the granularity of Toubkal. This paper proposes two granularities. For fine-grained granularity, Toubkal proposes regions multiple to 32 bytes and the region address start is aligned to 32 bytes. And for coarse-grained granularity, Toubkal proposes regions power of two and the region address start is aligned to its size. The second granularity is more constraining in comparison to the first one. However, as figure 4 shows, the coarse-grained requires less logic than the fine-grained to match addresses.

To configure memory regions in Toubkal, MLB registers can be hard-coded for more resiliency and less flexibility, or memory mapped and then the uCode is responsible in configuring them during software execution for more flexibility and dynamism. Making Toubkal accessible from the running software adds its attack surface. We discuss this point in the next section.

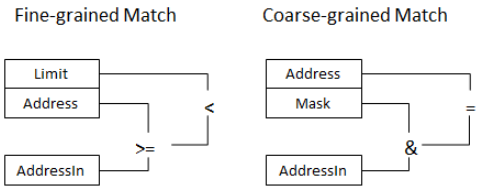

Fig. 4. A fine grained match is more complex than a coarse grained match. The second one compares the address with the base address and the limit address, while the first one compares the logical combination of the address and the mask corresponding to the size of the region with the base address.

D. uCode and Toubkal configuration

The uCode permits to a certain part of the software (micro-code) to configure Toubkal. uCode contains registers to store the start address and the end address of the micro-code in the memory, as well as the Id of the Master allowed to run this code. Once Toubkal is enabled, these registers have to be set and then locked until system reset. The micro-code is the only software part that can configure Toubkal MLB. Toubkal saves the Program Counter (PC) and whenever a read/write of MLB registers occurs, it checks if the PC is inside the micro-code region.

To make sure the micro-code is running in privileged mode without changing the CPU, we propose to map the MLB registers to the system memory region. In our case, as we are working on RISC-V and Arm cores, there are memory spaces [22], [23] for each core that can only be accessed by program code running in privileged mode. We take advantage of these spaces to memory map the uCode and MLB registers.

Toubkal is configurable either once or through a unique trusted Master. For the second case, uCode is mandatory, and

is the only one responsible in configuring Toubkal during run-time. Each slot can be configured and locked independently. If a slot is set locked, it can only be reconfigured after the system reset. If the slot is not locked, then it can be reconfigured as many time as needed by the micro-code.

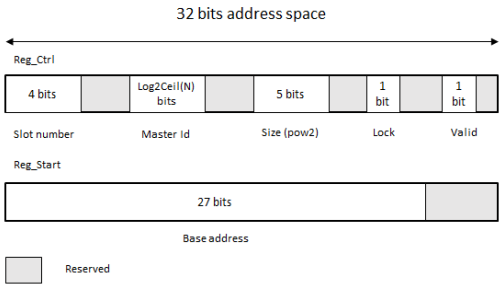

Fig. 5. Toubkal example of memory mapped registers. The example corresponds to a 32 bits address space with the UMLB configuration.

Toubkal is controlled by its own set of memory-mapped registers. Figure 5 shows an example of memory-mapped registers. The base address for these registers varies depending on which core is used. For Arm for example, the base address has to be between 0xE0000000 and 0xE003FFFF [22] to make sure that they are accessible only by code running in privileged mode. Toubkal mapped registers can change in terms of the chosen options.

E. Toubkal Policies

We put in place different policies to generate different designs of Toubkal. These choices impact the hardware area cost. Section IV compares between all the options. We implement three policies, *static*, *semi-static* and *dynamic*.

The *static* policy is a security policy that focuses on resiliency rather than flexibility. In this policy, memory regions configurations are hard-coded, or loaded using the Read Only Memory (ROM) or a One Time Programmable (OTP).

The *semi-static* policy is a little more flexible than the first one. Like the *static* policy, some configurations can be fixed from the beginning, but it offers the possibility to add other configurations during execution. To do so, we add a uCode that has to be run in privileged mode and we memory map some registers. These registers are only accessible from privileged mode. The uCode must be a predefined memory region, which means that outside this region Toubkal will not configure itself. Once a configuration is added, it cannot be modified or removed, although, this policy works only with the SMLB configuration, therefore, it is possible to validate or invalidate the configuration for each Master by flipping the corresponding bit. This can be useful in the use case presented in section II. It is also possible to lock a configuration so no one can change it until system reset.

The *dynamic* policy is the most flexible but the less secure of the three policies. From the uCode, it is possible to add, remove or change all configurations. The uCode runs under the same condition as in the *semi-static* policy. There is also a lock bit to lock a configuration so it cannot be modified until system reset.

In the next section, we evaluate Toubkal, compare and discuss the differences between each policy in terms of performances, cells area and security.

IV. Toubkal Evaluation

Toubkal proposes different options and policies. Some options and policies offer more flexibility to developers than other. For example, fine grained option is more flexible and user-friendly for developers than the coarse grained one. But it comes at a cost. The same observation can be done for policies, the more a policy is flexible, the more the surface area of Toubkal grows.

Policies can also impact the device security. For example, the dynamic and semi-static policies introduce the micro-code which is a software run to configure Toubkal. Adding a software exposes Toubkal to software attacks such as Return Oriented Programing (ROP) attacks. While the static one does not need a software, it eliminates the risk of such attacks.

We realized Toubkal design in Chisel3. Toubkal is written in around 480 Chisel3 lines of code and generates up to 3400 optimized and synthesizable Verilog lines of code. The generated Verilog codes are then synthesized. Synthesizing is compiling the Verilog code into a set of gates and wires to build a block of an Application-Specific Integrated Circuit (ASIC).

In this section we evaluate Toubkal from three different aspects: performances, cells area and security. We compare for each aspect the impact of each option and policy.

A. Run-time evaluation

To evaluate the run-time behavior, we evaluate the latency inside Toubkal. As seen above, we have two main parts in Toubkal, the uCode and the MLB. The uCode is responsible of decoding the data sent from the micro-code and then stores it into the right MLB slot. This operation is done in one clock cycle. Concerning the look-up&match, the result is instantly obtained for both granularities. Toubkal intercepts communication in the AHB. Therefore, it requires a negligible latency. The configuration of MLB is considered as read/write in a Static Random Access Memory (SRAM). The address checking must be done in less than a clock cycle. Toubkal uses parallel look-up. The implementation for the coarse granularity is less complex than the fine granularity. In the first one, only a match of a logical combination of the address and a mask is needed. While the second one needs more logical elements as we compare the address with the base and the limit.

To make sure Toubkal runs in less than a clock cycle, we synthesize the design with temporal constraints. The temporal constraints consist in delaying the Input and Output ports. Defining the input delay port value means that the data would reach the input ports after the defined value. And defining the output delay value means that the data should be present on the output port before the specified clock edge.

We synthesize our design in different clock periods up to 1GHz for a 90nm Low Power technology. The objective is to determine the fastest clock speed within the defined temporal constraints, which are 0.4 of clock period as an input delay and as output delay, by determining delay of the critical path inside Toubkal. If the delay is positive, then Toubkal assures data at the output ports in time. Otherwise, there are cases, e.g. the critical path, where Toubkal needs more than a clock cycle to deliver data.

The result of synthesis differs from a policy to another. For the static policy, even at 1GHz, the delay is always positive. For the semi-static and dynamic policies, the delay equals 0 at 200 MHz, and for more than 200MHz, the delay is negative. The critical path where the delay is negative matches the path done to configure a slot in the MLB. Assuming that Toubkal needs more than a clock cycle to configure the whole slot, we can add a timing exception for uCode module. During Toubkal configuration, the communication might be blocked for two or three cycles for each slot, depending on the granularity of the design as seen in section III.

As we target lightweight devices, a positive delay at 200MHz and less for 90nm Low Power technology is a good result. For the rest of the paper, all our syntheses are done in 100MHz for a 90nm Low Power technology.

B. Cells area evaluation

In this section, we evaluate the cells area of Toubkal targeting a 90nm Low Power technology for a clock frequency of 100MHz and compare between the different options and policies. In this experiment, tests are limited to a 32 bit address space. Toubkal surface is stated in Gates Equivalents (GE). kGE refers to thousands of GE. A GE is retrieved from the surface of a NAND2 gate.

1) Comparison between configuration methods: Section III-C presents two different configuration methods to store Masters Id, UMLB where the slot is valid for one Master and the SMLB where the slot is valid for multiple Masters. Table I shows the impact of each method on Toubkal area. In the UMLB, for n Masters, Toubkal needs *log2Ceil(n)* bits to store all possible values of Masters Ids. While in the SMLB, Toubkal needs n bits because it flips the p bit to activate a region for Master number p (see section III-C). However, n is greater than or equals to *log2Ceil(n)* for n strictly positive, therefore Toubkal needs more registers for SMLB comparing to UMLB. Here, the test was done for 2 Masters, and *log2Ceil(2)* equals 1.

Registers are not the only thing impacted. The combinational logic is impacted too. In the SMLB, Toubkal needs more logic than in the UMLB. To flip the p bit is more complex than just writing the value p in a register. The same is correct for the look-up&match. Toubkal has to look if the bit number p is up or not.

The choice between the two possibilities is strategic. In case there are more common regions between different Masters, it is more interesting to use the SMLB rather than the UMLB. This can save many slots. The use case presented in section II would work perfectly with the SMLB. Three Masters CPU, DMA and CE share the same region. Here the SMLB saves two slots. But in case Masters do not share regions, the UMLB is suitable.

TABLE I
TOUBKAL CELLS AREA FOR A DYNAMIC POLICY. (IN KGE)

	4 slots	8 slots	12 slots	16 slots
UMLB	3,36	6,21	9,10	12,04
SMLB	3,35	6,31	9,19	12,11

TABLE II
TOUBKAL CELLS AREA FOR A DYNAMIC POLICY. (IN KGE)

	4 slots	8 slots	12 slots	16 slots
Coarse granularity	3,36	6,21	9,10	12,04
Fine granularity	4,84	8,90	13,49	17,76

2) Comparison between coarse grained and fine grained granularities: Table II shows results of different syntheses for coarse and fine granularities. The fine grained option requires more area than the coarse grained one. The gap between the two is proportional to the number of available slots. This is due to two factors that were discussed in section III-C. The first factor is the growing size of the MLB registers. For the coarse granularity, the size of a region is a power of two and is on 5 bits. While for the fine grained granularity, instead of the size of a region we have the limit of the region which is on 27 bits for a 32 bit address space and an alignment to 32 bytes. Therefore for each slot, there are 22 additional bits in the fine grained granularity. Added to this the logic behind the look-up&match. The implementation of the parallel look-up in this case is more complex and requires more logical elements to correctly answer to the timing constraints.

Nevertheless, the coarse grained option is constrained. The user needs to configure regions with a size power of two, and the base address is aligned with the size. While the fine grained requires only that the size is multiple of 32 bytes and the base address is aligned to 32 bytes.

At this level of isolation, we believe that the coarse grained option would not be very constraining for developers. Depending on the device resources, some designers would opt for the coarse grained option to save some hardware area, other would opt for the fine grained one for more user-friendliness and less engineering. However, such advantage comes with a surface cost.

3) Comparison between policies: Here, we compare between the three policies, static, semi-static and dynamic. We fix granularity into the coarse one, configuration method into the second method, then for each policy we vary the number of slots.

Table III shows the results of our syntheses. Static policy results catch our eyes immediately because of the huge gap between this policy and the others. In fact, static policy results are normal. As seen in section III-E, the MLB configuration is

TABLE III
TOUBKAL CELLS AREA FOR A DIFFERENT POLICIES . (IN KGE)

	4 slots	8 slots	12 slots	16 slots
Static	0,15	0,16	0,18	0,21
Semi-Static	3,45	6,45	9,49	12,31
Dynamic	3,35	6,31	9,19	12,11

TABLE IV
TOUBKAL CELLS AREA FOR SEMI-STATIC POLICY. (IN KGE)

Hard-coded slots	4 slots	8 slots
0 slots	3,45	6,45
1 slots	2,74	5,85
2 slots	2,01	5,16

hard-coded. Therefore, when Chisel3 generates the optimized and synthesizable Verilog code, it transforms all registers into simple wires and assigns values. And there is less and less combinational logic in Toubkal. Chisel3 compiler also removes all redundant values. For example if some regions have the same size value, it will just assign the value once, and use it for all the concerned regions while looking-up&matching.

Meanwhile, the gap is smaller between the semi-static and dynamic policies. The semi-static policy cells area is a little bigger than the dynamic one because Toubkal needs more combinational logic. In semi-static policy, slots cannot be fully modified. Developers can only modify the Masters Ids registers, and activate or deactivate the configuration for a specific Master.

For the semi-static policy, some slots can be hard-coded. For each hard-coded slot, the reduction of Toubkal cells area is not negligible at all. Table IV shows some results. The size of Toubkal is reduced proportionally to the number of hard-coded slots. This reminds us of the static policy. The hard-coded slots are not stored in registers anymore. Hence, the surface area is reduced.

C. Security evaluation

In this part, we discuss how Toubkal helps protecting the device from a variety of attacks as well as its security limitations.

1) Direct Memory Access (DMA): Until today, the DMA has been a very serious weakness for many devices that make use of it. For example, the Nintendo switch attack was performed [24], [25] by taking advantage of a buffer overflow vulnerability found in the USB stack. This allowed an attacker, using the DMA, to extract secrets from memory or to execute arbitrary code. Toubkal can stop the DMA from accessing certain memory regions.

2) Heterogeneous systems: Considering a system with two different cores. An attacker can make use of a vulnerability to access data processed by the other core. An MMU can be a solution here, but it has a real impact on time critical devices. Toubkal can create a real separation between the two cores so each core cannot access the other core memory space without impacting run-time execution.

3) Hardware interrupts limitation: An application code can be interrupted by a certain interrupt. The CPU elevates the privilege. An attacker could take advantage of a vulnerability and attempt a code reuse attack to divert the control path and execute the micro-code. Therefore, the attacker can modify non locked regions. This is a current limitation that concerns the semi-static and dynamic policies.

To mitigate this weakness, some security development habits or the use of a security hypervisor such as uVisor [6] and the likes [26]–[28] could make it harder for an attacker to break into the micro-code. One way to make sure the micro-code is run according to the initial program flow is to add a Control Flow Integer (CFI). The CFI will not only make sure that the micro-code is run properly but the whole device. There are lots of works in this area, but there is still room for improvement, especially for lightweight devices.

V. RELATED WORK

In this section we compare Toubkal to related existing work. There has been several works, academic and industrial, that have explored this problematic. Some works have tried to solve the issue for all Masters. Others just tried to solve the DMA issue.

In the first category, such works are IOMMUs and the IOMPU. Several works have been published about the IOMMU. For example, [8] presents a solution that offers an MMU-like for peripherals. Therefore, addresses for components such as DMA are translated too and controlled. In this work, the IOMMU is embodied in a processor which connects the IOMMU with memories and peripherals using bridges. The IOMPU [12] takes advantage of the Non-Transparent Bridge (NTB) which isolates two hosts or memory domains yet allowing the exchange of data. Using the NTB, IOMPU can offer spacial isolation for I/O devices.

Arm proposes System Memory Management Unit [13] (SMMU) to complement its MMU. The SMMU is Arm version of the IOMMU. It works based on the same principles of the IOMMU.

The IOMMUs, the IOMPUs and the likes [8]–[11] intercept IO communications and perform address translation from device addresses to machine's physical addresses. They are similar to a classic MMU. They have page tables to translate addresses and caches frequent entries in a Translation Look-aside Buffer (TLB). However, this is unsuitable with lightweight devices as they have resources constraints and are time critical.

Moreover, the above solutions are only interfaced with the concerned components. For example, if there are two IO components, the device will need two IOMMUs or SMMUs to control accesses of each component. In the mean time, Toubkal controls accesses of multiple IO components.

[29] offers a solution to protect communication between a secure Master and a secure Slave. They propose to add two hardware blocks, one in front of the secure Master and

the other in front of the Secure Slave. We call the block in front of the secure Master MB and the block in front of the secure Slave SB. The SB defines a protected region. When the secure Master tries to access the protected region, the MB sends a signal to the SB to let the secure Master access the protected memory. If the SB does not receive a signal from an MB, it blocks the access for the protected memory region. This is limited comparing to Toubkal. First, each concerned Master and for each Slave need MBs and SBs respectively, and for each Slave, the SB protects only one memory space. Second, once the Master has an MB, it can access all the protected memory regions, while Toubkal offers a real separation between all Masters.

In the second category, there are works such as Intel SGX [16], Sanctum [15]. The two solutions tackle the DMA problem. Intel SGX for example offers a DMA filtering based on black-list approach. This joins Toubkal idea in proposing a black-list, except that Toubkal offers a black-list for different IO Masters and not only the DMA. Sanctum proposes a simple DMA filtering where there are two additional registers in the DMA arbiter (memory controller) to define the start address of a DMA safe region and end address.

TrustZone by Arm can also join the list of related work, as their *Secure/NonSecure* bit propagates to all hardware modules. TrustZone proposes a binary protection where there are two worlds, secure world and non secure world. We believe that Security in general is not binary, especially in isolation. A flaw in the secure world can lead to serious damages. Added to this, this paper [30] presents relevant attack scenarios in heterogeneous systems where a third-party hardware module exploits some failures of the TrustZone technology. First, Toubkal offers different protection domains for each Master. And second, Toubkal can help mitigate these attack scenarios. It provides memory protection to isolate Masters from each other and limit their address space.

VI. Conclusion

In this paper we emphasized on the need of a strong hardware isolation in lightweight devices. Traditional MPUs offers limited isolation that will become dangerous when an attacker escalates a privilege or succeeds in making use of a peripheral that accesses memory. Related work requires lots of resources and processing time. Furthermore, most of them are limited to DMA issue and need to be cloned for each concerned Master. For this purpose, we presented Toubkal, a first experimental hardware isolation module for lightweight devices. Toubkal is written in Chisel3 and provides a Verilog generator of hardware isolation modules. It offers many options and policies to generate the right design. Regarding the evaluation, Toubkal achieves a spacial separation with inexistent run-time overhead and tailored hardware area, from 0.08% to 8.5% of a single core Rocket Chip total area.

References

[1] ARM, "Arm the architecture for the digital world," https://www.arm.com/, 2015.

[2] O. Stecklina, P. Langendörfer, and H. Menzel, "Design of a tailor-made memory protection unit for low power microcontrollers," *2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pp. 225–231, 2013.

[3] E. Witchel, "Mondriaan memory protection," 2004.

[4] P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan, "Trustlite: a security architecture for tiny embedded devices," in *EuroSys*, 2014.

[5] "Freertos tcp/ip stack vulnerabilities put a wide range of devices at risk of compromise: From smart homes to critical infrastructure systems," 2018.

[6] ARM, "uvisor," https://github.com/armmbed/uvisor, 2015.

[7] L. Szekeres, M. Payer, T. Wei, and D. X. Song, "Sok: Eternal war in memory," *2013 IEEE Symposium on Security and Privacy*, pp. 48–62, 2013.

[8] M. H. et al., "Direct memory access (dma) address translation in an input/output memory management unit (iommu)," no. US7809923B2, 2010.

[9] A. K. et al., "Input/output memory management unit with protection mode for preventing memory access by i/o devices," no. US8631212, 2014.

[10] D. Münch, M. Paulitsch, O. Hanka, and A. Herkersdorf, "Mpiov: Scaling hardware-based i/o virtualization for mixed-criticality embedded real-time systems using non transparent bridges to (multi-core) multi-processor systems," *2015 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pp. 579–584, 2015.

[11] D. Evtyushkin, J. Elwell, M. Ozsoy, D. S. Ponomarev, N. A. Ghazaleh, and R. Riley, "Flexible hardware-managed isolated execution: Architecture, software support and applications," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, pp. 437–451, 2018.

[12] D. Münch, M. Paulitsch, and A. Herkersdorf, "Iompu: Spatial separation for hardware-based i/o virtualization for mixed-criticality embedded real-time systems using non-transparent bridges," *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, pp. 1037–1044, 2015.

[13] ARM, "Arm system memory management unit architecture specification," 2016.

[14] M. Ben-Yehuda and K. Rister, "The price of safety : Evaluating iommu performance," 2007.

[15] V. Costan, I. A. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *USENIX Security Symposium*, 2016.

[16] V. Costan and S. Devadas, "Intel sgx explained," *IACR Cryptology ePrint Archive*, vol. 2016, p. 86, 2016.

[17] A. A. Levy, B. Campbell, B. Ghena, D. B. Giffin, P. Pannuto, P. Dutta, and P. Levis, "Multiprogramming a 64kb computer safely and efficiently," in *SOSP*, 2017.

[18] J. Bachrach, H. Vo, B. C. Richards, Y. Lee, A. Waterman, R. Avizienis, J. Wawrzynek, and K. Asanovic, "Chisel: Constructing hardware in a scala embedded language," *DAC Design Automation Conference 2012*, pp. 1212–1221, 2012.

[19] P. Stewin and I. Bystrov, "Understanding dma malware," in *DIMVA*, 2012.

[20] F. L. Sang, V. Nicomette, and Y. Deswarte, "I/o attacks in intel pc-based architectures and countermeasures," *2011 First SysSec Workshop*, pp. 19–26, 2011.

[21] ARM, "Arm amba 5 ahb protocol specification," 2015.

[22] ——, "Cortex-m3 devices generic user guide," 2010.

[23] SiFive, "Sifive e31 core complex manual," 2017.

[24] M. S. Kate Temkin, "Fusee gelee exploit," https://nvd.nist.gov/vuln/detail/CVE-2018-6242, 2018.

[25] "Shofel2 exploit," https://github.com/fail0verflow/shofel2, 2018.

[26] F. F. Brasser, B. E. Mahjoub, A.-R. Sadeghi, C. Wachsmann, and P. Koeberl, "Tytan: Tiny trust anchor for tiny devices," *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2015.

[27] Tock, "Tockos," https://github.com/helena-project/tock, 2015.

[28] eChronos, "echronos," https://ts.data61.csiro.au/projects/TS/echronos/, 2018.

[29] GreenIPCore, "Amba-ahb security," 2018.

[30] E. M. Benhani, C. Marchand, A. Aubert, and L. Bossuet, "On the security evaluation of the arm trustzone extension in a heterogeneous soc," *2017 30th IEEE International System-on-Chip Conference (SOCC)*, pp. 108–113, 2017.