Advanced Covert-Channels in Modern SoCs

Lilian Bossuet, Carlos Andres Lara-Nino

Advanced Covert-Channels in Modern SoCs

Lilian BOSSUET and Carlos Andres LARA-NINO
Université Jean Monnet Saint-Étienne, CNRS, Institut d'Optique Graduate School,
Laboratoire Hubert Curien UMR 5516, F-42023,
SAINT-ETIENNE, France.
carlos.lara@univ-st-etienne.fr

Abstract

Modern SoCs can be protected against software attacks under the paradigm of secure enclaves, which are built employing technologies like ARM TrustZone. These protections are meant to enforce access policies so that the interaction between untrusted/trusted applications and hardware components is limited. However, the possibility of creating covert channels within the SoC threatens these isolation models. Among other approaches, it has been shown that it is possible to create covert channels by exploiting the frequency-modulation technology available in these platforms. These attacks are devastating, since digital circuits generally use a single power distribution network. This provides the medium for the implementation of such covert-channels. Heterogeneous SoCs are particularly vulnerable in this regard, as under these platforms multiple operating ecosystems coalesce. The problem is exacerbated because these systems have become more prevalent with each new generation. In this paper, we explore the implementation of frequency-based covert-channels using Zynq Ultrascale+ SoCs as case study. Our findings demonstrate that it is possible to exchange information between Linux-based applications, bare metal applications, and hardware modules, achieving transmission rates up to 750 Kbps.

Keywords: Covert channels, Frequency modulation, Multi-cluster, SoC-FPGAs, Zynq ultrascale+.

Please cite as:

```
@InProceedings{BL23,
title = {{Advanced Covert-Channels in Modern SoCs}},
author = {Bossuet, Lilian and Lara-Nino, Carlos Andres},
booktitle = {Proceedings of the IEEE International Symposium on Hardware Oriented Security
and Trust (HOST)},
pages = {80--88},
year = {2023},
publisher = {IEEE},
address = {San Jose CA, USA},
doi = {10.1109/HOST55118.2023.10133626},
isbn = {979-8-3503-0062-8}}
```

1 Introduction

A covert channel is, as the name suggest, an information pathway which is not evident to those unaware of the nature of the system. We might be familiar with a typical example found in classical media: morse code. In that case, the parties employ blinking patterns, quiet sounds, or discrete movements to transfer information stealthily. The channel being either the air or the light. In the context of circuits, the use of blinkers [GZE17] or noise [Gur+17] might be difficult to exploit. However, within a circuit there are other channels which can be used to the same end.

A typical system-on-a-chip (SoC) is an heterogeneous array of processing and memory elements. These components might be implemented within the same die or in different chips, but they generally share two kinds of wires: supply power and clocks. These lines are supposed to carry only DC components and square waves, respectively. But that does not mean that such signals cannot include additional information. As a practical example we might refer to smart grid applications [Ma+13]. In the case of clock signals, a modulation of the frequency of the wave or the duty cycle of the period can be used to encode a message covertly. For DC components, small power fluctuations can be induced in the signal to create a pattern which would encode the message. These strategies can be leveraged by different elements of the SoC to create a covert channel and transfer information, even if they are not supposed to communicate with each other.

The sender of the message is potentially a spy process which somehow manages to retrieve critical information from its surroundings (i.e. details from applications running in the same processor, activation signals, and even memory contents under certain scenarios). The receiver is then another application or circuit which can take advantage of this information to perform some processing or a different kind of attack. The entity which the channel aims to be hidden from is usually the owner of the platform; who would restrict the interaction between suspicious applications and circuits. As to where the sender and receiver might come from, some prime suspects include: kernel modules, drivers, third-party accelerators, hardware trojans, libraries, and binaries. The overseer would be the designer of the architecture, in charge of putting everything together. As the complexity of circuits increases, these scenarios result much more feasible since the use of foreign components becomes a necessity for speeding-up the design process.

Previous works on covert channels on SoCs [BB18; GER19; Gna+21; Mie+18] have focused on earlier technologies for these platforms. However, the design of newer devices has shifted towards an heterogeneous approach. Given the slowdown in Moore's Law [Eec17] it became evident that monolithic computing nucleus could no longer bear the burden of processing. Since then, we have seen the emergence of multi-core architectures. At the same time, as proposed by Amdahl's Law [HM08], optimizing the performance of the platform becomes more complex as the number of processors in the cluster grows. For this reason, the latest generations of SoCs now feature multiple processing clusters of different capabilities. Modern SoCs feature low-power processors, application processors, hardware accelerators, and even reconfigurable nucleus, which make it possible to integrate a diverse group of operating systems and applications in the same chip.

The goal of this work is to explore different approaches for creating frequency-based covert channels in heterogeneous SoCs. We study the interaction between different processor clusters of these platforms leveraging the reconfigurable fabric as the shared resource for the implementation of covert channels. These interactions are achieved by modifying some of the divider values with the intent of producing a change in the clock network. This clock tree is used as the channel for the covert transmission of data. Our main contributions can be summarized as follows:

1. We propose, for the first time, the creation of covert channels between different processor clusters of the SoC
2. We evaluate the performance of these attacks using a clear and well understood methodology
3. We demonstrate, also for the first time, the feasibility of implementing frequency-based covert-channels in the Zynq Ultrascale+ family of devices

The rest of the paper is structured as follows. In Section 2 we describe our experiments and methodology to evaluate the different covert channels in the Zynq Ultrascale+ SoCs. In Section 3 we provide our conclusions and final remarks.

1.1 Threat model

In [BB18], the authors used frequency modulation to create covert channels in the Zynq-7000 SoC. That work demonstrated that these architectures are vulnerable to these attacks even when the ARM TrustZone protections are enabled. In their case, a total of four covert channels were implemented, showing that the DVFS mechanisms available in SoCs could be used to bypass some ARM TrustZone protections. Other works, like [GER19] have demonstrated that it is possible to employ the electromagnetic emanation within the chip to implement covert channels. The authors exploited the cross-talk between long-wires to implement covert channels with transmission rates up to 6 Kbps in different FPGAs. More formally, voltage-based covert channel attacks were reported by [Gna+21], also for the Zynq-7000 SoCs. In that work, the authors managed to employ a power-waster circuit to generate fluctuations in the power supply of the circuit. Then, a sensor implemented in a different part of the reconfigurable fabric was used to retrieve the message. That work demonstrated that it was possible to implement power-based covert channels with transmission rates up to 8 Mbps.

The Zynq Ultrascale+ SoCs allow to use the RPU and the APU independently. The *real time* cores generally would run a real-time operating system like RTOS [Itu+15] or simply run standalone applications. The *application* cores, on the other hand, are complex and their full potential can usually only be achieved through the use of a kernel like Linux. Our work employs this model, which is based on reasonable assumptions about the use cases of the Ultrascale+ technology.

These systems also include a power management unit which is in charge of performing the monitoring and configuration of the Power Distribution Network. It includes anti-tampering characteristics which increase the difficulty of

modulating the power supply of the chip directly. That is, we might be able to induce a power fluctuation, but there is a chance that the SoC will go into lock-down. On the other hand, modifying the frequency of different clocks used in the architecture can be achieved through simpler mechanisms. By design a modification of the system frequency will be accompanied with a corresponding voltage scaling. Thus, the proposed covert channels target the frequency of the SoC but also affect its voltage.

In Fig. 1 we illustrate part of the clock tree in the Zynq Ultrascale+ SoCs. Four main reference clocks are available to source the five main PLLs of the architecture. To generate the output of these oscillators, the reference sources are multiplied by a small constant. Subsequently, the output of the PLLs is divided by up to two six-bit constants to produce multiple clocks for different parts of the architecture. From the same figure, it can be seen how there are three main power domains in these chips. The *Low Power Domain* will source the RPU, the peripherals, the on-chip memory, and one of the interconnect switches. The *Full Power Domain* will supply the APU, the memory management unit, the memory controller, and the central interconnect switch. And the *PL Power Domain* will supply the reconfigurable fabric. For each the low and full power domains, the five main PLLs can be used to generate clocks after applying one or two divider values. And for the FPGA, only three of the PLLs can be used to generate the four clocks available to this component (which come from the processing system, as it is also possible to use external clocks). Hence, modifying most of the clocks in the platform is just a matter of editing the value of the main multipliers or any of the dividers.

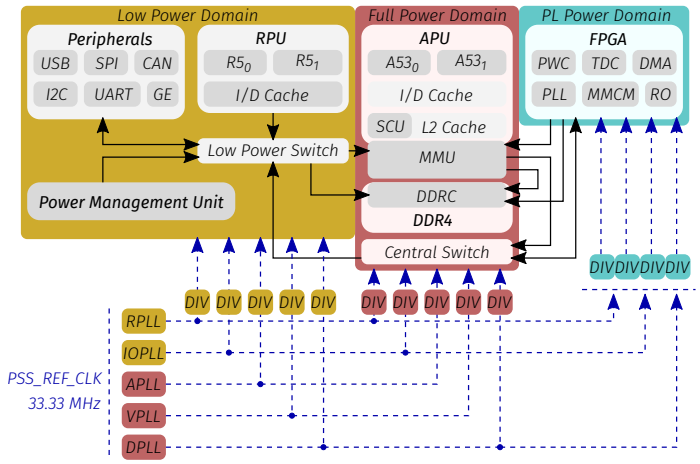


Figure 1: The clock tree and power domains of a Zynq Ultrascale+ SoC

2 Materials and Methods

Our work assumes that the spy process can gain access to phase locked loops (PLL) which can modify the oscillators in the SoC. We propose this assumption since the target scope of our work includes the Zynq Ultrascale+ SoCs which feature these components. Next, the receiver must be able to sample the channel

to retrieve the message. This is achieved with the use of a sensor module which can be implemented in the reconfigurable fabric. We must note that our work does not investigate a concrete scenario for these attacks to take place, but rather we intend to demonstrate the vulnerabilities present in this technology in order to mitigate such threats.

This work uses the TE0802 SoC (xczu2cg-sbva484-1-e) as target system. Compared to other Ultrascale+ SoCs, this board offers a reduced set of features. However, it can be used to demonstrate a wide range of vulnerabilities in this family of devices. We employ the AMD-Xilinx 2021.1 toolchain to design and implement the hardware architecture, as well as to develop and deploy the applications. The library `xtime_l.h` was used to obtain our time measurements.

2.1 Target platform

The Zynq Ultrascale+ SoC is an interesting case study for heterogeneous SoCs. These chips feature a main processing unit (APU), powered by an array of ARM Cortex-A53 cores. The secondary processing system (RPU) includes an array of ARM Cortex-R5F cores. Each one of these processors has independent instruction and data caches, and up to L2 cache in the case of the APU. The main memory of the SoC is an external DDR unit, driven by a silicon-based on-chip memory controller. There is also a smaller on-chip memory which can be shared by the different cores, and a memory management unit which performs the necessary assignments. What sets these architectures apart from other SoCs is the availability of an FPGA: an array of reconfigurable elements and silicon accelerators. The interconnection between processors and accelerators follows the AMBA-AXI specification through two main switches. The reconfigurable fabric of the SoC offers the possibility of implementing a wide range of customized accelerators.

In Fig. 1, in blue, we illustrate part of the clock tree in the Zynq Ultrascale+ SoC-FPGAs. A main reference clock (PSS_REF_CLK) is used to source the five main PLLs of the architecture (RPLL, IOPLL, APLL, VPLL, DPLL). To generate the PLL output, the reference clocks are multiplied by a constant. The resulting oscillators are then divided by one or two six-bit constants to produce specific clock domains for the different parts of the architecture.

From Fig. 1 it can also be seen how there are three main power domains in these Ultrascale+ SoCs. The *Low Power Domain* will source the RPU, the peripherals, the on-chip memory, and one of the interconnect switches. The *Full Power Domain* will supply the APU, the memory management unit, the memory controller, and the central interconnect switch. The *PL Power Domain* will supply the reconfigurable fabric. The goal for this separation of power domains is to improve the energy footprint of the system by allowing to shut down complete areas of the SoC when these are not required.

For the low and full power domains, the five main PLLs can be used to generate clocks. For the FPGA, only three of the PLLs (RPLL, IOPLL, DPLL) can be used to generate the four clocks available to the fabric (from the processing system, since it is also possible to use external clocks.)

Ultrascale+ SoCs allow to use the RPU and the APU independently. The cores in the RPU would normally run a real-time operating system like RTOS [Itu+15] or simply run standalone applications. The cores in the APU, on the other hand, are more complex and their full potential can best be drawn

through the use of a kernel like Linux. In this work, we presume that both clusters can be operated independently. We implement bare metal applications in the RPU and linux-based applications in the APU. These chips also feature a power management unit which is in charge of performing the monitoring and configuration of the power distribution network. It features anti-tampering characteristics which increase the difficulty of modulating the power supply of the chip.

2.2 Covert transmission of data

The proposed covert channels exploit the potential of the RPU and the APU for modifying the oscillators that source the FPGA. The frequency of the different clocks in Ultrascale+ SoCs can be modified by editing its multiplier or divider values. The multiplier register will affect the PLL output, and in turn modify the frequency of all the SoC components which rely on that given oscillator. In contrast, the divider registers are specific for a given clock and modifying them will only modify the frequency of a particular clock signal. There are clocks which use one divider and there are clocks which use two. All the dividers are stored as a six-bit section of a 32-bit register. To modify the frequency of an oscillators it is then necessary to edit the contents of these control registers.

At low level, like in bare-metal applications, the control registers of the SoC can be edited through direct access operations. For example using the `xil_io` library. However, to edit one of these control registers it is necessary to edit multiple security and configuration registers so that the frequency change is enacted. Furthermore, the application performing the operation must have the appropriate exception level.

In the presence of a kernel this task can be simplified with the help of drivers which allow to request the modification of specific clocks. For example, the processor clocks (by using the `cpufreq` driver of Linux) or the FPGA clocks (by using the `fclk` drivers of Xilinx). This scenario is more favorable for attackers since the complexity of the kernel allows to hide malicious applications more easily.

The SoC platforms utilized are by default protected with the ARM TrustZone firmware. The fabric is protected by an extension of this technology which allows to declare an IP as trusted. However, neither of these protections prevent the use of the clock drivers (`cpufreq`, `fclk`) so long as the application has root access.

2.3 Internal sensors

A delay sensor is a circuit created with digital components which can measure the variation in the propagation delay of a digital signal. These fluctuations are generated from variations in the power dissipation, electromagnetic coupling, and thermal fluctuations of the circuit [ZH12]. For this reason, such sensors have been employed to perform internal monitoring of the chip [ZS18; Gra+20]. The main types of such sensors are based in TDCs and ROs. The former are generally more accurate and provide greater resolution in the sampling, but must be calibrated precisely and placed directly in the platform. In contrast, the RO-based sensors (RO-S) do not require any fine-grained implementation

directives and provide sufficient information when enough samples are available. Our work focuses on the latter.

The main components of a RO-S are shown in Fig. 2. In this case, the ring of inverters provides a consistent oscillatory wave whose period fluctuates according to the nominal operation of the circuit. This signal is then used to source a binary counter which is subsequently sampled by an external clock to produce a measurement. The number of counts retrieved in a sampling period is thus correlated to the frequency of the ring, and in turn to the operation of the circuit. In a conventional binary counter we must consider the problem of carry propagation, which can drastically affect the critical path of the design. This problem, together with the need to reset the counter after each measurement, causes that the use of conventional counter designs derives into sensors with very low sampling rates. Both of these limitations were addressed by [Gra+19], who proposed to employ a carry-less ring counter which produces a Johnson encoding. This updated design is used in our work. However, we are more interested in the sampling clock of the sensor. By modifying this signal we can obtain an offset in the measurements due to the periodicity of the ring-counter.

As main sensing module we use the RO-S from [Gra+19]. This design features an acquisition rate over 350 Msps thanks to the highly optimized counter. The output of the sensors can be quantified with just ten bits, using the average of multiple sensors to mitigate the quantization error. The output of this module can be read directly from the FPGA, or retrieved from either the RPU or the APU through an AXI channel.

2.4 Physical characteristics of the channel

To understand the limits of the proposed covert channels we first characterized the behavior of a PLL in the target platform. Using a digital oscilloscope we sampled the time-response of these components when requesting a change in the output frequency. As reference, we generated a digital trigger through the processor's GPIOs. We then measured the width of these pulses. We also captured the activation of the MSB bit in the output of the RO-S (See Fig. 2). In Fig. 3 we illustrate our observations for this experiment.

Our findings suggest that the minimum response time for a frequency change is approximately $600ns$. That is the time elapsed from the moment one of the RPU cores modifies the register until the output of the sensor is updated ($t_{R5FtoFPGA}$). Therefore, assuming that we could transfer one bit per transition, the maximum bandwidth for the proposed channels would be 1.6 Mbps. Note that this is the theoretical limit, without considering the necessary delay to achieve a consistent transmission (low-error rate).

We observed that the response time to transition from a lower to a greater frequency differs from the time required to perform the opposite change. However, explaining this behavior falls out of the scope of our work; we simply take the maximum of both measurements to determine the minimal maximum-bandwidth. As we will demonstrate latter in the paper, this asymmetry does not weight on the feasibility of the proposed attacks.

Next, we intended to characterize the RO-S which would be used in our experiments. For this, we implemented a matrix of 64 RO-S and sampled it using different frequencies. Results for this experiment are provided in Fig. 4. At first glance it was possible to clearly differentiate between the multiple sample

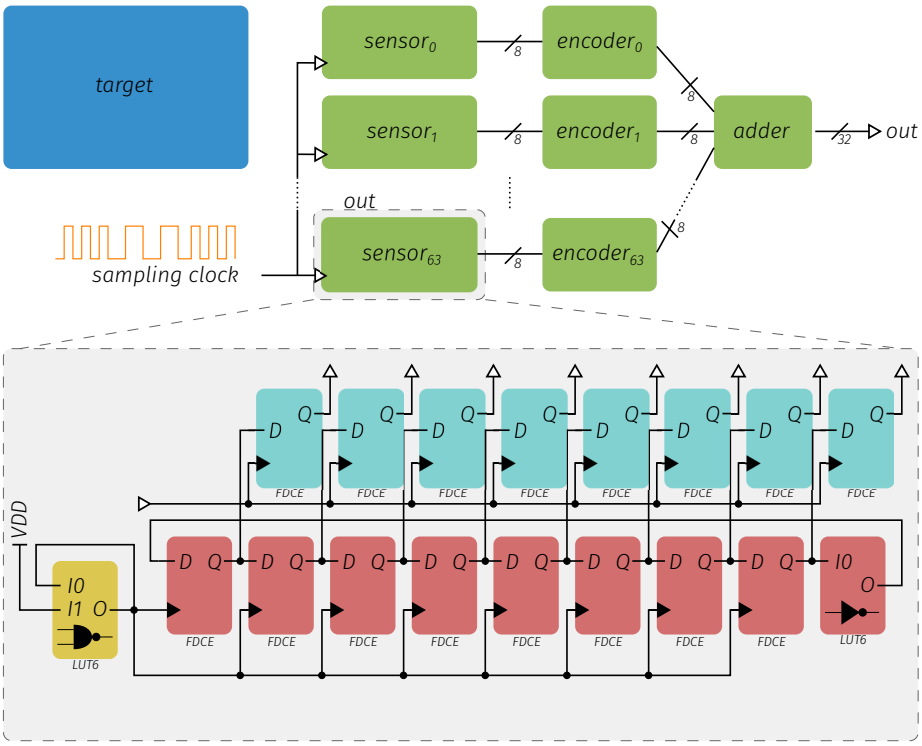
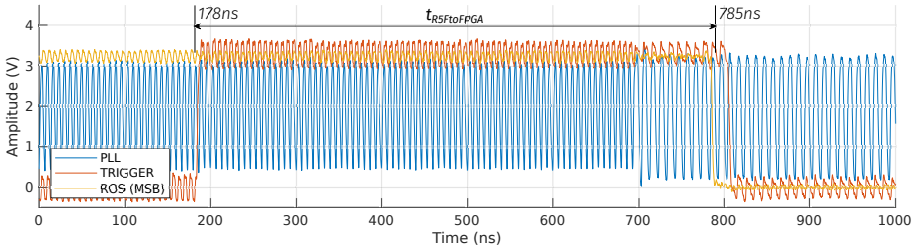


Figure 2: The architecture of the RO-S used in our work. This module is composed of three main groups of components. A group of sensors that produce a digital output in function of the operation of the circuit. An encoder for each sensor, which quantify the output of the counters. And an adder, whom merges all the encoders' outputs to mitigate the quantization error. There are also three main elements within each sensor: a ring oscillator (in yellow), a Johnson ring-counter (in red), and a register (in cyan). The acquisition rate is defined with an external sampling clock.

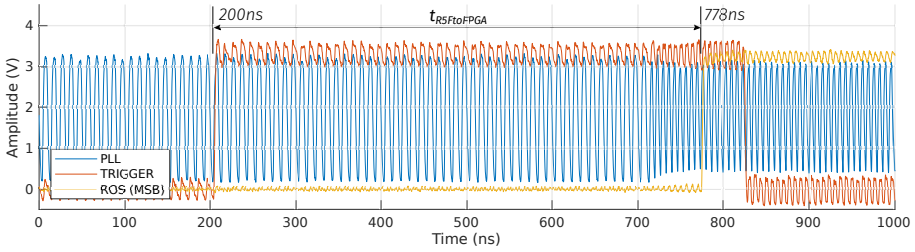
windows. However, for some cases, this separation would not be so apparent for a computer. The problem being that multiple sample values would overlap for different sampling frequencies. The outliers can be appreciated in the box diagrams included in Fig. 4. This kind of analysis was useful to identify the most adequate frequencies for implementing the proposed attacks. We selected frequencies that would simplify the implementation of the covert channels without requiring any filtering (100 MHz, 150 MHz, 300 MHz).

2.5 Covert channels between the RPU and the FPGA

The first class of covert channels we studied were those where an application running in the RPU (Cortex-R5F@533 MHz) acted as the transmitter, and a circuit implemented in the FPGA was the receiver. This scenario is illustrated in Fig. 5. Such attacks might be found in systems which use third party applications or accelerators. To perform the frequency modulation, the RPU-based application simply needed to access the `CRL_APB` module and overwrite



(a) Transition from 150 to 100 MHz



(b) Transition from 100 to 150 MHz

Figure 3: The time-response of a PLL in the Zynq Ultrascale+

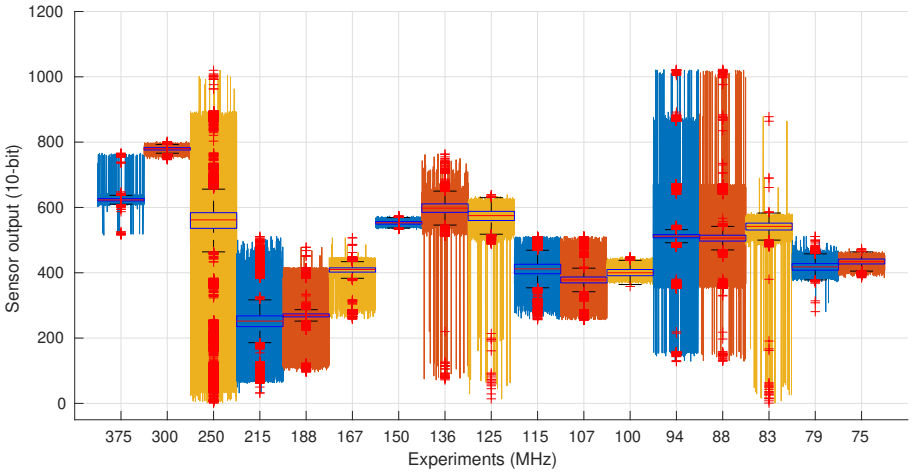
the six-bit dividers in the `PLX_REF_CTRL` registers (here `X` represents one of the four FPGA clocks). In our experimentation, these tasks were performed using the `xil_io.h` library.

Under this attack model, the bandwidth will be limited by the delay for the application to modify the target oscillator ($t'_{R5FtoFPGA}$). This delay is composed of the time required for a Cortex-R5F processor to modify a `CRL_APB` register (t_{R5F}), plus the transition delay of the PLL (t_{PLL}), and the time for the RO-S to update its output (t_{RO-S}). This model is given in Equation 1.

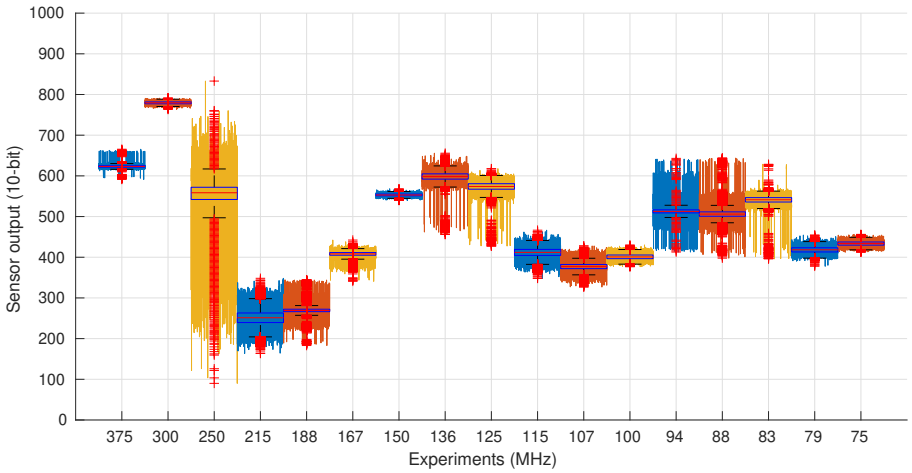
$$t'_{R5FtoFPGA} = t_{R5F} + t_{PLL} + t_{RO-S} \tag{1}$$

We went even further and decided to retrieve the output of the sensor from the processing system. This experiment sought to minimize $t'_{R5FtoFPGA}$ while maintaining an error rate of zero over the transmission of $12KB$ of data. For this, we used the modulation strategy in Alg. 1. This approach relies on three different frequencies: two that will represent ones and zeros and a third one that will act as a separator. By using three symbols it is possible to minimize the number of samples per window, as long as the different windows remain clearly differentiable. The results for this experiment are illustrated in Figure 6.

The samples shown in Fig. 6 were retrieved from the RPU by reading the output of the RO-S through an AXI link. Experimentally, we determined that $t'_{R5FtoFPGA} \approx 1.83$ μs. Whereas the FPGA could sample the output of the sensors with a rate of 333 MSps ($t_{FPGAtoFPGA} = 3$ ns), the RPU could only read the same output with an additional delay ($t_{FPGAtoR5F}$). Our experiments found that $t_{FPGAtoR5F} = 498$ ns. It follows that $t_{R5FtoR5F} = t'_{R5FtoFPGA} + t_{FPGAtoR5F} = 2.33$ μs. This is the latency for a Cortex-R5F core to communicate with the other Cortex-R5F through a covert channel which uses the FPGA as shared resource. The corresponding bandwidths are calculated in



(a) As a function of the sampling frequency



(b) After applying a moving average filter with a window of four samples

Figure 4: Characterizing the output of the RO-S as a function of the sampling frequency. The selected frequencies range from 375 MHz (which is the result of dividing the output of the IOPLL at 1.5 GHz by four) to 75 MHz (the result of dividing the same oscillator by 20). Each observation consists of 40,000 samples.

Equation 2.

$$
\begin{aligned}
B_{R5FtoFPGA} &= \frac{1}{2t'_{R5FtoFPGA}} = \frac{1}{3.66\ \mu s} \approx 273\ Kbps \\
B_{FPGAtoR5F} &= \frac{1}{2t_{FPGAtoR5F}} = \frac{1}{995\ ns} \approx 1\ Mbps \\
B_{R5FtoR5F} &= \frac{1}{2(t'_{R5FtoFPGA} + t_{FPGAtoR5F})} \approx 215\ Kbps
\end{aligned}
\tag{2}
$$

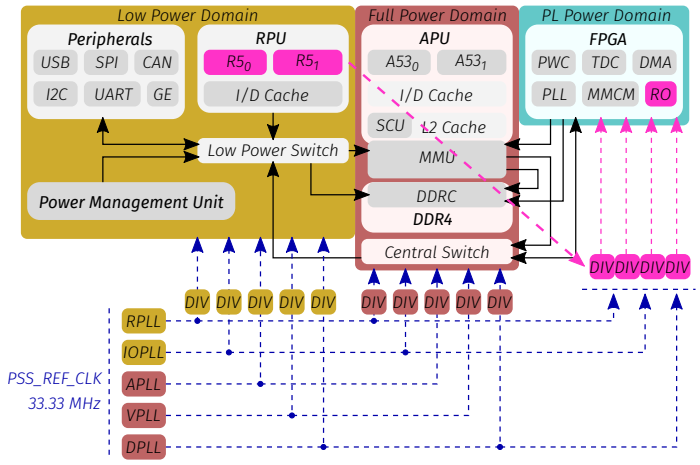


Figure 5: The information flow through a covert channel between the RPU (Sender) and the FPGA (Receiver)

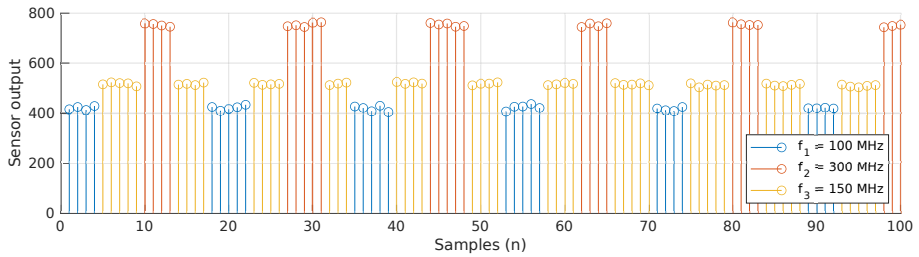


Figure 6: The transmission of a stream of bits over a covert channel from the RPU to the FPGA. The frequency-modulation is performed according to Alg. 1, with $f_1 = 100$ MHz, $f_2 = 300$ MHz and $f_3 = 150$ MHz.

2.6 Covert channels between the APU and the FPGA

The second type of covert channels under evaluation were those that originated from an application executed in the APU (Xilinx' Linux on Cortex-A53@1.3GHz). As in the previous attack, our intended receiver was the RO-S in the FPGA. This scenario is depicted in Fig. 7.

A regular Linux kernel, if configured properly, will feature the `cpufreq` driver which allows to modify the frequency of the underlying system. This might be leveraged to implement a covert channel between different cores controlled by the same operating system. In the case of the Xilinx' distribution of Linux, the kernel also features a set of APIs (`/sys/devices`) which allow to modify the frequency of the FPGA clocks. These two mechanisms use configuration files which can be managed from the application space. Thus, performing the modification of some oscillator is a matter of locating the adequate file, opening it, modifying its contents, and closing it again (the file must be closed for the change to be detected). Evidently, this is much more time intensive than writing a register. Therefore, for these attacks the bandwidth of the covert channel will be limited by the delay for the application (operating system) to modify the

Algorithm 1 A frequency-modulation strategy for low-width windows

Require: f_1, f_2, f_3: A given set of frequencies.

```
for byte in message do
  for bit in byte do
    if bit then
      fclk ← f1
    else
      fclk ← f2
    end if
    fclk ← f3
  end for
end for
```

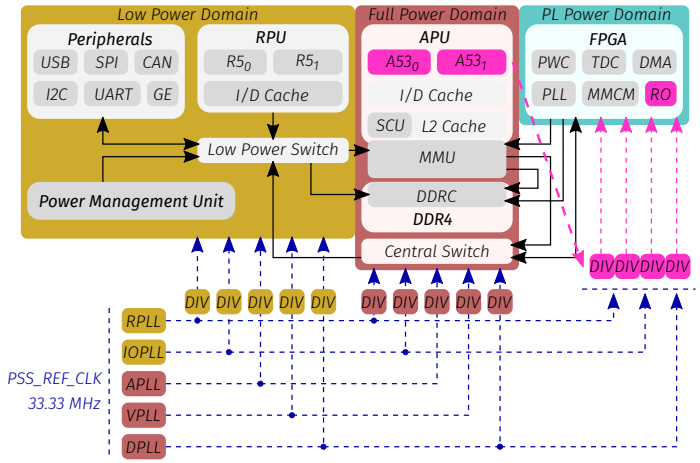


Figure 7: The information flow through a covert channel between the APU (Sender) and the FPGA (Receiver)

oscillator ($t_{OStoFPGA}$). This model is given in Equation 3.

$$t_{OStoFPGA} = t_{OS} + t_{PLL} + t_{RO-S} \tag{3}$$

In this case, since we expected $t_{OStoFPGA}$ to be much greater than $t_{R5FtoFPGA}$ we used the frequency modulation strategy in Alg. 2. Unlike the previous model, when there is a large number of samples it is possible to differentiate a transmitted zero from a one by adding a small delay (δ) in the transmission. If a divider value with minimum delay (t_{bit}) is used to separate the windows, then only two frequency values are required. The advantage of the new modulation strategy is that only three windows are used to encode two bits, rather than four. This results critical when the windows contain many samples. The results for an experiment with this type of channels are presented in Fig. 8.

For this experiment we saw it necessary to add a delay in the acquisition ($\zeta = 10$ μs) to reduce the number of samples collected. The data was retrieved using the RPU with a sampling period $t'_{FPGAtoR5F} = t_{FPGAtoR5F} + \zeta$ (it was experimentally corroborated that $t'_{FPGAtoR5F} = 11.05$ μs). We observed that $t_{bit} = 33 \times t'_{FPGAtoR5F} = 365$ μs and $t_{bit} + \delta = 39 \times t'_{FPGAtoR5F} = 431$ μs, so

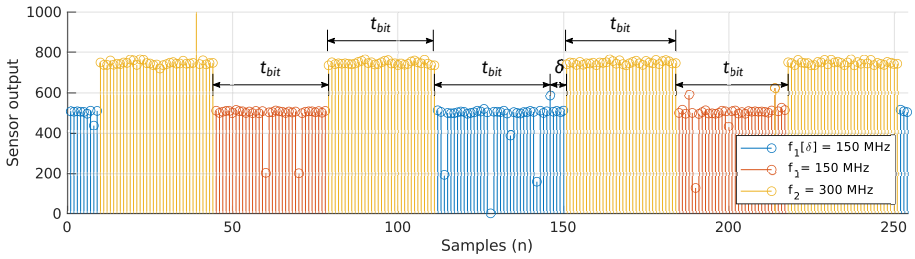


Figure 8: The transmission of a stream of bits over a covert channel from the Linux kernel to the FPGA. In this channel the modulation is performed according to Alg. 2, with $f_1 = 150$ MHz, $f_2 = 300$ MHz and $\delta \approx 66\mu s$.

Algorithm 2 A frequency-modulation strategy for large sample-windows

Require: f_1, f_2: A given set of frequencies.

```
for byte in message do
  for bit in byte do
    if bit then
      fclk ← f1
      wait(δ)
    else
      fclk ← f1
    end if
    fclk ← f2
  end for
end for
```

$\delta \approx 66$ μs (in practice δ was implemented as `usleep(1)`) and $t_{OStoR5F} = t_{bit}$. It follows that $t_{OStoFPGA} = t_{bit} - t_{FPGAtoR5F} \approx 354$ μs. As suspected, the transmission delay for this covert channel was about 200 times greater than the delay between the RPU and the PL ($t_{R5FtoFPGA}$). The corresponding bandwidths are given in Equation 4.

$$\begin{aligned} B_{OStoR5F} &= \frac{1}{2t_{bit} + 0.5\delta} = \frac{1}{795.94\ \mu s} \approx 1.31\ Kbps \\ B_{OStoFPGA} &= \frac{1}{2t_{bit} + 0.5\delta - t_{FPGAtoR5F}} \approx 1.33\ Kbps \end{aligned} \tag{4}$$

As it can be observed, the bandwidth for these covert channels is much lower than in the previous cases. However, modifying the clock frequency from the kernel space has two evident advantages: arguably it would easier to introduce malicious applications in a piece of software as large as Linux and the attacker does not need any specific knowledge of the architecture under attack. Which increases the range of potential targets.

2.7 Covert channels between the APU and the RPU

The experiments in the previous Subsection assumed that the transmitter in the Linux-enabled Cortex-A53 would use the `/sys/devices` APIs to perform

the frequency modulation of the FPGA clocks. However, this is not necessary if the application has access to the physical registers. To achieve this goal we can map the kernel memory through the `devmem` utility, then a register value can be read and written as any pointer from a C-language application.

Revisiting the case where a transmitter in the Cortex-A53 intends to send data covertly to an application in the Cortex-R5F (illustrated in Fig. 9), we now expected the number of samples per window to be much lower. Thus we continued to use the encoding from Alg. 1. Again, the APU transmitter would perform the frequency modulation but now through direct access to the PLL dividers. Then the RO-S in the FPGA would detect the frequency change, and its output would be read from the RPU. The total delay for the revisited covert channel from the APU to the RPU is modeled in Equation 5.

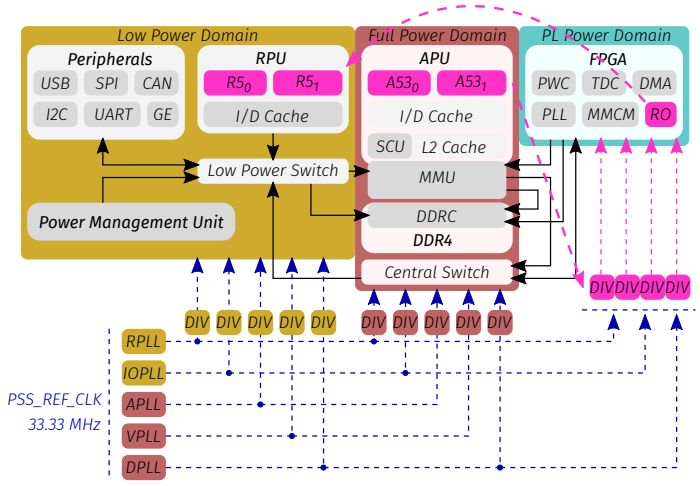


Figure 9: The information flow through a covert channel between the APU (Sender) and the RPU (Receiver) using the FPGA in the middle

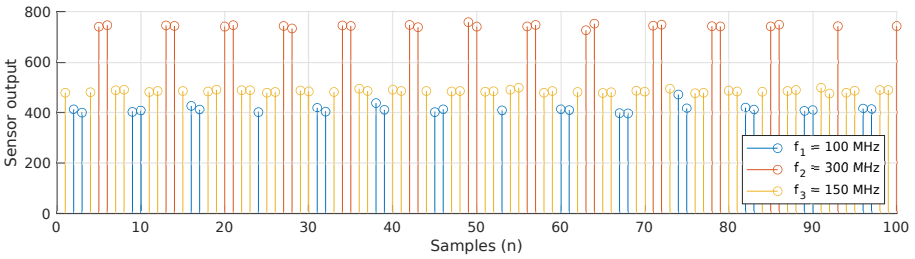
$$t_{A53toR5F} = t_{A53} + t_{PLL} + t_{RO-S} + t_{FPGAtoR5F} \tag{5}$$

Experimentally, we determined that it was possible to achieve a zero-error transmission over 30 KB of data with a small delay (η). In this case, the sender required to wait η after editing the register value. On the other hand, the receiver could continuously read the RO-S output.

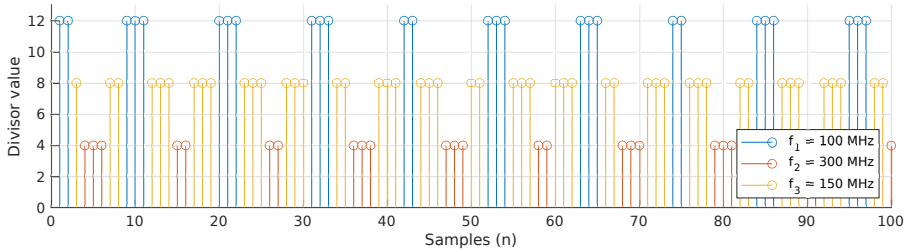
It results difficult to accurately measure $t_{A53toFPGA}$ from the kernel space, however, using the data in Fig. 10a we can estimate that $t_{A53toR5F} \approx 3 \times t_{FPGAtoR5F}$. With a conservative guess, we can then calculate the bandwidth for this cover channel as provided in Equation 6.

$$B_{A53toR5F} = \frac{1}{6t_{FPGAtoR5F}} = \frac{1}{2.99\ \mu s} \approx 335\ Kbps \tag{6}$$

This experiment contributed to demonstrate that both the Cortex-R5F and the Cortex-A53 cores can read and edit the registers in the `CRL_APB` module. Which means that we could simply create a register-to-register channel without the need to wait for the activation of the PLL. The results for this experiment are shown in Fig. 10b. With this approach the accuracy in the transmission



(a) using the FPGA as the shared resource



(b) using the registers of the `CRL_APB` module as the shared resource

Figure 10: The transmission of a stream of bits over a covert channel from the APU (Cortex-A53) to the RPU (Cortex-R5F). In this case the modulation is performed according to Alg. 1, with $f_1 = 100$ MHz, $f_2 = 300$ MHz and $f_3 = 150$ MHz.

increases considerably, to the point it is possible to use more complex modulation strategies, for example to increase the difficulty of detection, like the one in Alg. 3.

Formally, the difference in the register-register covert channel is that the Cortex-R5F application didn't have to retrieve the data from the FPGA (a difference of $\sim$ 90 ns). This is reflected in the delay model of Equation 7. Nonetheless, it remained difficult to estimate t_{A53} accurately.

$$t'_{A53toR5F} = t_{A53} + t_{PLL} + t_{RO-S} + t_{R5F} \tag{7}$$

Then, from Fig. 10b, since the sampling period can be considered equivalent to t_{R5F} and there are approximately five samples per bit transmitted, $t'_{A53toR5F} \approx 5 \times t_{R5F}$. The updated bandwidth is given in Equation 8.

$$B'_{A53toR5F} = \frac{1}{5t_{R5F}} = \frac{1}{2.04\ \mu s} \approx 490\ Kbps \tag{8}$$

Finally, we revisited the case where the transmitter would be the application in the Cortex-R5F and the receiver would be an application in a Linux-enabled Cortex-A53. In this case both applications could operate without any additional delay. We could measure the transmission delay for the channel, and by using the average number of samples per window estimate the delay for the Cortex-A53 to access a register in the `CRL_APB` module. Figure 11 illustrates some results for this experiment. From this evaluation we estimated that $t_{A53} = 243$ ns, and by consequence $B'_{R5FtoA53} \approx 750$ $Kbps$. The limiting factor being t_{R5F}.

Algorithm 3 A more complex frequency-modulation strategy

Require: f_1, f_2, f_3: A given set of frequencies.

```
for byte in message do
  for bit in byte do
    if bit then
      fclk ← f1
      fclk ← f2
      wait(δ)
    else
      fclk ← f2
      fclk ← f1
    end if
    fclk ← f3
  end for
end for
```

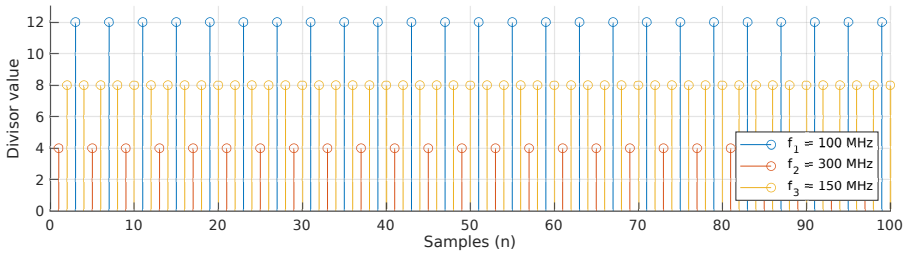


Figure 11: The transmission of a stream of bits over a covert channel from the Cortex-R5F to the Cortex-A53 using the registers of the `CRL_APB` module as the shared resource. As in the previous experiment, we do not sample the output of the RO-S but the value of a register. For this channel the modulation is also performed according to Alg. 1, with $f_1 = 100$ MHz, $f_2 = 300$ MHz and $f_3 = 150$ MHz.

2.8 Summary

In this Section we have described multiple covert channels which can be implemented between different clusters of the SoC and the reconfigurable fabric. For each experiment we have sought to maintain zero transmission errors and to estimate the minimum delay. While the performance results vary considerable between some of the attacks, it is necessary to remember that they have different use cases and advantages. In Table 1 we summarize our findings.

We have also retrieved results for selected works in the literature, which have implemented covert channels in SoCs and FPGAs. A direct comparison should be avoided since the implementation technologies differ. It is important to note that the qualitative characteristics of the channel should outweigh quantitative criteria such as the bandwidth.

In [BB18], the authors used frequency modulation to create covert channels in the Zynq-7000 SoC. That work demonstrated that these architectures are vulnerable to these attacks even when the ARM TrustZone protections are enabled. In their case, a total of four covert channels were implemented, showing that

Ref.	Sender	Receiver	Shared resource	Bandwidth (Kbps)
[BB18]	Cortex-A9	Spectrum analyzer	PDN	333
[BB18]	Cortex-A9	Cortex-A9	PLL	60
[BB18]	Cortex-A9	FPGA	PLL	125,000
[BB18]	FPGA	Cortex-A9	PLL	-
[GER19]	FPGA	FPGA	Long wires	6
[Gna+21]	FPGA	FPGA	PDN	8,000
This work	Cortex-R5F	FPGA	PLL	273
This work	Cortex-R5F	Cortex-R5F	FPGA	215
This work	Cortex-A53 (API)	FPGA	PLL	1.33
This work	Cortex-A53 (API)	Cortex-R5F	FPGA	1.31
This work	Cortex-A53	Cortex-R5F	FPGA	335
This work	Cortex-A53	Cortex-R5F	Register	490
This work	Cortex-R5F	Cortex-A53	Register	750

Table 1: Summary of the proposed covert channels

the DVFS mechanisms available in SoCs could be used to bypass some ARM TrustZone protections. Other works, like [GER19] have demonstrated that it is possible to employ the electromagnetic emanation within the chip to implement covert channels. The authors exploited the cross-talk between long-wires to implement covert channels with transmission rates up to 6 Kbps in different FPGAs. More formally, voltage-based covert channel attacks were reported by [Gna+21], also for the Zynq-7000 SoCs. In that work, the authors managed to employ a power-waster circuit to generate fluctuations in the power supply of the circuit. Then, a sensor implemented in a different part of the reconfigurable fabric was used to retrieve the message. That work demonstrated that it was possible to implement power-based covert channels with transmission rates up to 8 Mbps.

3 Final remarks

In this work, we have explored different alternatives for implementing covert channels in heterogeneous SoCs. Through experimentation we have demonstrated that it is possible to create covert channels between different components of these platforms. Using the Zynq Ultrascale+ SoCs as case study, we managed to create covert channels which achieved different transmission rates, going from a few Kbps to 750 Kbps. At the same time, we modeled the transmission delays and transmission bandwidth of the different covert channels adjusting the data with our empirical observations. These findings can be used to design more effective countermeasures for potential attacks based on covert channels.

As an additional contribution we characterized the response times of the PLLs in the Zynq Ultrascale+ SoCs. We demonstrated how these circuits have an asymmetric behavior as a function of the requested transition. For an ascending change from 100 MHz to 150 MHz we observed a transition delay of $600ns$; the equivalent descending change showed a slightly shorter transition delay.

We can conclude that, from an efficiency point of view, writing and reading the registers directly is the best option to implement covert channel communications. However, this assumes that the attacker has a) the necessary access level to read/write the control registers of the SoC and b) precise knowledge of

the architecture under attack. These assumptions limit the application of these kind of covert channels. On the other hand, performing frequency modulation from the kernel space is not so efficient, but it allows to mitigate these limitations. For the point a), it is much easier to sneak a malicious application into a large component like an operating system, and for the point b), the kernel will make sure that the drivers and APIs are pointing to the appropriate control registers regardless of the platform.

When comparing our results against the state of the art it is possible to reach mixed conclusions. However, we note that it is difficult to come up with a fair metric for comparison. First, because the implementation technologies and underlying phenomena are fundamentally different. And second, because the main goal of a covert channel is not to transfer a lot of data, but to do it stealthily. Finally, our work is not incremental to the related research; they are complementary. The vulnerabilities identified in this paper are architectural in nature, thus it would be possible to combine our work with other circuit-level principles for covert transmissions.

As future work, we intend to explore the impact of the modulation of the sampling frequency in the fine-output of the RO-S, which was not used in this work. We also intend to explore the possibility of creating the same covert channels when more restrictive control policies are implemented in the SoC, for example under trusted execution environments and power management software. Lastly, it might also be interesting to evaluate the application of the proposed covert channels under different SoCs which might not necessarily include an FPGA. This could be achieved through the use of hardware components which can be used to implement delay sensors [Gra+21].

Acknowledgments

The authors acknowledge the support of the French Agence Nationale de la Recherche (ANR), under grant ANR-19-CE39-0008 (project ARCHI-SEC).

References

[BB18] El Mehdi Benhani and Lilian Bossuet. "DVFS as a Security Failure of TrustZone-enabled Heterogeneous SoC". In: *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. Bordeaux, France: IEEE, 2018, pp. 489–492.

[Eec17] L. Eeckhout. "Is Moore's Law Slowing Down? What's Next?" In: *IEEE Micro* 37.04 (2017), pp. 4–5. ISSN: 1937-4143.

[GER19] Ilias Giechaskiel, Ken Eguro and Kasper B. Rasmussen. "Leakier Wires: Exploiting FPGA Long Wires for Covert- and Side-Channel Attacks". In: *ACM Trans. Reconfigurable Technol. Syst.* 12.3 (2019). ISSN: 1936-7406.

[Gna+21] Dennis R. E. Gnad, Cong Dang Khoa Nguyen, Syed Hashim Gillani and Mehdi B. Tahoori. "Voltage-Based Covert Channels Using FPGAs". In: *ACM Trans. Des. Autom. Electron. Syst.* 26.6 (2021). ISSN: 1084-4309.

[Gra+19] Joseph Gravellier, Jean-Max Dutertre, Yannick Teglia and Philippe Loubet-Moundi. "High-Speed Ring Oscillator based Sensors for Remote Side-Channel Attacks on FPGAs". In: *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. Cancun, Mexico: IEEE, 2019, pp. 1–8.

[Gra+20] Joseph Gravellier, Jean-Max Dutertre, Yannick Teglia, Philippe Loubet Moundi and Francis Olivier. "Remote Side-Channel Attacks on Heterogeneous SoC". In: *Smart Card Research and Advanced Applications (CARDIS)*. Cham: Springer International Publishing, 2020, pp. 109–125. ISBN: 978-3-030-42068-0.

[Gra+21] Joseph Gravellier, Jean-Max Dutertre, Yannick Teglia and Philippe Loubet Moundi. "SideLine: How Delay-Lines (May) Leak Secrets from Your SoC". In: *Constructive Side-Channel Analysis and Secure Design (COSADE)*. Cham: Springer International Publishing, 2021, pp. 3–30. ISBN: 978-3-030-89915-8.

[Gur+17] Mordechai Guri, Yosef Solewicz, Andrey Daidakulov and Yuval Elovici. "Acoustic Data Exfiltration from Speakerless Air-Gapped Computers via Covert Hard-Drive Noise ('DiskFiltration')". In: *2017 European Symposium on Research in Computer Security (ESORICS)*. Cham: Springer International Publishing, 2017, pp. 98–115. ISBN: 978-3-319-66399-9.

[GZE17] Mordechai Guri, Boris Zadov and Yuval Elovici. "LED-it-GO: Leaking (A Lot of) Data from Air-Gapped Computers via the (Small) Hard Drive LED". In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Cham: Springer International Publishing, 2017, pp. 161–184. ISBN: 978-3-319-60876-1.

[HM08] Mark D. Hill and Michael R. Marty. "Amdahl's Law in the Multicore Era". In: *Computer* 41.7 (2008), pp. 33–38.

[Itu+15] Xabier Iturbe et al. "Microkernel Architecture and Hardware Abstraction Layer of a Reliable Reconfigurable Real-Time Operating System (R3TOS)". In: *ACM Trans. Reconfigurable Technol. Syst.* 8.1 (2015). ISSN: 1936-7406.

[Ma+13] Ruofei Ma, Hsiao-Hwa Chen, Yu-Ren Huang and Weixiao Meng. "Smart Grid Communication: Its Challenges and Opportunities". In: *IEEE Transactions on Smart Grid* 4.1 (2013), pp. 36–46.

[Mie+18] Philipp Miedl, Xiaoxi He, Matthias Meyer, Davide Basilio Bartolini and Lothar Thiele. "Frequency Scaling As a Security Threat on Multicore Systems". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.11 (2018), pp. 2497–2508.

[ZH12] Kenneth M. Zick and John P. Hayes. "Low-Cost Sensing with Ring Oscillator Arrays for Healthier Reconfigurable Systems". In: *ACM Trans. Reconfigurable Technol. Syst.* 5.1 (2012). ISSN: 1936-7406.

[ZS18] Mark Zhao and G. Edward Suh. "FPGA-Based Remote Power Side-Channel Attacks". In: *2018 IEEE Symposium on Security and Privacy (SP)*. San Francisco, USA: IEEE, 2018, pp. 229–244.