# Computer Security by Hardware-Intrinsic Authentication

**Caio Hoffman**[1], **Diego F. Aranha**[1,2], **Mario Lúcio Côrtes**[1], **Guido Costa Souza de Araújo**[1]

[1]Institute of Computing – University of Campinas (UNICAMP)
Campinas – SP – Brazil

{caio.hoffman, cortes, guido}@ic.unicamp.br

[2]Department of Engineering – Aarhus University
Aarhus – Denmark

dfaranha@eng.au.dk

***Abstract.*** *The Internet of Things (IoT) has brought evident security concerns. New solutions in security for IoT will need to reduce the dependency on non-volatile memory for key storage, promote easier means to uniquely identify billions of devices, etc. Physical Unclonable Functions (PUFs) have been adopted as the future for key derivation and hardware fingerprinting. This work presents CSHIA: a new computer architecture that takes into account limitations and strengths of PUFs to provide code and data integrity and authenticity in a seamless design that does not demand changes in processors microarchitecture or software. We describe and analyze a full-fledged FPGA deployment of the architecture and consider attack scenarios, including side-channel attacks on PUFs.*

## 1. Introduction

As the Internet of Things (IoT) devices start to be popularized, their specifications, requirements, and, mainly, security concerns and threats become evident. While lower power consumption and area (i.e. cheap production cost) would be seen as the main drivers of the industry, security has been pointed out as a necessity for IoT rather than an option [ARM 2018]. To keep up with low costs and high viability, IoT security will need to reduce the dependency on non-volatile memory for key storage, promote easier means to uniquely identify billions (or perhaps trillions) of devices, guarantee authenticity and integrity of data and software, and yet easily integrate solutions of third parties (from hardware and software industries). Physical Unclonable Functions (PUFs) have emerged as the future for key distribution and hardware fingerprinting. However, their incorporation as a security solution for IoT demands new hardware architectures that will deal with PUFs' unreliability, additional area overheads, and other concerns. Endeavouring to facilitate the integration of PUFs to current hardware and software IPs, we present Computer Security by Hardware-Intrinsic Authentication (CSHIA). CSHIA is a new secure computer architecture that explores building blocks of computer architecture and the intricacies of PUFs to provide code and data integrity and authenticity, eliminating on-chip key storage, ensuring data origin, and making software tamper-evident.

**Contributions:** CSHIA's architecture is built upon a two bus organization [Shao et al. 2008]. One bus for main memory (as usual) and another for a tag memory. These tags are generated by pseudo-random functions using PUF-based keys and memory blocks.

Due to the intrinsic dependence of PUFs' behavior on fabrication process variations, each PUF-based tag (PTAG) is unique among different CSHIA instances, hence data cannot be moved from one device instance to another. An on-the-fly mechanism ensures that deviation between memory block content and its PTAGs are detectable. With the dual bus architecture, designers and engineers can deploy different bus technologies for the main memory and the PTAG memory. As a result, they can use different clock frequencies, arbitraging policies, word sizes, etc., to reduce verification overheads. In addition, because the processor is not aware of CSHIA, instructions and other modifications are not needed, resulting on 100 % software compatibility. Another aspect of CSHIA's flexibility is the possibility of using different countermeasures against replay attack. In a replay attack, an adversary restores previous values of authentic and non-tampered data, affecting the current state of systems, which enables attackers to tamper with registered consumption of smart meters, redo commands of smart switches, among others forgeries. In CSHIA, designers and engineers can opt between Timestamps or Merkle Tree without having to redesign any aspect of CSHIA's mechanism of code and data authentication and integrity verification.

Regarding PUFs, CSHIA presents a new way of extracting keys from SRAM-PUFs using the processor's cache – thus mitigating the need of additional circuitry. Finally, we implemented and tested a full-fledged FPGA deployment of the architecture using as baseline a well-known processor, and which has show limited overheads: average performance penalty between 2.76 % to 7.05 % (depending on the choice of countermeasures); area increment between 29 % to 38 % for FPGA (disregarding additional memory). At last, this work also delved into security evaluation of employing PUFs for unique device identification. We examined template side-channel attacks in a PUF variant, called XOR Arbiter PUFs, and we show that it could be possible to unveil one's hardware fingerprint using the knowledge of another with accuracy of up to 80 %.

## 2. Background and Related Work

This section navigates the essential concepts of PUFs and modern secure computer architectures, and what threats against IoT devices they could help to mitigate.

### 2.1. Physical Unclonable Functions

A PUF is a physical system that behaves similarly to a surjective function in which an input is a physical stimulus called *challenge*, and an output is called *response*. Although physical systems can macroscopically behave indistinguishably, the physical elements that constitute them are imperfect at some level (molecular level, atomic level, etc.). In electronics, a family of devices, despite equivalent functionality, will not have one instance microscopically equal to another, since the fabrication process is inexorably uneven. Hence, electronic PUFs exploit this intrinsic inequality to uniquely generate pairs of binary digital inputs (challenges) and outputs (responses).

As Challenge-Response Pairs (CRPs) of PUFs are distinctive for each physical instance, they are a manifestation of entropy. Extracting this entropy allows one to create singular root keys, which can be deployed in many cryptographic applications, including to certify origin of data, even when it is produced by equal devices, thus providing authenticity. In terms of leverages in system design, keys can be regenerated at any moment

by challenging PUFs, eliminating the need for non-volatile memory for key storage. This can bring enormous advantages for chip design. For instance, in micro-controllers, the absence of non-volatile memory has been reported to increase performance and reduce production cost by enabling higher clock speeds and decreasing the number of masks involved in the fabrication process [Mutschler 2020].

Nevertheless, due to the microscopic nature of the events from which information is extracted, variations in PUF responses occur, making them inconsistent over time and hindering key regeneration. Then again, this unreliability is remarkably contained, allowing ingenious schemes to overcome it. Among the new reliability schemes that were developed, Fuzzy Extractors (FEs) have been the most prominent. They use Error Correcting Codes (ECCs) and/or cryptographic primitives to boost entropy and decrease bias [Maes et al. 2016]. Simply put, FEs can be split into two procedures: enrollment and regeneration (reconstruction). In enrollment, FEs use responses extracted from PUFs to generate a key and an information-theoretically secure helper data. In regeneration, given the helper data and an approximation of the original PUF responses – containing noise due to inherent variations, FEs reconstruct the original key. With diligent design choices, FEs allow PUFs to match the robustness and security demanded by the IoT.

### 2.2. Modern Secure Computer Architectures

The main goal of a secure computer architecture is to incorporate mechanisms of confidentiality and integrity in hardware since such approaches "cannot be easily bypassed or subverted as software" [Szefer and Martonosi 2018]. To face the challenges of security in the IoT era, modern secure computer architectures should abandon non-volatile storage for keys, which can comprise security not only by the fact that keys are permanently stored, even in turned-off devices, but also because managing billions of keys is utterly difficult. As a result, PUFs seem to be a good alternative. During the almost two decades of PUF research, initiatives to construct secure computer architectures layered with PUFs as key generators were presented. AEGIS [Suh et al. 2005] was the first PUF-based architecture proposal and tackled many architectural and PUFs security concerns, providing a solid solution that gives confidentiality and integrity to code and data. However, it introduces additional instructions and modifications in operating system, hampering its adoption in low cost devices. Recently, SEPUFSoC [Sepúlveda et al. 2019] was proposed to provide code/data integrity and authenticity for multi-system-on-chip architecture. Interestingly, SEPUFSoC relies heavily on CSHIA's security design [Hoffman et al. 2015]: deployment of integrity tags for memory blocks which are generated from PUF-based keys applied to a pseudo-random function (PRF); verification of manipulation of helper data. Yet, in this work, we touch far deeper questions like the reuse of available circuitry in chips as PUFs and key extraction with measured reliability. All that without relegating essential matters of computer design such as area and performance overheads.

### 2.3. Common Threats and Threat Model

A threat model for IoT has to incorporate physical access to devices by adversaries. Devices will mostly run autonomous and unsupervised, and attackers will be able to explore vulnerabilities in the field or maybe move devices to places of their convenience. But, assuming that adversaries can decapsulate chips and use high-end tools like lasers, photonic emission detectors, among others, would not be realistic, since that involves slow procedures, years of expertise, and expensive equipment. It does not mean it cannot happen,

but rather it is unlikely to. Thus, we assume that at least the main chip of the architecture cannot be attacked invasively. From the point of the view of computer architecture, attacks include instruction and data manipulation and modification of peripheral hardware. These are translated into attacks to IoT devices like tampering with data integrity (e.g. manipulation of current measurements), tampering with data authenticity (e.g. moving data from one device to another), and tampering with code behavior (e.g. malware insertion). Finally, there is a gray area in the model regarding side-channel attacks because equipment is not highly expensive, skills are widely taught, and they do not need to be invasive. Countermeasures can be employed, albeit guarantees mostly depend on implementation as they are not standardized and different hardwares leak different information. Now, given the scenarios and challenges discussed here, we present CSHIA next.

## 3. CSHIA

As previously discussed, even under physical attacks, IoT devices need mechanisms to ensure that modifications in data and software are tamper-evident. CSHIA is an computer architecture that creates integrity tags for each memory block in main memory, guaranteeing detection of unwanted changes in code and data on runtime. Nonetheless, that would not be enough, because attackers can replace uncorrupted data and software between different versions and instance of IoT devices. To avoid it, CSHIA authenticates each tag with a PUF-based key. Hence, even if an adversary swaps uncorrupted memory blocks and their tags between equal devices, CSHIA can detect it, providing authenticity to code and data. What is more, a secure computer architecture cannot add prohibitive performance penalties and area overhead, in that CSHIA proposes (a) separated buses for main memory and the memory of PUF-based tags (PTAGs) and (b) reuse of processor's cache memories as SRAM-PUFs. As a result, designers and engineers can adopt different parameters, technologies, and versions of buses to match performance requirements. Due to PUFs rely on microscopic differences to generate randomness and uniqueness, their design – mainly in ASIC – demand great expertise to avert highly biased and low entropy implementations. However, SRAM start-up presents PUF behavior and given its naturally demonstrated randomness and reliability [Wang et al. 2020], besides being a very well-known standard in the industry, they can save additional area when available in systems. In CSHIA, we designed and analyzed an algorithm to extract stable and reliable keys from SRAM-PUFs, and we proposed it to be used in processor's cache.

The overview of the architecture of CSHIA can be seen in Figure 1. CSHIA has three main components: Bus Handler (BUS-HDLR), Security Engine (SEC-ENG), and the PTAG Memory (PTAG-MEM). PTAG-MEM is filled up with PTAGs computed from the initial values of the memory blocks during a secure enrollment procedure. Manufacturers and/or trust vendors should do it during production. Posteriorly, when is CSHIA functional, BUS-HDLR monitors processor requests and captures addresses and memory blocks that are handed in to SEC-ENG (A). In its turn, SEC-ENG consults PTAG-MEM through its memory management unit, PMMU, (B). Meanwhile, the PTAG generator (PTAG-GEN) uses the memory block and its address received from BUS-HDLR to recompute a PTAG and send it for comparison (C). The PTAG from PTAG-MEM is also sent to comparison (D). If it fails, BUS-HDLR inhibits altered code and data to reach the processor (E). When the core wants to write a cache line back to main memory, BUS-HDLR will hand in captured input to SEC-ENG (A). But, this time, SEC-ENG uses the

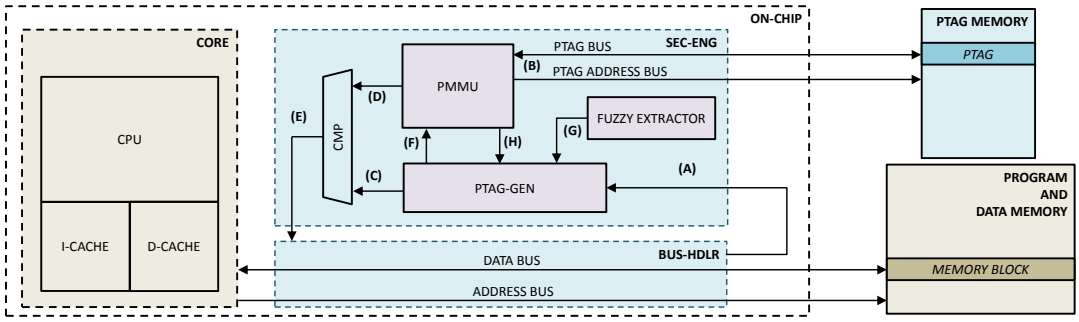


**Figure 1. Architecture of CSHIA.**

PTAG-GEN to generate an up-to-date PTAG that is passed to PMMU (F) and which writes it into PTAG-MEM (B). This process is transparent for the processor, main memory, and other peripherals of traditional computing systems. The Fuzzy Extractor is only activated during enrollment and restart of the system. It extracts a PUF-based key and delivers it to PTAG-GEN (G). The extraction can be done as proposed, using the processor's cache, or using any other PUFs. PMMU can provide information to PTAG-GEN (H) when countermeasures against replay attacks are deployed. With Timestamps, it would be the time of each PTAG last update. With Merkle Tree, PMMU would need to provide blocks of PTAGs to update ascendant PTAGs in the tree. Because this process is complex and can take multiple iterations, a PTAG cache memory should be used for better performance.

## 4. Implementation and Results

A proof-of-concept implementation of CSHIA was realized over the open-source code of Gaisler's Leon3 processor. The implementation used the Terasic DE2-115 Board that has an Altera's FPGA. Between conceptual design and implementation a major trouble had to be addressed: In this particular FPGA, SRAM memories are always initialized to zero. Despite having an external SRAM in the DE2-115 board, we decided not to use it, since, among many downsides, the SRAM could not be turned on and off independently from the rest of the board: essential feature to implement our algorithm of key extraction from processor's caches. Consequently, we opted to use a FPGA implementation of Arbiter PUFs (APUFs) for key extraction. The major impact of such a limitation was that comparing practical and analytical results of the algorithm was not possible. In the analytical experiments, using data from SRAMs of multiple boards, we emulated key extraction – as CSHIA would do – that consisted in locating stable words among multiple startups of cache memories. Combinations of those words enabled random keys whose probability of failure would be 1 in 1,000,000 regenerations, besides having small helper data.

Regarding architecture, BUS-HDLR was implemented over the AMBA Bus between Leon3's core and main memory. We use FPGA internal memories for PTAG-MEM and the additional memories of the replay attack countermeasures. Regarding security, we used SipHash as CSHIA's PRF, which requires a 128-bit key and produces 64-bit hashes. Thus, in terms of brute-force efforts, our keys would require $2^{128}$ guesses and PTAGs would demand $2^{64}$ attempts to forger them. As a result, considering the birthday problem, CSHIA's replay attack countermeasures can be effective for distinctive collections under $2^{32}$ PTAGs. In other words, if an attacker observes a singular memory block and collects $2^{32}$ different values of that memory block, he/she has 50 % of chance to find, in such a collection, two equal PTAGs, making viable a replay attack, but in that specific

**Table 1. Performance overhead in % of the seven CSHIA instances evaluated in comparison to running times in *Leon3 Baseline*.**

| | *CSHIA-TS* | *CSHIA-MT* instances | | | | | |
|---|---|---|---|---|---|---|---|
| | | 16x8-LRU | 32x4-LRU | 64x2-LRU | 16x8-ALAP | 32x4-ALAP | 64x2-ALAP |
| ***Average*** | **2.76** | **5.99** | **6.12** | **5.77** | **7.05** | **6.03** | **5.99** |

memory block only. In Appendix D of the Thesis [Hoffman 2019], we apply some examples of replay attack in our CSHIA's FPGA implementation and we demonstrate how each countermeasure, Timestamp or Merkle Tree, thwart those attacks.

For performance evaluation, we chose some benchmarks of Mibench: `basicmath`, `bitcount`, `dijkstra`, `fft`, `qsort`, `sha`, `string_search`, and `susan`. Table 1 shows the average performance overhead computed from benchmark runs in multiple configurations of CSHIA. *CSHIA-TS* is when Timestamps are used to deal with replay attacks and *CSHIA-MT* represents the use of Merkle Tree. A 4-KB cache memory under different combinations (lines versus sets: 64x2, 32x4, 16x8) and policies (LRU and ALAP) was used to store PTAGs and speed up Merkle Tree integrity verification. Disregarding additional memories that each countermeasure imposes, the proportional area overhead of logic elements is 29 % for *CSHIA-TS* and 38 % for *CSHIA-MT*. Later, after the finalization of the thesis, we synthesized CSHIA using Cadence synthesis tool and those overheads were reduced to a maximum of 21 % (without memories).

Notwithstanding, CSHIA overheads are comparable (or better) to those in the literature. For instance, SEPUFSoC presents an area increment of at least 33 % (in FPGA) and AEGIS of 90 %. In terms of worst-performing benchmark, SEPUFSoC reaches 25.4 % degradation and AEGIS 73.1 %. Our instance of *CSHIA-MT* 16x8-ALAP is the one with the worst performance with 24.85 % degradation running `sha`. Notice that our overheads of area include the APUFs and their additional controllers. They would be smaller if we could use SRAM-PUF as solution. As performance and area can be improved by optimizing the implementation, one should look to these results as worst-case scenarios for the particular benchmarks and configurations. It is worth to notice that, to be secure, Merkle Tree and Timestamps solutions require on-chip non-volatile memory. Although non-volatile storage of keys is insecure, access to metadata of replay attack countermeasure is noncritical, as long as attackers cannot write in such memories. Regarding overheads, while we only need to store the tree root of Merkle Tree, for Timestamps, we need to store the latest time-counter value of each PTAG. Despite that, *CSHIA-TS* had better performance and smaller additional logic than *CSHIA-MT*. Choosing between these solutions will be up to the designers/engineers, but CSHIA's features facilitate it by avoiding changes in the integrity mechanism, PUFs, FE, software, and peripherals, regardless of which option is picked. An easier path to widespread adoption of more security in IoT.

## 5. Security Evaluation

As aforementioned, one of the key features obtained from using PUFs in IoT is the assignment of unique identity to each device. These identities could originate from the fact that memory content of one device cannot be moved to another, or that the entire set of integrity tags of memory blocks define a particular instance of a device. Both ways could be seem as forms in which CSHIA deliveries identity. In the case of a successful attack in CSHIA, it would indicate that an adversary guessed the PUF-based key or was able

to create PTAGs that match his/her tampered code/data. Such tampered memory blocks, however, could not be moved to other device unless the attacker can figure out again the device key or can luckily guess the PTAGs accepted by it. Still, one hypothetical attacking scenario remains. What if adversaries could learn information of one device and use it to gain knowledge of the hidden information of other devices? In particular, could attackers unveil a device's key using one originated from another instance of that device.

As machine learning (ML) is all about using previous knowledge to predict unknown information, one could imagine that it would be the answer of the previous question. But, many works in the literature have applied ML to PUFs [Rührmair et al. 2010] and none have shown or indicated that such an approach would work. Given that and the fact that side-channel attacks can be a real threat to IoT devices, we decided to mount a variant of them, called Template Attack. It uses a template created by collecting power traces of responses of a profiled PUF to try to obtain unknown responses of a different instance of that PUF. If an attack like this works, it would indicate a possible way in which an attacker could circumvent the uniqueness that PUFs stablish.

For this experiment, we implemented one XOR Arbiter PUF (XOR-APUF) in two equal FPGAs. A XOR-APUF combines responses of multiple APUFs using a XOR logic function to generate responses. We used one FPGA to create a template of power traces of a randomly-picked CRP set, and we attacked the other FPGA, aiming at unveiling PUF responses of a different randomly-picked CRP set. As a matter of fact, choosing different and random CRP between template creation and attack gives a more realistic scenario. Among multiple configurations of attacks, we were able to reach 80 % of accuracy. Meaning that if an attacker creates a template of responses of each individual XOR-APUF in one device, it would be possible, with 80 % of probability, to correctly guess individual XOR-APUF responses in a different instance of that device[1]. Despite the fact that the experiments were conducted in the controlled environment of a laboratory, they show attack viability, and considering that our threat model acknowledges side-channel attacks, concrete situations where they can occur should not be overlooked.

## 6. Conclusion

Security for IoT devices will not be solely achieved by software measures and traditional computer architecture. To face the challenges we shed light on, the incorporation of PUFs into secure architecture designs seems to be the best available approach. However, this also brings new security challenges to deal with. For this reason, in this work, we presented CSHIA, a secure computer architecture that provides authenticity and integrity to code/data by means of PUFs. CSHIA innovates in tackling PUF unreliability and overheads together with flexible design that aims at mitigating performance and area overhead, which are inherently common in secure computer architectures. Resulting overheads are comparable (and sometimes better) than current options presented in the literature. Besides all that, we also delved into performing attacks in CSHIA and PUFs. In particular, we showed how one could circumvent the unique keys that can be derived from PUFs in a side-channel attack. Yet, such attack needs to be proved viable in the field. Thus, overall, CSHIA is a robust solution that addresses many threats that IoT devices will be facing.

[1]The accuracy of the attack reached 82 % when the profiled and attacked PUF instance was the same.

## 7. Publications

First publication: [Hoffman et al. 2015] (A2 CAPES' Qualis); Second publication: [Hoffman et al. 2019] (B1 CAPES' Qualis). A patent deposit: https://patents.google.com/patent/WO2017008133A1/en. Finally, a new and improved version of the second paper from Chapter 2 of the thesis is under preparation for resubmission to a journal.


## Acknowledgements

We thank FAPESP (File N°2015/06829-2 and File N°2016/25532-3), CNPQ (Files N°147614/2014-7 and N°200362/2016-0), Intel's Research grant "Energy-Efficient Security for SoC Devices – Physical Unclonable Function", and CAPES.


## References


ARM (2018). Arm cyber security manifesto 2018/2019. Retrieved Jun 21, 2020 from https://www.arm.com/resources/manifesto/iot-security.

Hoffman, C. (2019). *Computer Security by Hardware-Intrinsic Authentication*. PhD thesis, University of Campinas. Availale at: http://repositorio.unicamp.br/bitstream/REPOSIP/333630/1/Hoffman_Caio_D.pdf.

Hoffman, C., Cortes, M., Aranha, D., and Araujo, G. (2015). Computer security by hardware-intrinsic authentication. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2015 International Conference on*, pages 143–152.

Hoffman, C., Gebotys, C., Aranha, D. F., Cortes, M., and Araujo, G. (2019). Circumventing uniqueness of xor arbiter pufs. In *2019 22nd Euromicro Conference on Digital System Design (DSD)*, pages 222–229. IEEE.

Maes, R., van der Leest, V., van der Sluis, E., and Willems, F. (2016). Secure key generation from biased pufs: extended version. *Journal of Cryptographic Engineering*, 6(2):121–137.

Mutschler, A. S. (2020). Non-volatile memory tradeoffs intensify. Retrieved Jun 27, 2020 from https://semiengineering.com/non-volatile-memory-tradeoffs-intensify/.

Rührmair, U., Sehnke, F., Sölter, J., Dror, G., Devadas, S., and Schmidhuber, J. (2010). Modeling Attacks on Physical Unclonable Functions. CCS '10, pages 237–249. ACM.

Sepúlveda, J., Wilgerodt, F., and Pehl, M. (2019). Towards memory integrity and authenticity of multi-processors system-on-chip using physical unclonable functions. *it - Information Technology*, 61(1):29 – 43.

Shao, F., Sun, R., Diao, K., and Wang, X. (2008). A new secure architecture of network computer based on single cpu and dual bus. ISEC '08, pages 309–314. IEEE.

Suh, G. E., O'Donnell, C. W., Sachdev, I., and Devadas, S. (2005). Design and implementation of the aegis single-chip secure processor using physical random functions. *SIGARCH Comput. Archit. News*, 33(2):25–36.

Szefer, J. and Martonosi, M. (2018). *Principles of Secure Processor Architecture Design*. Morgan & Claypool Publishers.

Wang, R., Selimis, G., Maes, R., and Goossens, S. (2020). Long-term continuous assessment of sram puf and source of random numbers. DATE '20, pages 7–12, San Jose, CA, USA. EDA Consortium.