On the Design of a Processor Working Over Encrypted Data

Thomas Hiscock, Olivier Savry
Univ. Grenoble Alpes,
F-38000 Grenoble, France.
CEA, LETI, MINATEC Campus,
F-38054 Grenoble, France.
Email: thomas.hiscock@cea.fr, olivier.savry@cea.fr

Louis Goubin
Laboratoire de Mathmatiques de Versailles,
UVSQ, CNRS, Universit Paris-Saclay,
F-78035 Versailles, France.
Email: louis.goubin@uvsq.fr

***Abstract*—Gentry's breakthrough of Fully Homomorphic Encryption (FHE) in 2009 revolutionized the field of secure computation. Since then, most applications of homomorphic encryption have been oriented towards offloading computations to the cloud in a secure fashion. Indeed, the user usually does not have full confidence in the cloud provider and wants to keep its data secrecy. A similar situation appears in most embedded systems, where information leakages through hardware or software side-channel attacks might compromise data confidentiality.**

In this work, we attempt to leverage Homomorphic Encryption in a different threat model, adapted to CPS (Cyber-Physical Systems) use cases. The main challenge is that, even today's most promising FHE schemes remain orders of magnitude too big to fit in a constrained system. To address this issue, we show how a trade-off can be achieved by securing a noise reduction module against side-channel leakages. This approach is described and evaluated on FPGA using the BGV scheme, a very efficient homomorphic scheme based on Ring-LWE encryption. We conclude that such homomorphic encryption can fit in an embedded system, while offering reasonable performances with respect to the security provided.

I. Introduction

Memory isolation in processors is a fundamental building block for programs security. It ensures that a given program cannot access or modify the memory of another program. On most processors, memory isolation is enforced using virtual memory, which exposes different address spaces for different processes. Access permissions are checked on every access by a special component, known as the memory management unit (MMU).

However, such isolation is not as strong as expected in many use cases. For instance, a system physically close to the user like an embedded system is subject to a wide range of physical attacks. External memories like FLASH, or DRAMs may be extracted, disclosing anything stored in memory. Other example threats include logical interfaces (debug, JTAG) which are accessible to an attacker, buses that can be physically probed and also side-channel attacks [1] that can extract sensible information from physical observations (power consumption, electromagnetic field, acoustic noise...). Even without a physical access, memory isolation can be bypassed by the so-called software side-channel attacks. Indeed, it has been known for years that information may leak through caches memories [2], [3], with attacks like Flush+Reload [4] or Prime+Probe [3]. Recently, the Spectre [5] and Meltdown [6] attacks showed that some microarchitectural elements (related to speculative execution and Out-of-Order execution) can be exploited to leak critical information, like bits from kernel memory. As these attacks are purely software-based, they can be executed remotely.

A stronger isolation can be achieved with memory encryption and is used in many architectures [7]. Observing only the encrypted data cannot leak information as long as the keys are managed properly. Encryption and decryption are usually inserted between the chip and the memory interfaces, so that nothing can be extracted from memories. However, leakages on shared hardware like caches, branch predictors or other microarchitectural elements remain possible. Indeed, data still needs to be decrypted to be processed and any hardware influenced by these unencrypted values might introduce a leakage.

Fully Homomorphic Encryption (FHE), discovered by Gentry in 2009 [8], allows arbitrary computation over encrypted data. The typical application of this scheme is considered in the cloud, where the user does not have any trust in the service provider. To maintain its privacy, his data must remain encrypted from end to end. In this work, we attempt to leverage Homomorphic Encryption in a totally different threat model, adapted to CPS (Cyber-Physical Systems) like an autonomous car where the manufacturer of a product has a total control over it during all its life cycle and as a consequence it is confident about the management of its hardware platform. Those applications challenge us to rethink homomorphic encryption to fit in embedded processors. Indeed, FHE offers a very convenient protection, since it allows data to remain encrypted in memory and even during computations. It can still be processed, without any fear of information leakage.

However, homomorphic encryption is sadly known to be inefficient. Since Gentry's breakthrough, the only way to achieve fully homomorphic encryption remains through a procedure called the *bootstrapping*. In a nutshell, the bootstrapping is the process of evaluating the decryption function homomorphically, over an encrypted ciphertext (that is, two layers of encryption). The bootstrapping is an important bottleneck in the performances of FHE. And many active research try to tackle this issue by proposing schemes with more efficient bootstrapping [9].

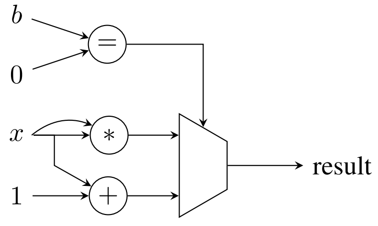
Contributions: In this work, we describe our approach to design a processor working over encrypted data. The performance bottleneck implied by the bootstrapping is prohibitive for an embedded implementation. As an alternative, we propose an approach which is consistent with a CPS use case model where it is reasonable (and often needed) to store a secret key in a protected way in the core of the processor. The bootstrapping procedure is then transformed into an efficient hardware noise refreshing procedure, which achieves the same functionality but is secure under some hypothesis that we detail. Finally, we evaluate the hardware cost with a FPGA implementation and discuss the synthesis results. We conclude by evaluating the performances of a homomorphic evaluation of an AES circuit and compare it with current state of the art results.

II. Description of the Architecture

Before diving into the details, let first draw an overview of the architecture. A given algorithm to be protected, is first transformed into a circuit, namely, a directed acyclic graph of high-level operations. Through this process, conditional branches are turned into a multiplexor form, using *if-conversion*, and all loops are unrolled. An example of such conversion is shown in Figure 1. Once transformed into a circuit, the algorithm becomes constant time and memory oblivious (memory accesses are independent of the input data).

```
int square_or_inc(int x, int b) {
    if (b == 0) return x * x;
    else        return x + 1;
}
```

(a) Example program



(b) Equivalent circuit

Fig. 1: Example of a simple program to circuit transformation for homomorphic evaluation

From the high-level circuit, all operations (like integer arithmetic, comparisons, ...) are lowered to a very small set of low-level operations. We selected the Brakerski, Gentry, Vaikuntanathan (BGV) [10] scheme to implement homomorphic operations. With a suited, encoding it can provide the logical operations XOR, AND and bit shifting on 8 bits data. To end up the conversion, all inputs and intermediate constants nodes of the circuit are encrypted under a public key and stored into memory. The circuit of lowered operations is then converted to an assembly program (calling mostly homomorphic operations) evaluating it.

Each basic homomorphic operation is followed by a refreshing operation, whose purpose is to maintain a tight bound on the noise in a ciphertext. The next section describe our efficient implementation of this operation.

A. An Efficient Noise Refreshing Procedure

After too many homomorphic evaluations, the noise in a ciphertext becomes too high and decryption correctness cannot be guaranteed anymore. When such situation occurs, the so-called bootstrapping technique must be used. In a nutshell, the bootstrapping is the process of evaluating the decryption function over encrypted data. Since Gentry's breakthrough work in 2009 [8], it remains the only known way to achieve Fully Homomorphic Encryption (FHE). Several paths are explored to improve the efficiency of homomorphic encryption, either through theoretical improvements on the scheme itself [9], or with better algorithms.

A straightforward, but efficient way to achieve an bootstrapping functionality is to decrypt the ciphertext (which removes the noise) and then encrypt it back. This produces a ciphertext with better noise bounds satisfied, thus called a ”fresh ciphertext”. Obviously, this introduces a security issue, as the secret key is required for the decryption, and the clear message will be manipulated in various places.

However, considering a hardware implementation, this flow can be secured. An overview of our secure recryption procedure is depicted Fig. 2. Before being decrypted (Fig. 2), the encryption of a random value is added to the input message (denoted m). The additive property of the homomorphic encryption allow this mask to be applied and removed in an oblivious way on the ciphertext. A similar technique was used in [11] on a RLWE encryption scheme to randomize the decryption against DPA (unfortunately, no formal security could be proved). Here, the goal isn't to randomize the decryption (which is already assumed leak-free) but rather to protect the intermediate plain message m over potential leakages during the encryption.

The security of the noise refereshing procedure depicted on Fig. 2 relies on two assumptions :

1) The decryption implementation must be leak-free or designed to be resilient against d-th order attacks. Such resilience can be achieve with a generic masking scheme like ISW [12] or by adapting the probabilistic decoder of [13].
2) The generation of the random encryption, the term $\texttt{Encrypt}_{pk}(r)$, must be leak-free.

Under these assumptions, in the worst possible situation an adversary would only be able to recover $\texttt{Encrypt}_{pk}(m)$, $\texttt{Encrypt}_{pk}(r)$, $m \oplus r$. In this context, the security falls into a classic key encapsulation mechanism (see [14], theorem 11.12), extensively used in practice to establish symmetric communications channels. The value r can be seen as a private key being encrypted under the public key and $m \oplus r$ be an encryption (using a one-time pad) of m. In other words, an

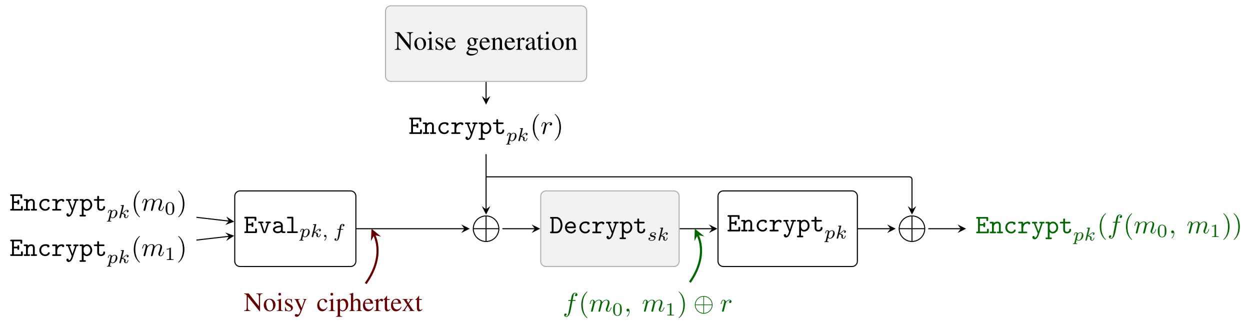


Fig. 2: Data flow of the proposed hardware noise refreshing procedure

adversary cannot learn anything about either m or r from these three values, otherwise, it would break the IND-CPA security of the encryption scheme (which holds under the RLWE assumption).

A good property of this approach is that there is no need to protect the encryption function against leakages. Several strategies can be applied to generate the random encryption term $\texttt{Encrypt}_{pk}(r)$:

- a first one would be to store them into a huge storage and use them only once. Obviously this seems to limit the range of applications.
- A second one is to generate uniformly random values and encrypt them. This has to be done in a tamper proof area, so that nothing leaks about r.
- Another interesting approach would be to design a PRNG for such values, which would have stable noise magnitude. Whether this is possible is an open question, needless to say it would have interesting applications.

III. Evaluation

A. Implementation

The architecture described in this paper is currently implemented in a mixed software hardware design targeting FPGA. The whole homomorphic-related components are grouped into a tile (see Fig. 3), that we call a homomorphic processing element. It receives execution commands (assembly-like instructions) from a classical core, and shares a section of the main DDR3 memory with the CPU. A MIPS soft core is currently used for this purpose, but any soft core processor would be fine for this task.

Fast modular multiplication is done using Barrett quotient approximation method. The resulting circuit is fully pipelined and has an initialization delay of 9 cycles. Polynomial multiplication is done using using the analogous of the Fast Fourier Transform over a finite field called the Number Theoric Transform (NTT) [15]. The NTT is implemented using a fully pipelined radix-2 butterfly connected to banked RAMs to support parallel memory accesses.

All uniformly random values are generated with Trivium [16] stream ciphers seeded randomly. A standalone Trivium can generate up to 64 bits per clock cycle without

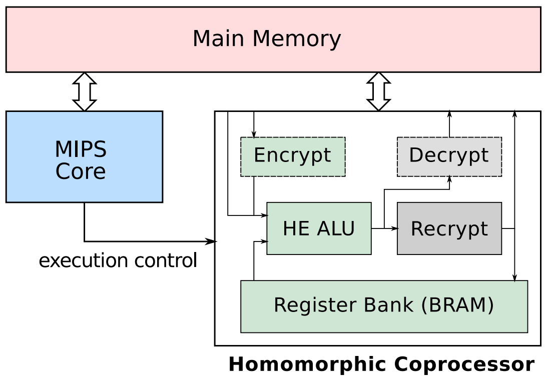


Fig. 3: Integration of an homomorphic element into a SoC

increasing circuit depth. To generate more bits per clock cycle, these circuits are just replicated in parallel.

Another important component required by the BGV is a sampler for Gaussian distributions. The sampler is implemented using an inversion-based method [17] with sufficient buffered output to produce one sample per clock cycle.

B. Results

The design depicted in Figure 3 is successfully synthesized on a mid-range Intel FPGA ArriaV-GX. The system is clocked at 100 Mhz, and the maximum achievable frequency is 128 MHz. This is obtained with the optimization profile set to ”balanced” (the default option) in Quartus.

Synthesis results are given in Table I. The discrete Gaussian sampler is clearly the biggest component, and two of them are required in a single encryption core. The remaining components have a pretty low reconfigurable blocks (called ALM for Intel FPGAs) usage.

This component achieves pretty good performances. Indeed, a homomorphic XOR including refreshing requires $t_{xor} = 309.9\mu s$ and a homomorphic AND requires $t_{and} = 310\mu s$.

TABLE I: Synthesis results on FPGA Arria V GX

Component	Adaptive Logic Module (ALM)	DSP multipliers	BRAM
`NTT`	628 (0.4%)	11 (1%)	68KB (3.1%)
`Gauss`	1881 (1.3%)	-	-
`Encrypt`	7521 (5.49%)	77 (7.3%)	245KB (11%)
`Decrypt`	1866 (1.36%)	55 (5.2%)	68KB (3.1%)
`Recrypt`	9387 (6.85%)	132 (12.5%)	313KB (20%)

C. Performance Estimation on AES Evaluation

We selected the AES algorithm as a case study to evaluate the performances of the proposed architecture. The AES algorithm, is a widely adopted benchmark for homomorphic encryption. Our motivation to explore homomorphic AES evaluation is to perform encryption or decryption, while fully hiding the secret key. First we built an assembly program that evaluates the AES algorithm (as discussed in section II). We applied a very systematic translation manually, which includes replacing substitution boxes (SBox) with their evaluation function, and then lowering every operation into an equivalent boolean circuit. Hopefully, some open-source tools are now available (e.g., Cingulata[1], formally known as Armadillo [18]) and may help to automate the translation process. Moreover, such approach should be able to produce a far more optimized circuits[2] compared to the manual approach we adopted.

We then simulated the execution and measured each instruction count. This allowed us to build the Table II, where we each instruction is associated with its count as well as a runtime estimation based on our synthesis results.

TABLE II: Instruction repartition for an homomorphic AES execution

Instruction	Count	Estimated time
Or	165233	$929.8\mu s$ $(2t_{xor} + t_{and})$
ShiftL	151830	$309.9\mu s$ (t_{xor})
And	110201	$310\mu s$
Xor	72502	$309.9\mu s$
Memory Operation	68770	$4.7\mu s$
ShiftR	67990	$309.9\mu s$ (t_{xor})
Non homomorphic operation	63221	$\approx 10ns$

Putting it all together, we found that with our approach, one can evaluate an AES in 4 minutes using 171 MB of RAM. This compare well with [19] who reported an evaluation time of 14 min (which includes pre and post encryption time) and 3 GB of RAMs on a modern desktop computed and a somewhat homomorphic version of the BGV scheme. A fully homomorphic BGV evaluation requires in the same setting 62 min and 3.7 GB of RAM. The latter should be compared with our results, since we may continue to perform computations afterwards.

Thus, our approach drastically reduces the memory cost in terms of memory (by a factor $\times 30$) and is faster than the fully homomorphic version of [19]. Furthermore, our implementation is likely to be more power-efficient.

IV. Conclusion

In this paper we presented an architecture to protect data confidentiality on systems where memory can be extracted and are subject to passive non-invasive side channel attacks. Such architecture can be seen as a tool to enforce very strong isolation, completely free from information leakages. Our approach benefits the advantages of fully homomorphic encryption. Namely, it provides a convenient reasoning framework about securing arbitrary program execution.

Another contribution of this work was an efficient noise refreshing procedure, as an alternative to bootstrapping. We saw that if one is able to protect very well-defined components of this slightly modified encryption scheme, the whole evaluation chain can be secured. This allows to drastically reduce the parameters and make the scheme as efficient as a somewhat homomorphic one. Finally, this architecture was successfully implemented and evaluated on a FPGA, which demonstrates interesting performances regarding the generality and security achieved.

[1]https://github.com/CEA-LIST/Cingulata

[2]Generating an optimal circuit for a given algorithm is close to a hardware synthesis problem

References

[1] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *CRYPTO'96* (N. Koblitz, ed.), vol. 1109 of *LNCS*, pp. 104–113, Springer, Heidelberg, Aug. 1996.

[2] C. Percival, "Cache missing for fun and profit," 2005.

[3] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: the case of aes," in *Cryptographers Track at the RSA Conference*, pp. 1–20, Springer, 2006.

[4] Y. Yarom and K. Falkner, "Flush+ reload: A high resolution, low noise, l3 cache side-channel attack.," in *USENIX Security Symposium*, pp. 719–732, 2014.

[5] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," *arXiv preprint arXiv:1801.01203*, 2018.

[6] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown," *arXiv preprint arXiv:1801.01207*, 2018.

[7] M. Henson and S. Taylor, "Memory encryption: A survey of existing techniques," *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, p. 53, 2014.

[8] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *41st ACM STOC* (M. Mitzenmacher, ed.), pp. 169–178, ACM Press, May / June 2009.

[9] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachne, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds." Cryptology ePrint Archive, Report 2016/870, 2016. https://eprint.iacr.org/2016/870.

[10] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *ITCS 2012* (S. Goldwasser, ed.), pp. 309–325, ACM, Jan. 2012.

[11] O. Reparaz, R. Clercq, S. S. Roy, F. Vercauteren, and I. Verbauwhede, "Additively homomorphic ring-LWE masking," *Post-Quantum Cryptography*, Jan. 2016.

[12] Y. Ishai, A. Sahai, and D. Wagner, "Private circuits: Securing hardware against probing attacks," in *CRYPTO 2003* (D. Boneh, ed.), vol. 2729 of *LNCS*, pp. 463–481, Springer, Heidelberg, Aug. 2003.

[13] O. Reparaz, S. S. Roy, F. Vercauteren, and I. Verbauwhede, "A masked ring-LWE implementation," in *CHES 2015* (T. Güneysu and H. Handschuh, eds.), vol. 9293 of *LNCS*, pp. 683–702, Springer, Heidelberg, Sept. 2015.

[14] J. Katz and Y. Lindell, *Introduction to modern cryptography*. CRC Press, 2014.

[15] J. M. Pollard, "The fast fourier transform in a finite field," *Mathematics of computation*, vol. 25, no. 114, pp. 365–374, 1971.

[16] M. Rogawski, "Hardware evaluation of eSTREAM candidates," 2007.

[17] N. C. Dwarakanath and S. D. Galbraith, "Sampling from discrete gaussians for lattice-based cryptography on a constrained device," *Applicable Algebra in Engineering, Communication and Computing*, vol. 25, no. 3, pp. 159–180, 2014.

[18] S. Carpov, P. Dubrulle, and R. Sirdey, "Armadillo: a compilation chain for privacy preserving applications." Cryptology ePrint Archive, Report 2014/988, 2014. https://eprint.iacr.org/2014/988.

[19] C. Gentry, S. Halevi, and N. P. Smart, "Homomorphic evaluation of the AES circuit," in *CRYPTO 2012* (R. Safavi-Naini and R. Canetti, eds.), vol. 7417 of *LNCS*, pp. 850–867, Springer, Heidelberg, Aug. 2012.