ÉDITFE
edite-de-paris.fr

p4bl0
pablo.rauzy.name

Télécom ParisTech
telecom-paristech.fr

Midterm PhD Report

Formal Software Methods for Cryptosystems' Implementation Security

Pablo Rauzy
rauzy@enst.fr

December 4, 2013

Introduction

My PhD started in October 2012. Since then, I have been working with Sylvain Guilley and Jean-Luc Danger in the field of *implementation security*. More precisely, I try to increase the use of *formal methods* in this field, at the software level since it is what I know and where I come from in my previous studies. Implementation security is a very young subject (approximately 15 years old), and a very practical one: a lot of industrial research-and-development / engineering participate to the field, which explains why the use of formal methods is not widespread, to say the least. I will explain why and how I aim to spread the use of formal methods, but first I will give a quick overview of what is implementation security.

When we say **implementation security**, we talk about actual, physical implementations, but the security we are talking about is the one of what is implemented[1]. Of course, we work on systems which security is important, namely *cryptosystems*. It is obvious that cryptographic implementations should not leak any information about the secrets that they are trying to protect. What is less obvious however, is that even a perfect cryptographic algorithm can leak information once it is implemented on a physical device, because of the properties of the physical device itself: computing needs resources, and the usage of these resources may give information on the computation. Indeed, so-called *side-channels* such as power consumption, time, temperature, or electro-magnetic radiation may directly depend on the running computation. Attacks exploiting such bias are called *side-channel attacks*. They work very well in practice; for instance, on an unprotected AES cryptoprocessor, it is possible to extract the full key in about 1000 side-channel measurements (refer to `http://www.dpacontest.org/v2/`). Thus, it is mandatory to implement *countermeasures* against them. Side-channel attacks are *passive attacks*, meaning that only observation is needed to carry them out. When the attacker is able to have an impact on the system, it is also possible to conduct *active attacks*. For instance in the case of *fault injection attacks*, the goal is to modify the result

[1] Note however that in some threat-models the security of the implemented system and the physical security of the device which implements it are tied together, but that's not at all what I'm working on.

of the computation to get something which will leak information if the attacker knows how to interpret it. Such attacks can be more or less *invasive*: faults can be injected by voluntarily glitching the clock or the voltage (non-invasive) or by using a laser to modify values in memory or registers at some point in the computation (invasive).

The use of **formal methods** to study these attacks and the countermeasures against them to be able to trust the cryptosystems seems obvious. Yet, their use in our domain is timidly beginning only now. This can be explained by several facts. First, this domain is very practical, and many advances come from the industry rather than the academia, which means that they are more of an engineering effort than research result. Indeed, many countermeasures are developed by trial-and-error until they reach some sort of fixed-point, at which time they are put in production. Most of the time, this is satisfactory enough from an engineering point of view. Also, formal analysis requires a formal model of the studied system, but there is a discrepancy between a proper modelization and the complexity of an actual physical system which may seem like an obstacle.

This covers why I think it is important to spread the use of formal methods in the field of implementation security. I intend my PhD to be a substantial participation toward this end. This means my goals aim to address the reasons why formal methods are not widely used yet. I aim at doing so by lifting several scientific and technological barriers:

- Develop models adapted to the study of side-channel and fault injection attacks and countermeasures, finding ways to avoid the discrepancy between the model and the actual implementation;
- Use these models to develop methods, and the tools which implement them, that are easy to comprehend and use, and "sexy" enough to make people want to use or mimic them;
- Show the necessity of formal methods by using the tools to break, prove, and/or optimize existing countermeasures, thereby improving the state-of-the-art.

In the rest of this report I will present the work I have done since the beginning of my PhD and how I intend to pursue it in the coming months.

Formally Proved Security of Assembly Code Against Power Analysis

I started my work with the study of *power analysis* software countermeasures. My goal was to have a tool which would be able to automatically protect arbitrary assembly code against power analysis attacks, while provably preserving the semantics of the code, and which would output a provably protected code.

Power consumption is traditionally modeled by Hamming weight of values or Hamming distance of updates of values [KJJ99]. This modelization is not perfect but it works well enough in practice and is used to carry out real-world power analysis attacks.

There are two main types of countermeasures against power analysis: "palliative" versus "curative". The two defense strategies are 1. to make the leakage constant, irrespective of the manipulated data (*hiding* or *balancing* [MOP06, Chp. 7] strategy), or 2. to make the leakage as decorrelated from the manipulated data (*masking* [MOP06, Chp. 9] strategy) as possible. The second strategy,

masking, relies on randomness which is a strong requirement and is hard to capture formally. The first strategy, balancing, appears to have a clear invariant (constant leakage) and a software implementation of balancing using *dual-rail with precharge logic* (DPL [TV06]) had been developed by our lab [HDD11]. So I naturally went with the DPL balancing option.

The DPL countermeasure consists in computing on a redundant representation: each bit b is implemented as a pair $(y_{\text{False}}, y_{\text{True}})$. The bit pair is then used in a protocol made up of two phases:

1. a *precharge* phase, during which all the bit pairs are zeroized $(y_{\text{False}}, y_{\text{True}}) = (0, 0)$, such that the computation starts from a known reference state;
2. an *evaluation* phase, during which the pair $(y_{\text{False}}, y_{\text{True}})$ is equal to $(1, 0)$ if it carries the logical value 0, or $(0, 1)$ if it carries the logical value 1.

The DPL has mostly been used as a hardware-level countermeasure, as it was developed as such. However, it is possible to implement it in software, by working at the bit level. By replacing each sensitive instruction with a *DPL macro* which uses a look-up table to compute the same result as the original instruction while respecting the DPL protocol.

Results. I defined a *generic assembly language* and its semantics. It is generic in that it uses a restricted set of very generic instructions that can be mapped one-to-one to and from virtually any actual assembly language. This makes it possible to work with a single assembly language, while still working on the actual implementation, thus avoiding the discrepancy between it and the model. It permitted to develop a tool, called paioli, that is able to automatically DPLize any *bitsliced* assembly code. Many *block-ciphers* are already available bitsliced as it is a common optimization technique [Bih97]. The transformation has been proved to be *semantics preserving*. Another part of the tool does a *symbolic execution* of the resulting code which *statically* verifies that the *security invariant* is well respected. The symbolic execution of the assembly code is carried out using sets of possible values instead of actual values. Each bit of the sensitive data (the *clear text* of the message and the *encryption key*) starts with a value of $\{0, 1\}$ (or their DPL encoded counterparts) and each instruction computes all the possible results given the sets of values of its operands. After each cycle, the security invariant is verified: for each register, memory cell, an address bus that has changed, the Hamming distance between any of its previous possible values and any of its new possible values should be constant. If everything goes well, the code is formally proved to be well-balanced.

Using this tool we were able to produce a provably protected implementation of the *PRESENT* [BKL+07] block-cipher's encryption algorithm. However, this is a software level proof, and when the code is run on actual, non-idealized hardware, not all the bits leak the same amount (some physical bit may consume more power than others). Thus, before DPLizing a code, it is important to *profile* the hardware on which it will be implemented. This profiling allows to choose the two bits which leak the more similarly, which will then be used as the y_{True} and y_{False} in the DPL protocol to guarantee maximum security. I am currently working on this with Zakaria Najm using the DPL balanced PRESENT on an AVR smartcard. Preliminary results are very promising. This work shows that it is feasible to use formal methods in the field of implementation security even when the security properties are physical rather than functional.

Publications. I gave a talk at the 2013 edition of the COSADE conference, and presented a poster at the 2013 edition of the CHES conference. We posted a preliminary version of the paper on the IACR ePrint Archive [RGN13]. Each of these received a warm welcome and attracted the interest of researchers who are already waiting for the final version of the paper to be published. The final paper is soon to be completed and will be submitted to the 2014 edition of the CHES conference.

Future work. I need to clean/rewrite the code and make it usable as it is currently "research code" and no one except me can be expected to use it as is. I will also make an attempt at automated bitslicing of arbitrary assembly code, it will be interesting to explore what can be done automatically with a reasonable complexity. Another interesting path to follow would be to try to find optimizations which do not break the DPL protocol of balanced code, since we have a tool able to statically verify the respect of the security invariant.

Formal Proofs of CRT-RSA Countermeasures Against Bell-CoRe Attacks

Another subject I worked on is the formal proof of countermeasures against the *BellCoRe* fault injection attack on *CRT-RSA* (RSA [RSA78] optimized for memory usage and computation time using the *Chinese remainder theorem*).

RSA is both an *encryption* and a *signature* scheme. It relies on the identity that for all message $0 \leq M < N$, $(M^d)^e \equiv M \mod N$, where $d \equiv e^{-1} \mod \varphi(N)$, by Euler's theorem. For example, if Alice generates the signature $S = M^d \mod N$, then Bob can verify it by computing $S^e \mod N$, which must be equal to M unless Alice is only pretending to know d. Therefore (N, d) is called the private key, and (N, e) the public key.

In CRT-RSA, the private key is a more rich structure than simply (N, d): it is actually a 5-tuple (p, q, d_p, d_q, i_q), where:

- $d_p \doteq d \mod (p-1)$,
- $d_q \doteq d \mod (q-1)$, and
- $i_q \doteq q^{-1} \mod p$.

The algorithm is presented in Alg. 1

Algorithm 1: Unprotected CRT-RSA

Input : Message M, key (p, q, d_p, d_q, i_q)
Output: Signature $M^d \mod N$

1 $S_p = M^{d_p} \mod p$ `// Signature modulo` p
2 $S_q = M^{d_q} \mod q$ `// Signature modulo` q
3 $S = S_q + q \cdot (i_q \cdot (S_p - S_q) \mod p)$ `// Recombination`
4 **return** S

Injecting faults during the computation of CRT-RSA can yield malformed signatures that expose the prime factors (p and q) of the public modulus ($N = p \cdot q$) [BDL97]. If the intermediate variable S_p (resp. S_q) is returned faulted as $\widehat{S_p}$ (resp. $\widehat{S_q}$), then the attacker gets an erroneous signature $\widehat{S}$, and is able to recover p (resp. q) as $\gcd(N, S - \widehat{S})$.

Notwithstanding, computing without the fourfold acceleration conveyed by the CRT is definitely not an option in practical applications. Therefore, many countermeasures have appeared that consist in step-wise internal checks during the CRT computation.

Results. I wrote a tool[2] called finja which works within the framework of *modular arithmetic*, which is the mathematical framework of CRT-RSA computations. The tool allows a full fault coverage of the CRT-RSA algorithm, thereby keeping the proof valid even if the code is transformed (*e.g.*, optimized, compiled, partitioned in software/hardware, or equipped with dedicated countermeasures). The general idea is to represent the computation term as a tree (just like the AST in a compiler or interpreter) which encodes the computation properties. This term is then simplified by our tool. The simplification works like a naive interpreter would, except that it is a pure *symbolic interpretation* using rules from arithmetic in the $\mathbb{Z}$ ring and its $\mathbb{Z}_n$ subrings. The tool also knows about a few theorems, namely *Fermat's little theorem*, its generalization *Euler's theorem*, the *Chinese remainder theorem*, and a particular case of the *binomial theorem*. Fault injections in the computation term are simulated by changing the properties of a subterm, thus impacting the simplification process. The computation is given in a convenient high-level input language. Indeed, the model of the computation can remain as abstract as pseudocode such as it is usually employed in papers, especially for the computational parts: for instance a fault in the implementation of the multiplication (or the exponentiation) is either inoffensive, and we do not need to care about it, or it affects the result of the multiplication (or the exponentiation), and our model takes it into account without going into the details of how the multiplication (or exponentiation) is computed. An attack success condition is also given and used on the term resulting from the simplification to check whether the corresponding attack works on it.

The tool was used to break implementations which were known to be broken and to formally prove two others: that of Aumüller *et al.* [ABF+02], and that of Vigilant [Vig08] (a repaired version by Coron *et al.* [CGM+10]). Prior to this work no existing BellCoRe countermeasures had been proved, except for a specific implementation of the latter [CCGV13], but we found that the repaired version included fixes for weaknesses that did not exist in the original version. Indeed, these weaknesses had been introduced by eager speed-oriented optimizations. We also found that 2 out of its 9 verifications were useless, and that some added security against power analysis actually weaken the fault injections resistance. This work shows the importance of using formal methods in the field of implementation security, not only to be able to really trust cryptosystems, but also to enable speed and security oriented optimizations.

Publications. A first paper [RG13] was accepted at the 2013 edition of the PROOFS workshop and an extended version will appear in a future issue of the JCEN journal. We have submitted a second paper to the 2014 edition of the PPREW workshop.

[2] `http://pablo.rauzy.name/sensi/finja.html`

Future work. The finja tool is only able to inject faults in the data and cannot fault instructions yet, it would be interesting to explore this kind of fault injections [HMER13]. The tool would also benefit a lot from the parallelization of its computations: multiple-fault attacks can take very long to compute and the different possible faults injections are entirely independent.

I will also work with Gilles Barthe and François Dupressoir at the IMDEA lab (Madrid, Spain) in the beginning of 2014 as they showed an interest in my work on fault injection. We will try to use the EasyCrypt [BGZB09] tool that they work on to do the same kind of formal proofs, and also to use program synthesis techniques to automatically find countermeasures.

Sylvain and I will also work on a high-order variation of the Aumüller countermeasure which would be customizable to resist any order (multiple faults) of fault injection attacks.

Other Perspectives

I would like to use finja to formally study fault injection attacks and countermeasures on other cryptosystems than CRT-RSA, and to use paioli to protect at least one other block-cipher, such as AES, on another hardware platform.

Apart from this and the future work I listed for the two subjects that I already started to tackle, I have two ideas that I'd like to investigate:

- try to model the caching behavior of microprocessors and use the same kind of symbolic execution techniques to formally study timing attacks;
- try to use a homomorphic cryptosystem such as the one of Paillier to mask (against power analysis) its own computation.

References

[ABF+02] Christian Aumüller, Peter Bier, Wieland Fischer, Peter Hofreiter, and Jean-Pierre Seifert. Fault Attacks on RSA with CRT: Concrete Results and Practical Countermeasures. In Burton S. Kaliski, Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 260–275. Springer, 2002.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In *Proceedings of Eurocrypt'97*, volume 1233 of *LNCS*, pages 37–51. Springer, May 11-15 1997. Konstanz, Germany. DOI: 10.1007/3-540-69053-0_4.

[BGZB09] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella-Béguelin. Formal certification of code-based cryptographic proofs. In *36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009*, pages 90–101. ACM, 2009.

[Bih97] Eli Biham. A Fast New DES Implementation in Software. In Eli Biham, editor, *FSE*, volume 1267 of *Lecture Notes in Computer Science*, pages 260–272. Springer, 1997.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and Charlotte Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In *CHES*, volume 4727 of *LNCS*, pages 450–466. Springer, September 10-13 2007. Vienna, Austria.

[CCGV13] Maria Christofi, Boutheina Chetali, Louis Goubin, and David Vigilant. Formal verification of an implementation of CRT-RSA Vigilant's algorithm. *Journal of Cryptographic Engineering*, 3(3), 2013. DOI: 10.1007/s13389-013-0049-3.

[CGM+10] Jean-Sébastien Coron, Christophe Giraud, Nicolas Morin, Gilles Piret, and David Vigilant. Fault Attacks and Countermeasures on Vigilant's RSA-CRT Algorithm. In Luca Breveglieri, Marc Joye, Israel Koren, David Naccache, and Ingrid Verbauwhede, editors, *FDTC*, pages 89–96. IEEE Computer Society, 2010.

[HDD11] Philippe Hoogvorst, Jean-Luc Danger, and Guillaume Duc. Software Implementation of Dual-Rail Representation. In *COSADE*, February 24-25 2011. Darmstadt, Germany.

[HMER13] Karine Heydemann, Nicolas Moro, Emmanuelle Encrenaz, and Bruno Robisson. Formal Verification of a Software Countermeasure Against Instruction Skip Attacks. Cryptology ePrint Archive, Report 2013/679, 2013. `http://eprint.iacr.org/`.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In *Proceedings of CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer-Verlag, 1999.

[MOP06] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, December 2006. ISBN 0-387-30857-1, `http://www.dpabook.org/`.

[RG13] Pablo Rauzy and Sylvain Guilley. A Formal Proof of Countermeasures Against Fault Injection Attacks on CRT-RSA. Cryptology ePrint Archive, Report 2013/506, 2013. `http://eprint.iacr.org/`.

[RGN13] Pablo Rauzy, Sylvain Guilley, and Zakaria Najm. Formally Proved Security of Assembly Code Against Power Analysis. Cryptology ePrint Archive, Report 2013/554, 2013. `http://eprint.iacr.org/`.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[TV06] Kris Tiri and Ingrid Verbauwhede. A digital design flow for secure integrated circuits. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 25(7):1197–1208, 2006.

[Vig08] David Vigilant. RSA with CRT: A New Cost-Effective Solution to Thwart Fault Attacks. In Elisabeth Oswald and Pankaj Rohatgi, editors, *CHES*, volume 5154 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2008.