Design and Optimization of the Input Modules of a DPA Toolbox

A. Fuentes Rodríguez,[1], L. Hernández Encinas[1],
A. Martín Muñoz[1] and B. Alarcos Alcázar[2]

[1] *Instituto de Tecnologías Físicas y de la Información (ITEFI)*
Consejo Superior de Investigaciones Científicas (CSIC)
C/ Serrano 144, 28006-Madrid, Spain
{alberto.fuentes, luis, agustin}@iec.csic.es

[2] *Departamento de Automática, Escuela Politécnica Superior*
Universidad de Alcalá (UAH)
Carretera A2, km 32, 28871–Alcalá de Henares, Spain
bernardo.alarcos@uah.es
Contact author: Agustín Martín Muñoz

Abstract. Theoretical security of cryptographic systems does not guarantee their security in practice when those systems are implemented in physical devices. The aim of this work is to present the design and optimization of the input modules of a toolbox to carry out differential power analysis attacks against the physical implementation of a given cryptosystem. Text and Key modules allow to input the plaintext or ciphertext to the targeted cryptographic algorithm and the corresponding hypothetical values about the used key, respectively. Once configured, the toolbox Power Traces module controls a digital oscilloscope which acquires the power traces during the operation of the device and automatically performs the necessary traces alignment. It can also perform statistical operations with the stored values representing the acquired traces. An analysis about different object oriented trace representation options to implement the toolbox is performed and results are presented.

1 Introduction

During the last years the use of different electronic devices which implement cryptographic features to perform different operations (personal identification, payment cards, etc.) has increased worldwide. Many of those uses are relevant enough to require important security guarantees.

It is known that symmetric cryptography uses a unique key (secret) to encrypt and decrypt messages between two parties who want to exchange confidential information, but one of its main problems is to determine what secret key will be used. This problem is solved by using Key Agreement Protocols (KAP). Nowadays, Quantum Key Distribution (QKD) uses the properties of quantum mechanics to guarantee secure communication for KAP. The bits of the key are individually encoded in states of a quantum system (polarisation states of single photons, for instance), and then distributed between the legitimate parties.

Any eavesdropper's attempt to intercept bits of the key implies that some measurements will be performed on the quantum system, unavoidably changing its quantum state and therefore introducing errors which reveal her presence. QKD systems are suitable for optical links [1], [2], whereas for wireless radio communication channels, secure KAP can be achieved by modulation of the information, at physical layer, by the thermal noise experienced by the link between two terminals [3].

On the contrary, asymmetric cryptography does not need key agreement protocols as it uses a pair of keys (public/private) to encrypt/decrypt messages. Until the publication in 1996 of the paper by Kocher [4], the cryptographic community considered that the security of an asymmetric cryptosystem lied in the strength of the mathematical problem in which it was based on. For instance, the security of the RSA algorithm is based on the difficulty of factorizing the RSA composite modulo, which can be achieved by using large prime numbers.

With the quick and widespread development of portable cryptographic tokens, typically smart cards, which usually have limited memory and computational capabilities, several cryptosystems and lightweight protocols [5] have been developed. Elliptic curve cryptosystems, for instance, allow the use of much shorter keys to achieve a level of security similar to that of RSA [6].

1.1 General Concepts about Attacks to Physical Devices

Kocher's work demonstrated that it was possible to break the security of embedded cryptographic systems, even easily, by means of an attack which, instead of trying to solve the underlying mathematical problem, took advantage of the information which could be obtained by the fact that the cryptosystem was physically implemented in a device. Kocher showed that useful information to find fixed Diffie-Hellman exponents, factor RSA keys, and break other cryptosystems could be obtained by measuring the time required to perform private key operations. His work pointed out that timing attacks are a highly meaningful tool for vulnerability analysis. The continuous development of device-implemented cryptography (see, for instance, [7]) is accompanied by an increasing number of physical attacks [8–10].

The information could also be leaked by other *side channels* as could be the power consumption —thus enabling Simple Power Analysis (SPA) or Differential Power Analysis (DPA) attacks [11–13]—, and the electromagnetic field radiated by the device during its operation —enabling Simple ElectroMagnetic Analysis (SEMA) or Differential ElectroMagnetic Analysis (DEMA) [14–16]—. The hypothesis on which side channel attacks are based is that the magnitudes of these characteristics directly depend on the instructions, mathematical operations, and data used by the processor during the cryptographic operations. In this way, the cryptographic key used can be deduced by an adequate analysis of the information obtained by measuring the leakage by these side channels. The procedures to develop both a power analysis and a electromagnetic analysis attack are quite the same; the only difference is the measured magnitude. The reason is that the electromagnetic field radiated by the chip is caused by

the currents circulating within the chip circuits. These currents are especially significant when transistors switch their state [15].

The mentioned attacks, which only capture the information leaked from the device without altering it, are known as passive (non-invasive) attacks. Moreover, active attacks (invasive or semi-invasive) have also been developed to tamper with the correct behaviour of the device in order to obtain secret information; some of these fault attacks can alter or even destroy the device [17], [18].

In physical attacks it is assumed, as in the case of classical cryptanalysis, that Kerckhoffs' [19] principle is verified, that is, the potential eavesdropper has access to the device and knows the cryptographic algorithm that is running in the chip, so as the details of the implementation; the only thing she doesn't know is the key. Furthermore, it is supposed that she can operate the device the number of times she needs, choosing the input values, and obviously, that she can interact with the device or measure certain parameters in its surroundings.

1.2 Example of a SPA Attack

One of the most widely used technique among all side channel attacks is power analysis which employs power consumption traces measured during the operation of the cryptographic device. These traces are captured by using a digital oscilloscope that measures the voltage across the resistor connected in serial to the power source terminal of the device which communicate with the chip.

By means of SPA attacks, the attacker, having a detailed knowledge of the implemented algorithm, succeeds in obtaining the key after capturing a single trace, or a set of a few traces. For example, let us consider the RSA algorithm which executes a modular exponentiation, $y = x^k \mod n$, where the binary representation of the key is $k = (k_{r-1} \dots k_0)_2$. The square and multiply algorithm for the modular exponentiation, processing the bits from left to right, is:

1. $y \leftarrow 1$.
2. For i from $(r-1)$ to 0 do:
   (a) $y \leftarrow y^2 \mod n$.
   (b) If $(k_i = 1)$ then $y \leftarrow (y \cdot x) \mod n$.
3. Return (y).

This way, if $k = 23 = (10111)_2$, then $x^k = x^{23} = (((x^2)^2 x)^2 x)^2 x$. As can be observed, when the algorithm processes a '0' bit, the multiplication included in step 2(b) is not executed and, thus, the power consumption will be lower than what would be needed for a '1' bit. If an attacker knows that a smart card is executing a RSA which implements the above algorithm and, measuring the consumption, she captures the SPA trace represented in Figure 1, she would easily identify the key $k = 653642 = (10011111100101001010)_2$.

Without the adequate countermeasures, this kind of attack is very effective and requires a small amount of resources. If the relationship between the consumed power and the cryptographic key is not clear, the captured signal uses to have a very low level as compared to noise, and SPA attacks are not feasible. In

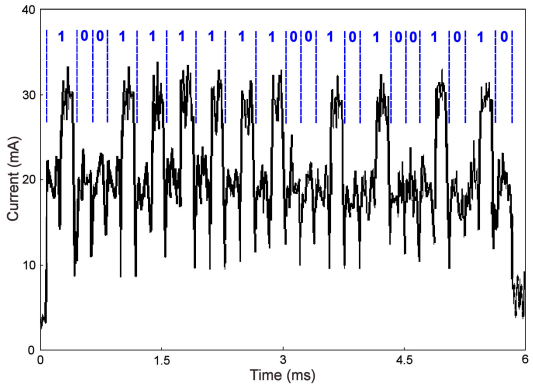

Fig. 1. Power consumption during the execution of a RSA with a key $k = 653642$.

those cases, statistical techniques are used in order to perform a DPA attack, requiring a huge amount of data to be captured and processed.

In this work the design and optimization of the input modules of a software tool to carry out DPA attacks is presented. The initial stages of the toolbox development, describing its storage and alignment capabilities were presented in [20]. This new toolbox is an open source project intended to provide a baseline from where new techniques or approaches could be shared and reviewed. It is oriented to be clear, user-friendly, and easy to understand, in order to easily choose between different known techniques and options without the need of knowing their internal characteristics. Another key feature is that it is modular because, being DPA attacks a very active research field with different aspects (i.e., alignment techniques, statistical analysis, etc.), the toolbox should provide the capability of integrating new findings in an easy way. It should also be as efficient as possible so as to minimize the resources needed to carry out the attack (main memory, cache usage, multithreading programming, etc.), being open to new optimization techniques.

In the following section a description of the general procedure to develop a DPA attack is outlined. In section 3, the toolbox Text, Key and Power Traces input modules are described. Different implementation issues to optimize the toolbox are analyzed in section 4. Finally, results of a performance analysis and conclusions are presented in sections 5 and 6, respectively.

2 Description of a DPA Attack

In order to obtain a secret key, an eavesdropper must have the targeting device and know or guess the model of power consumption (the better the guess, the better the result); it is also assumed that she knows the type of cryptographic algorithms which is being executed, the plaintexts or the ciphertexts, and can measure all the power traces she needs.

To carry out a DPA attack, a large amount of plaintext are ciphered and the corresponding power consumption traces are measured by means of a digital oscilloscope. Those traces are later on, stored and syncronized, that is, an alignment procedure is made with the captured traces. The goal is to guarantee that a correct comparison between the traces is performed (the values of all traces must be compared in their correct time instant, to ensure that they are caused by the same operation) all along the execution of the algorithm [21]. The procedure for a DPA attack is:

1. Choose an intermediate result of the executed algorithm from a function that uses as inputs part of the cryptographic key and known data, usually the plaintext or ciphertext.
2. Create a power profile, measuring the power consumption for D encryption/decription operations with different plaintexts/ciphertexts. For each run the power trace is created with T samples, obtaining a matrix $P_{D\times K}$.
3. Calculate a hypothetical intermediate value for every possible choice of the key k. If K is the total number of possible choices for k, the result of this step is a matrix $V_{D\times K}$.
4. Map the hypothetical intermediate values V to a matrix H of hypothetical power consumption values which are simulated according to a certain power model, commonly the Hamming-distance or the Hamming-weight model [21, §3.3]. These calculations result in a matrix $H_{D\times K}$ which contains these hypothetical power consumption values.
5. Compare the hypothetical power consumption values with the power traces at each position. This results a matrix $R_{K\times T}$ containing the results of the comparison. Different algorithms are used for the comparison (difference or distance of means, generalized maximum-likelihood testing, etc.).

If the statistical method uses a correlation coefficient, the attack is known as Correlation Power Analysis (CPA). This kind of attack was first introduced in [22]. Compared with DPA, CPA requires less number of power traces to launch a successful attack because in DPA all unpredicted data bits contribute to generate a worse Signal to Noise Ratio (SNR). The SNR of DPA could be improved if multiple bits are used in prediction [23].

Figure 2 shows a power trace measured with our PicoScope 5204 digital oscilloscope with a Pintek DP-30HS high-sensitivity differential probe during the execution of a DES in a NXP JCOP41/72k smart card. Thousands of similar traces are captured with our tool and then processed to carry out a DPA attack.

DPA attacks can be generalized to Higher-Order Differential Power Analysis (HODPA), where several points in a power trace are used instead of a single one. Usually, a n^{th}-order DPA attack uses n samples simultaneously, corresponding to n different intermediate values in the same captured trace. HODPA are specially useful to break implementations which include countermeasures [24–26].

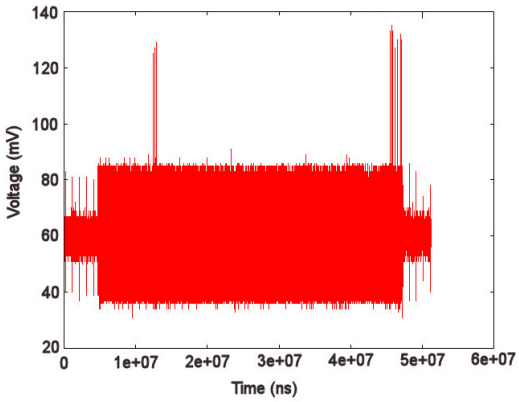

Fig. 2. Power consumption captured during the execution of DES.

3 Toolbox input modules

According to points 1 and 2 of the DPA attack flow described in §2, three main blocks constitute the input to the kernel of a DPA software tool: a set of known data (plaintext or ciphertext depending upon the chosen intermediate result), a set of hypothetical values which constitute part of the key, and a set of power consumption traces measured during the encryption/decryption operation of the cryptographic algorithm where the set of plaintexts/ciphertexts are involved. The main classes created for each of these modules are described next.

3.1 Text block

Classes ***CInputData*** and ***CInputDataSet*** have been created to model the input plaintexts or ciphertexts. Both classes contain methods to add, modify and read data. *CInputDataSet* allows users to choose between adding values specified by themselves or generated at random. No inheritance relationship exists between the text input data.

3.2 Key hypotheses block

This module has two classes to model the key hypotheses. ***CKeyHypothesis*** contains the hypothetical values of the key (or part of the key). As in the case of ***CInputData***, a method to set the hypothetical key size is also called.

CKeyHypothesisSet, whose inheritance diagram is shown in Figure 3, is used as an abstraction to generate the key space. It is implemented as *CKeyHypothesis8Set*, *CKeyHypothesisAllValues8Set*, *CKeyHypothesis16Set*, and *CKeyHypothesisAllValues16Set*, depending on the size of the key (8 or 16 bit, respectively). In *CKeyHypothesisAllValues8Set* and *CKeyHypothesisAllValues16Set* the

set of keys is filled in, at object initialization, with all possible 8 and 16 bit values, respectively. Although it could also be possible to implement a class for a 32-bit key, this option has been discarded because it requires dealing with $4 \cdot 10^{10}$ elements of 32 bits, which implies operating with 128 GB in memory. The generation of the whole key space takes 0.000173 s for 8-bit keys and 0.019521 s for 16-bit keys.

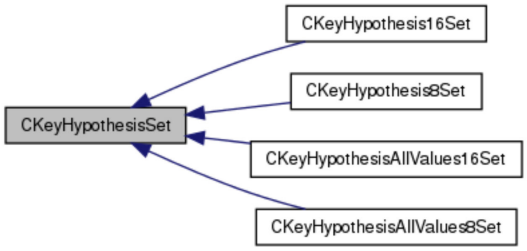

Fig. 3. Inheritance diagram of the CKeyHypothesisSet class.

3.3 Power trace block

This module is in charge of managing the acquisition, storage and alignment of the power consumption traces. It includes the following trace-related classes:

CTrace is used as an abstraction to caller objects of the resolution of the trace. It is implemented as *CTrace8*, *CTrace16*, or *CTrace32*, depending on resolution of the values (8, 16, or 32 bits, respectively).

CStatTrace represents statistical data obtained from several traces (i.e., mean, variance).

CTimeSlice: A time slice contains the values taken at a time point from several traces. This class is used as an abstraction to caller objects of the resolution of the time slice. It is implemented as *CTimeSlice8*, *CTimeSlice16*, or *CTimeSlice32*, depending on resolution of the values.

CTraceSet contains sets of traces of the same device taken with the same timing, in order to do statistical analysis. Trace Sets can be in two states: Statistical Mode and Alignment Mode. Method availability or performance may depend on the mode. It has a subclass, *CPreProcTraceSet*, with the added property that the traces can be preprocessed: aligned, compressed, etc. Traces alignment through least square matching pattern technique is made by the subclass *CAlignMatchSqrTraceSet* —the displacement between the pattern (first trace of the set) and the power trace is calculated—. For traces alignment through integration, subclass *CAlignSumTraceSet* is used —the displacement between the pattern (first trace of the set) and the power trace is calculated—.

A schematic diagram of these classes is shown in Figure 4, where two interface classes, *CFileStorage* (which specifies that the subclasses can be stored in a

file) and *CGraphicable* (which specifies that its subclasses can be printed out if gnuplot is installed), are also included.

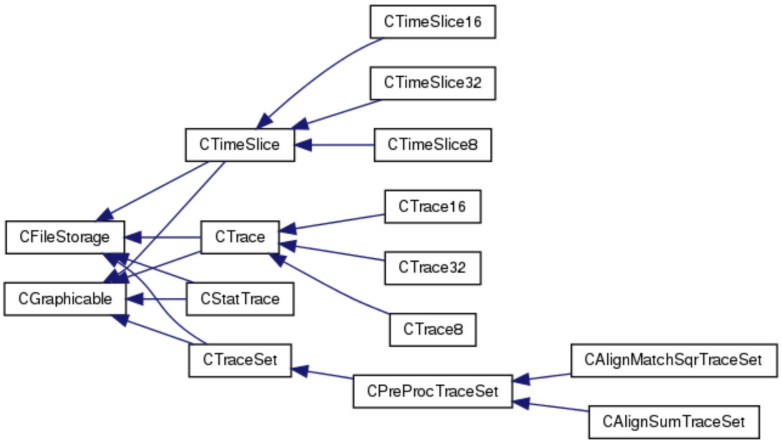

Fig. 4. Hierarchy of classes to manage trace acquisition and alignment.

Depending on the used class, the type of graphical representation provided by *CGraphicable* will be different. As an example, Figure 5(a) shows a plot of 1000 voltage samples —measured by the probe with a sampling rate of 2 ns—, as provided by *Ctrace*, while Figure 5(b) represents the corresponding histogram, number of occurrences of different voltage values in a given time.

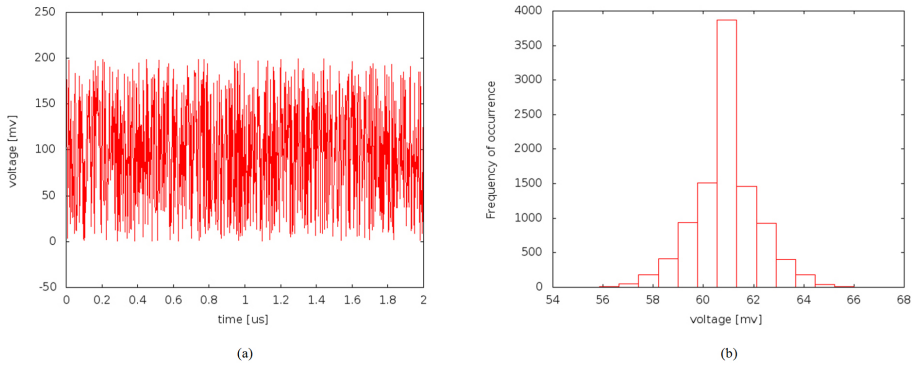

Fig. 5. (a) Power consumption provided by *Ctrace*. (b) Histogram of voltage values in the first instant as provided by *CTimeSlice*.

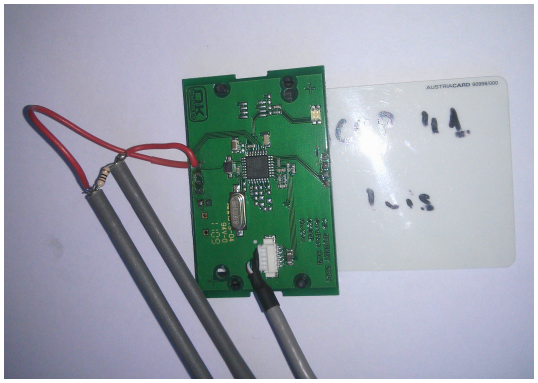

Fig. 6. View of the experimental setup.

Each trace is captured by means of a digital PicoScope 5204 oscilloscope and a Pintek DP-30HS high sensitivity differential probe and is stored as a vector of values for further processing. A picture of the experimental setup showing the card, the reader and the probe ends is shown in Figure 6. The main parameters that have to be properly adjusted when storing the traces are vertical sensitivity, sampling rate, resolution, memory size and DC offset [27]. The size of the elements of the vector depends on the resolution of the voltage values provided by the oscilloscope. Once the capture of the power values is done, data are transferred from the oscilloscope memory to the PC memory. If streaming mode is used, data are transferred during the capture process, although this is only possible with low sampling rates.

Being the data corresponding to the traces the biggest amount of information that the toolbox has to deal with, all the rank of resolutions that the architecture can manage (8, 16 or 32 bits) must be provided. In this way, the toolbox would be as versatile as possible and it would minimize the memory consumption and cache usage (the internal representation of traces is a key factor determining the performance).

As an example, for a sampling rate of 4 ns, a computation process that requires 25 ms by the crypto device would require a trace with $25{\cdot}10^{-3}/(4{\cdot}10^{5}) = 62.5 \cdot 10^{5}$ sampled values. The memory needed to store 1000 traces in an 8-bit architecture is 5.96 MB. The noisiest the signal is, the higher the number of traces that must be measured. Thus, the internal trace representation can determine if a computer can be used to carry out an attack or not.

In order to represent traces as objects, three possible options have been analyzed, either using *C++ templates*, *object oriented inheritance*, or *float values*.

1. **C++ templates**: A class *trace* has been created that operates with a generic type which specifies the resolution (i.e., *uint8_ t* for 8 bits). As the unique abstraction required for trace representation is the resolution of the elements stored, it fits with the C++ template feature.

2. **Object oriented inheritance**: Inheritance in object oriented programming is one of the main building blocks together with encapsulation, polymorphism and abstraction. Inheritance provides an *Is-a* relationship between objects. From this point of view, an "abstract" class *CTrace* has been created. This class has a *derived* class for each resolution option. Using this abstraction mechanism, other classes can use *CTrace* class without the need of knowing which resolution is being used.
3. **Float representation of values**: Keeping the values as float does not require abstraction between the different resolutions. Thus, values are represented as 32-bit floats and the internal values represent the voltage, that is, conversion from raw values to voltage values is done only once, while traces are being captured.

In next section we analize the code optimization of the different mentioned options, although we could tackle others as, for example, plain C by means of Abstract Data Types based on opaque pointers. However, since the library has been designed according to code reutilization principles so further methods and classes could be incorporated upon the proposed programming interface, options that do not provide this feature are not evaluated.

4 Optimization analysis

This section discusses the different implementation options to optimize the toolbox. Using C++ templates, type abstraction is resolved at compile time. Thus, the compilation time is longer but there is not need of solving polymorphism at execution time that could impact the performance, but as it is one of the toolbox objectives this option must be taken into account. Also templates are checked at compile time for type consistency, so it avoids type errors at execution time.

Template definitions cannot be separated into a header (.h) and source file (.cpp) because templates are instantiated at compile time, not at link time. Thus, declaration and implementation must be located in the same file. The solution to this problem is the explicit instantiation. When templates are explicitly instantiated the programmer defines which possible values the type can acquire. With this approach the compiler will create object code for each specified value and neither compiler nor linker errors will appear. Although currently C++ allows implicit template instantiation to be defined in a separated source file by using the *export* keyword, this feature is not allowed by most C++ compilers.

Another important aspect which must be considered about using C++ templates is that the trace class is used to form other classes in the toolbox (i.e., *CTraceSet*). These must be able to deal with all possible type values that the traces may acquire, but the only way to do that is to define such classes as templates as well. This imply that the templates will extend to many parts of the toolbox source code, which would become less clear. As the toolbox is expected to be used and modified by other developers, clearness of source code is essential.

Using inheritance, resolution operations are done at *CTrace* class level, so other classes can be build without taking resolution into account. In this case,

the code is clear and easy to understand, an important advantage with respect to templates. However, abstract methods are solved at execution time. Thus, part of the processing time will be spent in the *dynamic dispatch*.

The option of creating an abstract class representing an unique data element (voltage value) has also been evaluated. In case of using object inheritance, an operation that affects the data elements of a trace requires to call an abstract method for each data element instead of just calling the abstract method of the *CTrace* class. As each call to an abstract method may be solved dynamically, it requires much more computation than in the case of using templates.

As mentioned above, keeping the values as float does not require abstraction between the different resolutions, simplifying source code. In addition, as values stored are the voltage values themselves, conversion from raw to voltage values is done once, while traces are being captured, requiring less computational power than the previously analyzed options. However, the size of internal values is 32 bits. Usually, oscilloscopes offer a maximum resolution of 12 or 16 bits. In those cases the memory and disk spent to store trace information is at least twice the required. Also, storing values with a 32-bit size has a computational disadvantage. Most toolbox operations access trace values sequentially. So, less elements will be found at cache memory and the probability of finding the next value in cache is low. Each time that a value is not found in cache (cache miss), it has to be fetched from its original storage location (RAM memory) which is comparatively slower. So, cache miss delays may counteract the benefits of computing only once the conversion from raw to voltage values.

In Table 1 a summary of the advantages and disadvantages of each object oriented representation option is given.

Table 1. Comparison between the object oriented representation options.

	Advantages	Disadvantages
Templates	Only one class Resolved at compile time Cache optimization	Templates extend over the toolbox Cannot be easily separated in header and source
Inheritance	Abstraction at CTrace level Well known mechanism Easier to understand Less memory and disk required Cache optimization	Solved at execution time Dyamic dispatch
Float	Conversion is done once No abstraction is required Source code simplified	Each value requires 32 bits Less elements at cache memory

5 Performance analysis

Performance has been analyzed by means of a benchmark, *Dyn/Stat*, which has been designed to compare the computational time spent in executing the same operations with the different approaches used to internally represent trace values. The aim of *Dyn/Stat* is to check the performance differences between dynamic binding (binding is solved at execution time) and static binding (binding is solved at compile time). Dynamic binding is done with object oriented inheritance with virtual classes (*Object oriented inheritance* approach), while static binding is done when virtual classes are not used (*C++ templates* approach).

Traces are vectors, so the benchmark checks the performance of storing values in vectors and executing vectorial operations. It has been run in a Intel(R) Core(TM)2 Quad CPU Q9550 @ 2.83GHz with GCC compiler version 4.4.3. *Dyn/Stat* benchmark outputs the time of computation for assigning values to two vectors for both approaches with different vector lengths and compilation parameters. Results presented in Table 2 show that, when `-O0` minimum optimization is used, compilation time is reduced. In this case, the time penalty for dynamic binding represents a 6.60% in case of a vector with 10^7 elements and 6.46% for 10^9 elements. Thus, it can be assumed that the number of elements does not impact in the dynamic binding time penalty. For `-O3` maximum optimization, all compiler optimizations are used. One of these optimizations is `-fdevirtualize` that allows the compiler to convert the dynamic binding into static binding. Devirtualization can be done when the subclass that implements the virtual method can be defined at compilation time [28]. Thus, the virtual (Dynamic in Table 2) solution is 12.67% faster in case of a vector with 10^7 elements and 11.01% for 10^9 elements.

Table 2. Dyn/Stat benchmark output.

	Time (s)			
	`-O0` optimization		`-O3` optimization	
Length (elements)	Static	Dynamic	Static	Dynamic
10 000 000	0.177548	0.189265	0.135924	0.120644
100 000 000	1.776202	1.891002	1.333785	1.201456

It can be pointed out that the system works properly if the number of data to process is increased, i.e., processing time grows linearly. Moreover, when the minimum optimization is used (`-O0`), the virtual solution has a time penalty of about 6.5%, whereas if maximum optimization (`-O3`) is considered, the time spent in computations is about 11–12.7% shorter with respect to the template-based solution, for vectors of length 10^7–10^9.

6 Conclusions

The design and optimization of the input modules of a multi-platform object-oriented toolbox to carry out DPA attacks is presented.

It has been pointed out that, globally, developing the code by using inheritance is better than doing it by using templates. Results show that the design is computationally efficient and has also optimized storage capabilities. In addition, the object-oriented modular design allows new algorithms to be easily implemented.

Acknowledgements. This research was partly supported by both Ministerio de Ciencia e Innovación (Spain) under the grant TIN2011-22668, and Comunidad de Madrid (Spain) under project reference S2013/ICE-3095-CIBERDINE-CM.

References

1. Weier, H.: European Quantum Key Distribution Network. PhD thesis, Faculty of Physics. Ludwig Maximilians Universiät, München (Germany) (2011) http://xqp.physik.lmu.de/publications/files/theses_phd/phd_weier.pdf.
2. García-Martínez, M.J., Denisenko, N., Soto, D., Arroyo, D., Orue, A.B., Fernandez, V.: High-speed free-space quantum key distribution system for urban daylight applications. Appl. Opt. **52**(14) (May 2013) 3311–3317
3. Mucchi, L., Ronga, L., Del Re, E.: A novel approach for physical layer cryptography in wireless networks. Wireless Personal Communications **53**(3) (2010) 329–347
4. Kocher, P.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. Lecture Notes in Comput. Sci. **1109** (1996) 104–113
5. Munilla, J., Peinado, A.: HB-MP: A further step in the HB-family of lightweight authentication protocols. Computer Networks **51**(9) (2007) 2262–2267 (1) Advances in Smart Cards and (2) Topics in Wireless Broadband Systems.
6. Hankerson, D., Menezes, A.J., Vanstone, S.: Guide to elliptic curve cryptography. Springer-Verlag, New York, NY, USA (2004)
7. Wold, K., Petrovic, S.: Behavioral model of TRNG based on oscillator rings implemented in FPGA. In: Proceedings of the 14^{th} IEEE International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS). (April 2011) 163–166
8. Moradi, A., Kasper, M., Paar, C.: Black-box side-channel attacks highlight the importance of countermeasures. Lecture Notes Comput. Sci. (Proceedings of The Cryptographers' Track at the RSA Conference 2012, San Francisco, CA, USA, February 27 – March 2, 2012) **7178** (2012) 1–18
9. De Mulder, E., Örs, S.B., Preneel, B., Verbauwhede, I.: Differential power and electromagnetic attacks on a FPGA implementation of elliptic curve cryptosystems. Comput. Electr. Eng. **33**(5-6) (2007) 367–382
10. Sun, S., Yan, Z., Zambreno, J.: Experiments in attacking FPGA-based embedded systems using Differential Power Analysis. In: Proceedings of the IEEE International Conference onElectro/Information Technology (EIT). (may 2008) 7–12
11. Kocher, P., Jaffe, J., Jun, B.: Introduction to differential power analysis and related attacks. Technical report, Cryptography Research Inc. (1998) http://www.cryptography.com/resources/whitepapers/DPATechInfo.pdf.

12. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. Lecture Notes in Comput. Sci. **1666** (1999) 388–397
13. Kocher, P., Jaffe, J., Jun, B., Rohatgi, P.: Introduction to differential power analysis. J. Cryptograp. Eng. **1** (2011) 5–27
14. Quisquater, J.J., Samyde, D.: A new tool for non-intrusive analysis of smart cards based on electromagnetic emissions, the SEMA and DEMA methods. In: EUROCRYPT2000 Rump Session. (2000)
15. Quisquater, J.J., Samyde, D.: ElectroMagnetic Analysis (EMA): Measures and counter-measures for smart cards. Lecture Notes in Comput. Sci. **2140** (2001) 200–210
16. Quisquater, J.J., Samyde, D.: Eddy current for magnetic analysis with active sensor. In: Proc. of 3^{rd} Conference on Research in SmartCards, E-Smart'02, Nice, France (2002) 185–194
17. Boneh, D., DeMillo, R., Lipton, R.: On the importance of checking cryptographic protocols for faults. Lecture Notes in Comput. Sci. **1233** (1997) 37–51
18. Skorobogatov, S.: Semi-invasive attacks-A new approach to hardware security analysis. PhD thesis, University of Cambridge, Darwin College. UK (2005) http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-630.pdf.
19. Kerckhoffs, A.: La cryptographie militaire. Journal des sciences militaires **IX** (1883) 1–2, 5–38, 161–191
20. Fuentes Rodríguez, A., Hernández Encinas, L., Martín Muñoz, A., Alarcos Alcázar, B.: A toolbox for DPA attacks to smart cards. Advances in Intelligent Systems and Computing (International Joint Conference SOCO'13-CISIS'13-ICEUTE'13) **239** (2014) 399–408
21. Mangard, S., Oswald, E., Popp, T.: Power analysis attacks: Revealing the secrets of smart cards (Advances in Information Security). Springer Science+Business Media, NY, USA (2007)
22. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. Lecture Notes Comput. Sci. **3156** (2004) 16–29
23. Messerges, T., Dabbish, E., Sloan, R.: Examining smart-card security under the threat of power analysis attacks. IEEE Trans. Comput. **51**(4) (2002) 541–552
24. Peeters, E., Standaert, F.X., Donckers, N., Quisquater, J.J.: Improved higher-order side-channel attacks with FPGA experiments. Lecture Notes Comput. Sci. **3659** (2005) 309–323
25. Muller, F., Valette, F.: High-order attacks against the exponent splitting protection. Lecture Notes Comput. Sci. **3958** (2006) 315–329
26. Standaert, F.X., Veyrat-Charvillon, N., Oswald, E., Gierlichs, B., Medwed, M., Kasper, M., Mangard, S.: The world is not enough: Another look on second-order DPA. Lecture Notes Comput. Sci. **6477** (2010) 112–129
27. Fuentes Rodríguez, A., Hernández Encinas, L., Martín Muñoz, A., Alarcos Alcázar, B.: Diseño de un conjunto de herramientas software para ataques por canal lateral. In: Libro de Actas del VII Congreso Iberoamericano de Seguridad Informática, CIBSI'2013. (2013)
28. Namolaru, M.: Devirtualization in GCC. In: Proceedings of the GCC Developers' Summit. (2006) http://ols.fedoraproject.org/GCC/Reprints-2006/namolaru-reprint.pdf.