

NESSIE

Project Number	IST-1999-12324
Project Title	NESSIE
Deliverable Type	Report
Security Class	Public
Deliverable Number	D06
Title of Deliverable	Methodology for Comparing the Performance of Primitives on a Fair and Equal Basis
Nature of the Deliverable	
Document Reference	NES/DOC/UCL/WP4/D06/1
Contractual Date of Delivery	Y1 M09
Actual Date of Delivery	Y1 M10
Editors	Mathieu Ciet, François Koeune, Gilles Piret, Jean-Jacques Quisquater, Francesco Sica
Abstract	This deliverable defines a methodology to compare in a fair and acceptable way the performance of the submitted primitives.
Keywords	
Disclaimer	The information in this document is provided as is, and no guarantee or warranty is given or implied that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

Contents

1	Introduction	3
2	Test Platforms	3
2.1	Smart Cards	3
2.2	Home PC	3
2.2.1	Hardware	3
2.2.2	Operating System	3
2.2.3	Compiler	4
2.2.4	Assembly Language	4
2.3	64-bit Architecture	4
2.3.1	Theoretical Considerations	4
2.3.2	Practical Implementation	4
2.4	Hardware Implementation	4
3	Performance	5
3.1	Smart Cards	5
3.2	PC, 64-bit machine,	5
3.3	FPGA	5
4	Resistance to Attacks	5
5	Dissecting Algorithms	6
5.1	Frequent Rekeying	6
5.2	Occasional Rekeying	6
5.3	No Rekeying	6
6	Non constant-time algorithms	6
6.1	Data or key dependence	6
6.2	Asymmetry	6
6.3	Security Parameters	6
7	Specific Testing	7
7.1	Platforms	7
7.2	Scenarios	7
8	How are the results combined?	7
9	Remarks	8
9.1	Precision	8
9.2	Implementation Strategy	8
9.3	Speed Measurement	8
9.4	Further Questions	9

1 Introduction

Here is the deliverable D6 of the NESSIE project ; it will define a methodology to compare in a fair and acceptable way the submitted primitives. It may evolve according to remarks of the Nessie members or Industry Board, and with problems encountered.

2 Test Platforms

Due to their wide variety, we had to select only some of the possible platforms to evaluate the different primitives. Our choices were largely guided by the particular competences of Nessie members, and the availability of these resources. Reactions of the Industry Board with regard to this point are particularly welcome.

2.1 Smart Cards

We will analyse the implementations on three models of smart cards.

- Low-cost smart cards : two main choices were possible : 8051 and 6805. We chose to use the 8051.
- Powerful smart cards, without cryptoprocessor : we propose to use an ARM-based smart card.
- Powerful smart cards, equipped with cryptoprocessor : we have no suggestions. We welcome any proposal.

2.2 Home PC

We take into account several variables in the implementation on a 32-bit PC.

2.2.1 Hardware

It is important to fix such hardware details as cache size, CPU type, clock frequency ..., since small differences in these may have an important impact on performance. Comparisons must therefore be performed using exactly the same hardware. The precise reference platform will be chosen later according to material available to each Nessie partner.

2.2.2 Operating System

- Due to its widespread use, a Windows platform will be taken as reference.
- A "Unix-like" reference also seems important to be taken into account. We chose Linux, but FreeBSD / OpenBSD / Solaris would be possible too.

2.2.3 Compiler

- The MSVC++ 6.0 compiler will be used on computers running Windows.
- Using Linux, we will use gcc 2.95.2.

Note that if a submitter suggests to use another particular compiler to run his algorithm, this will be taken into account (this was the case, in the AES process, when Mars' authors stated that their candidate ran much slower on the reference Borland compiler than on a Gnu-C one).

2.2.4 Assembly Language

Assembly language is by far the most relevant implementation when performance is really a concern, and will be the only one considered in high-demanding contexts.

2.3 64-bit Architecture

A more powerful machine, based on 64-bit architecture, may be more difficult to choose, since existing machines (DEC Alpha, UltraSparc, PA-RISC and MIPS) are quite different. However, since this 64-bit architecture may become widespread in a near future, it is very interesting to test the primitives in this context.

A new 64-bit Pentium IV (Itanium) is also of interest but since it is not likely to be available before the end of 2001, we feel it is a little premature to use this type of processor.

2.3.1 Theoretical Considerations

Algorithms will have to be paper-and-pencil analysed to predict how they can adapt to a 64-bit architecture. In view of the variety of machines, flexibility must be considered as a major advantage. During WP4, this analysis could suffice for performance evaluation.

2.3.2 Practical Implementation

Due to the high uncertainty as to the nature of the "machine of the future" we recommend to use only one OS and one compiler. These will be chosen later, as well as the machine used.

2.4 Hardware Implementation

Pure hardware implementation is technically difficult to perform (and to perform correctly) for each candidate. As a compromise, we will consider semi-hardware implementations (FPGA's). In WP4, we feel it is enough to study the feasibility of such implementations (implementation of critical sections in VHDL, synthesis using tools like Cadence or Synopsis, CLB count, . . .), without performing them completely.

3 Performance

Depending on the platform, performance measurements will take different parameters into account.

3.1 Smart Cards

In decreasing order of importance we have:

- RAM usage,
- Speed,
- Code size.

Of course we have to consider these parameters as a whole (e.g. accept for a much faster application to be slightly more memory-consuming). However, especially for low-cost smart cards, RAM usage will be a very important criterion.

3.2 PC, 64-bit machine,...

Speed is the main concern. RAM, code size are not important, provided they remain within reasonable limits and their impact on performance is taken into account (e.g. too big tables won't fit in cache).

3.3 FPGA

The parameters we propose are:

- throughput,
- latency,
- chip area,
- power consumption.

4 Resistance to Attacks

When this does not hinder performance too much, applications should be constant-time (we think of blockciphers). In other cases and for other attacks (SPA, DPA,...), ad-hoc policies will be used to attain a fair and equivalent degree of resistance for each candidate, since it is difficult to design generic ones.

5 Dissecting Algorithms

In each algorithm, we will consider different parts: setup (independent of key and data), precomputations (independent of data, e.g. key schedule) and the primitive itself (that must be repeated for every use).

Three scenarios will be considered (ability to adapt to other ones will be considered as a major advantage):

5.1 Frequent Rekeying

Precomputations must be redone (e.g. key changes) for every block.

5.2 Occasional Rekeying

We will process data by chunks of 10Kbytes, with precomputation being re-done between chunks.

5.3 No Rekeying

We will process 1Mbyte of data, without any new precomputation.

6 Non constant-time algorithms

An algorithm may be non constant-time for the following three reasons:

6.1 Data or key dependence

Data dependence will be handled by considering performance in the mean case. Key dependence will be handled by considering performance in the mean case and the worst cases, except for the most improbable ($< 2^{-40}$ say) worst cases.

6.2 Asymmetry

This is the case when encryption and decryption have different speeds. In this case, we shall consider the two figures separately, as there are cases where a slow device has to perform only one of them.

6.3 Security Parameters

In the beginning, we will base ourselves on the values suggested by authors. However, it's clear that performance is highly dependent on the security parameters. As a consequence, we will as soon as possible use values recommended by security evaluation (high interaction with WP3).

7 Specific Testing

All tests are not relevant to all the types of algorithm. For example a low-end smart card is probably not powerful enough to perform a signature. We will thus limit ourselves to the following contexts.

7.1 Platforms

Low-cost smart cards will be used as test platforms only for

- Blockciphers,
- MAC's,
- Hashing,
- Streamciphers,
- PRNG,
- Identification schemes.

Regarding other platforms, all primitives will be considered.

7.2 Scenarios

No-rekeying scenarios will not be considered on a smart card, since it doesn't make much sense to use a smart card to encrypt, sign, etc. a huge amount of data.

The following scenarios will be considered:

	Block Cipher	MAC	Signature	Stream Cipher
Frequent rekeying	Y	Y	Y	Y
Occasional rekeying	Y	Y	Y	Y
No rekeying	Y	Y	N	Y
	Public-key Encryption	PRNG	Identification Scheme	
Frequent rekeying	Y	N	Y	
Occasional rekeying	Y	Y	N	
No rekeying	Y	Y	N	

8 How are the results combined?

In order to make a choice between different candidates, we should opt for those which are:

- at least acceptable on each platform,
- performing above the average on most platforms where there is a significant difference,

- flexible, because test platforms only reflect current architectures, and it is difficult to predict what future machines will look like.

Note finally that the performances obtained for each platform will not necessarily have the same importance (e.g. what is the importance of speed on a smart card as opposed to a PC ?). The input of the industry board will be used for this purpose.

9 Remarks

9.1 Precision

The goal of performance evaluation is to measure speed *range*. Small differences (< 10%) will not be considered as significative.

9.2 Implementation Strategy

The code provided by submitters will be used as a starting point. It will, for example, receive a careful examination and possibly a recoding will be done to avoid advantaging good implementers rather than good algorithms. Algorithm documentation will be widely used, to make sure “speed-up” tricks proposed by submitters are taken into account.

On the other hand, obscure tricks will be ignored. We will not accept “as it is” an assembly language implementation that proves to be fast, but for reasons which are impossible to explain. In the same way, self-modifying codes will not be accepted.

In accordance to the guidelines for WP4, we shall not perform full implementations; only the critical parts will be implemented. Full implementation will be made later(WP6).

9.3 Speed Measurement

Speed measurement will not yield any problem with smart cards (direct clock cycles count).

In multitask environments, we will try to minimize the consequences of multitasking (i.e. as few other processes running as possible, etc.). Several strategies are possible:

- repeat execution several times (with same key and data) and take the “minimum” among execution times,
- use tools like Truetime, Visualcoverage (cf. [5]). Problem: do they take cache size, etc. into account? Isn’t their sole presence disturbing the measure?
- use a virtual PC to simulate execution. Problem : idem.

- To exclude the influence of multitasking under Windows 95/98 we may start up the computer under DOS, and use (Watcom) C in combination with a DOS-extender. Timings are done by reading out the processor's cycle count registers. On Pentium computers we could obtain reproducible results that way, and, in the case of assembly written code, corresponding to the cycle count of the code ¹.

We leave the door open for other suggestions. In particular, should we also consider the scenario of a heavily-loaded server (with many context switches etc.)?

9.4 Further Questions

Other questions will certainly appear during evaluation. They will be widely discussed and answers disseminated, to make sure the same comparison rules are applied everywhere.

¹Suggested by A. Bosselaers

References

- [1] E. Biham, A. Bosselaers, O. Dunkelman and al. *Comments by the NESSIE Project on the AES Finalists*, internal report of the NESSIE project, 24/05/2000.
- [2] S.D. Brown, R.J. Francis, J. Rose and Z.G. Vranesic. *Field-Programmable Gate Arrays*. Kluwer Academic Publishers, 1997.
- [3] A.J. Menezes, P.C. van Oorschot and S.A. Vanstone. *Handbook of applied cryptography*. CRC Press, 1997.
- [4] <http://csrc.nist.gov/encryption/aes/>
- [5] <http://www.numega.com/>
- [6] *Proceedings of the second advanced encryption standard candidate conference*, Mar 22-23 1999.
- [7] *Proceedings of the third advanced encryption standard candidate conference*, Apr 13-14 2000.
- [8] Thomas S. Messerges. Securing the AES finalists against Power Analysis Attacks. In the *Preproceedings of Fast Software Encryption Workshop 2000*, Apr 10-12 2000.