



Project Number	IST-2000-12324
Project Title	NESSIE
Deliverable Type	Public Report
Security Class	Public
Deliverable Number	D14
Title of Deliverable	Report on the Performance Evaluation of NESSIE Candidates I
Document Reference	NES/DOC/UCL/WP4/D14/2
Contractual Date of Delivery	Y2 M6
Actual Date of Delivery	Y2 M9
Editors	Mathieu Ciet and Francesco Sica
Abstract	A preliminary performance assessment of cryptographic primitives submitted to the NESSIE project is given in this deliverable.
Keywords	NESSIE, Performance Evaluation.

Version 3.3

November 20, 2001

Report on the Performance Evaluation of the NESSIE Candidates [†]

B. Preneel¹, B. Van Rompay¹,
L. Granboulan², G. Martinet²,
M. Dichtl³, M. Schafheutle³, P. Serf³,
A. Bibliowicz⁴, E. Biham⁴, O. Dunkelman⁴,
M. Ciet⁵, J-J. Quisquater⁵, F. Sica⁵

November 20, 2001

[†]The work described in this report has been supported by the Commission of the European Communities through the IST program under contract IST-1999-12324. The information in this document is provided as is, and no 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.

¹Katholieke Universiteit Leuven, Dept. Elektrotechniek-ESAT/COSIC, Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium

²École Normale Supérieure, Département d'Informatique, 45 Rue d'Ulm, Paris 75230 Cedex 05, France

³Siemens AG, Otto-Hahn-Ring 6, München 81732, Germany

⁴Technion, Computer Science Dept., Haifa 32000, Israel

⁵Université Catholique de Louvain, Dept. ELEC, Place du Levant 3, B-1348 Louvain-la-Neuve, Belgium

Executive Summary

Performance Evaluation I

(NESSIE Deliverable D14)

This document reports on the results of NESSIE workpackage Performance Evaluation I (WP4).

The goal of this first phase of Performance Evaluation is to provide an initial assessment of performance of the NESSIE submissions to support the choice of primitives that are to be retained for further analysis in the second stages of Security and Performance evaluation (WP5 and WP6). The requirement is to identify those submissions with such problems as to disqualify them from consideration as NESSIE recommendations, and so to eliminate them from the ongoing selection process.

The results given here have been updated and finalised following the NESSIE Workshop and Industry Board meeting, 12/13 September 2001 and subsequent comments received. Performance criteria are defined that provide discrimination between levels of performance assessment. A number of 'standard' algorithms are adopted as benchmarks to aid comparability.

The evaluation environment consists of: - software measurement tools provided by NESSIE WP2 - Toolbox development; - standard instrumentation provided by the platforms; - a representative range of computing platforms (Pentium PCs, Compaq Alpha, SPARC); - a standard template to identify and document critical characteristics and computational operations influencing performance.

The performance analysis comprises two components: - theoretical analysis of the primitives, based on the template data; - practical measurement of actual implementations running on the computing platforms.

Some very substantial discrepancies are observed between the practical measurements on the submissions. This is partly attributable to inherent performance characteristics, and part to implementations ranging from near-optimal to fairly crude, despite attempts to normalise and to obtain or to generate good quality code wherever possible.

The theoretical work is less susceptible to this variability, but nevertheless judgments must be made about the weightings given to computational operations on the various platforms. The results provide a meaningful comparison of underlying performance characteristics. The analysis also gives an indication of the degree of optimization in the implementations and of the applicability to the platforms under consideration.

Some of the primitives were especially designed for use on a smart card. Our platforms, together with the theoretical analysis, provide a first performance evaluation by which we can compare similar primitives.

The combination of the theoretical and practical processes provides an interim assessment that will be used in conjunction with the security evaluation to select the submissions to be retained for further evaluation.

Contents

Executive Summary	i
1 Introduction	1
2 Performance Evaluation Methodology	2
2.1 Performance Criteria	2
2.2 Comparison with Standards	2
3 Theoretical Analysis	3
3.1 Work Description and Justification	3
3.2 Template for Performance Evaluation	3
3.3 Template Results	4
3.3.1 Block Ciphers	4
3.3.1.1 CS-cipher	6
3.3.1.2 Hierocrypt-L1	7
3.3.1.3 IDEA	8
3.3.1.4 Khazad	9
3.3.1.5 Misty1	10
3.3.1.6 Nimbus	11
3.3.1.7 NUSH legacy version	12
3.3.1.8 SAFER++ legacy version	13
3.3.1.9 Anubis	14
3.3.1.10 Camellia	15
3.3.1.11 Grand Cru	16
3.3.1.12 Hierocrypt-3	17
3.3.1.13 Noekeon	18
3.3.1.14 NUSH	19
3.3.1.15 Q	20
3.3.1.16 RC6	21
3.3.1.17 SAFER++	22
3.3.1.18 SC2000	23
3.3.1.19 SHACAL	24
3.3.1.20 Rijndael	25
3.3.1.21 DES/triple-DES	26
3.3.1.22 Conclusion	29
3.3.2 Synchronous Stream Ciphers	29
3.3.2.1 LEVIATHAN	31
3.3.2.2 LILI-128	32
3.3.2.3 SOBER-t16 and SOBER-t32	33
3.3.2.4 SNOW	35
3.3.2.5 BMGL	36

3.3.2.6	Arcfour	37
3.3.2.7	Conclusion	38
3.3.3	Message Authentication Codes	39
3.3.3.1	Two-Track-MAC	39
3.3.3.2	UMAC	41
3.3.4	Collision Resistant and one-way Hash Functions	42
3.3.4.1	Whirlpool	42
3.3.4.2	SHA-1	44
3.3.4.3	SHA-256	45
3.3.4.4	SHA-512 and SHA-384	46
3.3.4.5	Conclusion	47
3.3.5	Asymmetric Encryption Schemes	49
3.3.5.1	ACE Encrypt	49
3.3.5.2	ECIES	51
3.3.5.3	EPOC-1	52
3.3.5.4	EPOC-2	54
3.3.5.5	EPOC-3	56
3.3.5.6	PSEC-1	58
3.3.5.7	PSEC-2	60
3.3.5.8	PSEC-3	62
3.3.5.9	RSA-OAEP	64
3.3.5.10	Summary	66
3.3.6	Asymmetric Digital Signature Schemes	68
3.3.6.1	ACE Sign	68
3.3.6.2	ECDSA	70
3.3.6.3	ESIGN	71
3.3.6.4	RSA-PSS	73
3.3.6.5	FLASH	75
3.3.6.6	SFLASH	77
3.3.6.7	QUARTZ	79
3.3.6.8	Summary	81
3.3.7	Asymmetric Identification Schemes	82
3.3.7.1	GPS	82
4	Claimed Performance	83
4.1	Block Ciphers and MACs	83
4.1.1	CS-Cipher	83
4.1.2	Hierocrypt-L1	83
4.1.3	IDEA	83
4.1.4	Khazad	84
4.1.5	MISTY1	84
4.1.6	NUSH	84
4.1.7	DES	85

4.1.8	Anubis	85
4.1.9	Camellia	85
4.1.10	Grand Cru	85
4.1.11	Hierocrypt-3	86
4.1.12	Noekeon	86
4.1.13	NUSH	86
4.1.14	RC6	86
4.1.15	SAFER++	87
4.1.16	SC2000	87
4.1.17	Rijndael	87
4.1.18	Shacal	87
4.1.19	Two-Track MAC	88
4.1.20	UMAC	88
4.2	Asymmetric Primitives	89
4.2.1	Asymmetric Encryption Schemes	89
4.2.2	Asymmetric Digital Signature Schemes	89
4.2.3	Asymmetric Identification Schemes	89
5	Practical Software Implementation	90
5.1	Measurements for Symmetric Primitives	90
5.1.1	Description of Tools	90
5.1.1.1	Block Ciphers Tests	90
5.1.1.2	Stream Ciphers Tests	90
5.1.1.3	Tests for MAC and Hash Functions	90
5.1.1.4	Conclusion	91
5.1.2	Block Ciphers Testing	92
5.1.3	Stream Ciphers Testing	93
5.1.4	MAC and Hash Functions Testing	94
5.2	Measurements for asymmetric primitives	95
5.2.1	Asymmetric Encryption Schemes	95
5.2.2	Asymmetric Digital Signature Schemes	96
5.2.3	Asymmetric Identification Schemes	96
6	Comparisons Between Candidates	98
6.1	Block Ciphers	98
6.2	Stream Ciphers	98
6.3	Message Authentication Codes and Hash Functions	98
6.4	Asymmetric Encryption Schemes	99
6.5	Asymmetric Digital Signature Schemes	99
A	Elliptic Curve Cryptosystem	101

List of Figures

1	Data lengths	81
2	Block ciphers (64-bit), without CS-cipher and Hierocrypt-L1: encryption/decryption results	102
3	Block ciphers (128- and 160-bit) without Grand Cru and SC2000: encryption/decryption results	103
4	Stream ciphers and MACs' performance (without UMAC): key setup . . .	104
5	Stream ciphers performance (without BMGL): key stream generation . . .	105
6	MACs' and hash functions performance	106
7	Digital signature : time for key setup in seconds	107
8	Digital signature : time for signature in seconds	107
9	Digital signature : time for verification in seconds	107

List of Tables

1	Block ciphers template results	30
2	Stream ciphers template results: key setup	38
3	Stream ciphers template results: key stream generation	38
4	Message authentication schemes and hash functions template results	47
5	Usual parameters lengths (in bytes)	66
6	Ciphertext Length (in bytes) (message length = 16 bytes)	66
7	Data lengths	81
8	Claimed performance of asymmetric encryption schemes.	89
9	Claimed performance of digital signature schemes	89
10	Claimed performance of asymmetric identification schemes	89
11	Block ciphers performance results	92
12	Stream ciphers performance results	93
13	MACs' and hash functions performance results	94
14	Estimated performance of asymmetric encryption schemes	95
15	Time needed in seconds on the Pentium III for a 20 bytes message.	96
16	Description of GPS Test Platforms	97
17	Performance times for GPS	97

1 Introduction

NESSIE (New European Schemes for Signature, Integrity, and Encryption) is a research project within the Information Societies Technology (IST) Programme of the European Commission. The participants of the project are:

- Katholieke Universiteit Leuven (Belgium), coordinator;
- Ecole Normale Supérieure (France);
- Royal Holloway, University of London (U.K.);
- Siemens Aktiengesellschaft (Germany);
- Technion - Israel Institute of Technology (Israel);
- Université Catholique de Louvain (Belgium); and
- Universitetet i Bergen (Norway).

NESSIE is a 3-year project, which started on 1st January 2000. The main objective of the project is to put forward a portfolio of strong cryptographic primitives obtained after an open call and evaluated using a transparent and open process. The evaluation concerns both the security and the performance aspects of the primitives. This report presents the state of Performance Evaluation work at the end of its first phase.

Detailed and up to date information on the NESSIE project is available at the project web site: <http://cryptonessie.org>.

Deliverable D14 is the final document of Performance Evaluation I. It provides a detailed review of all performance procedures and results undertaken up to August 2001 and serves, along with D13 [8], in the selection of submissions retained in phase 2.

The report is organised as follows. After this brief introduction, we discuss the value and necessity of performance evaluations. The main motivation arises from industrial applications, some of which are described. We define the performance criteria that will help us sieve candidates into phase two, and we select standard algorithms that will serve as reference benchmarks for performance evaluation.

In the second part, we describe and analyse theoretical results of the candidates using a template identifying and describing the number and nature of basic operations as well as other variables which will influence performance. This pencil and paper analysis enables us to decide the degree of optimization, based on an actual implementation, and to judge malleability and aptitude to different architectures and platforms.

In the third part, we focus on software implementations. After describing test platforms, we introduce a set of tools developed by the NESSIE team to produce fair and reproducible performance measurements of all candidates, on all our platforms. We also provide working versions for all candidates that conform to the published NESSIE requirements. We have endeavoured to provide a consistent and comparable degree of optimisation for all the (C) sourcecode used for performance measurement. Once done this allows us to run a fair

performance evaluation of all candidates and a to measure them against each other and against our chosen standards. The results will be further validated by a cross analysis with previous theoretical results.

The output of this process will allow us to make a judicious choice of the candidates. They are analysed together with inputs from the security evaluation [8] to determine which candidates are retained for the second phase of performance and security evaluation.

2 Performance Evaluation Methodology

Performance evaluation is an essential part in determining the practicality of a cryptographic algorithm. A good performing algorithm stands better chances to be used.

The candidates will be used on several platforms, for example PCs, smart cards... and for various other purposes. Some applications impose very high performance requirements, such as hard disk encryption, and protection of high speed communications (Gigabit networks). For others, an acceptable performance is required in a low-end hardware platform and/or in compact hardware (cellular phone, smart card).

Consequently, evaluation of speed and size of code and hardware is important and mandatory for all candidates which meet the security requirements; performance evaluation may provide the means to identify the best suited ones

In general, we will retain the candidates which are flexible on more than one platform and those which perform above average on a particular platform.

2.1 Performance Criteria

Since the platform tests considered up to now are PC (under Windows and Linux), and Unix machines, speed is the main concern. We measure speed in the most uniform possible way, that is using clock counts as measured by the C `clock` function.

2.2 Comparison with Standards

The NESSIE project also takes into account existing and emerging standards, even if these have not been formally submitted to the NESSIE project. Two recent examples in this context come from the standardisation efforts run by NIST. The NESSIE project has contributed extensive comments to the AES process, and Vincent Rijmen, one of the designers of the AES algorithm ‘Rijndael’ has been a member of the NESSIE project team. It is therefore clear that in the evaluation of block ciphers, the Rijndael algorithm will be used as a benchmark. While it is not possible to anticipate the NESSIE results, one can expect that the research by NESSIE on the security of block ciphers may well increase the confidence in and acceptability of the Rijndael algorithm as a standard. The NESSIE project will also study the security and performance of SHA-256, SHA-384 and SHA-512, the new hash algorithms proposed by NIST to extend the result of SHA-1 to hash results between 256 and 512 bits.

The speed performance of candidates will be compared to the following well-known algorithms:

- DES and triple-DES for 64-bit block ciphers,
- Rijndael, the FIPS standard for 128-bit block ciphers,
- Arcfour, such as distributed in the OpenSSL package, for synchronous stream ciphers,
- CBC-MAC with Rijndael as underlying block cipher for MACs,
- SHA-1, SHA-256, SHA-384 and SHA-512 for hash functions.

3 Theoretical Analysis

3.1 Work Description and Justification

Theoretical analysis has been carried out in the first stages of performance evaluation. It consisted in evaluating the degree of optimisation of the submitted C codes in view of a fair comparison of implemented primitives. This was achieved by dissecting each algorithm and comparing the complexity of a given algorithm to its tested speed. We used a template to produce this result. We were later involved in optimising all the algorithms in order to perform a fair machine evaluation.

More specifically, the order of performance speed can be evaluated in this way for block ciphers, stream ciphers, MAC's, hash functions. However since asymmetric schemes may rely on operations that are not standard, their theoretical analysis will be predominant.

3.2 Template for Performance Evaluation

The rationale to have a common template for each primitive is to allow us to identify their basic performance characteristics in an absolute manner as simply as possible. In this way we are able to judge the candidates complexity with respect to others of the same category. Here is how the template is laid out.

The technical specification requested are:

- candidate name, word size, key size, memory requirements, code size.

Then follows a description of the basic operations involved in running the primitive. They are:

- For symmetric primitives: Shifts/Rotations, table look-ups, permutations, multiplications, additions/XOR/AND (with word size),

- For asymmetric primitives: modular reduction, modular exponentiation, multiplication by an integer, modular multiplication, modular inversion (with parameter size). This list is not complete, since some asymmetric primitives (FLASH, SFLASH, QUARTZ) perform a totally different kind of operations.

Next the template asks for a first performance evaluation on a PC/UNIX machine. This is a raw figure which reflects only the submitted implementation. Nevertheless, one can already infer up to this point the implementations that will need to be reconsidered for optimisation; this question is indeed raised at the end of the template. At this first stage we want to consider whether the coding has trivial defaults which render them non-optimised. The template ends with a table on data- or key-dependent running times (which will have to be reconsidered later).

How can we judge optimisation from description and performance? It is right that this method may sound somewhat empirical, however, once trivial optimisation has been carried out as described above in the first stage, this is a valid way of producing fair estimates for all primitives.

Thus, the role of the template is to assess the degree of code optimisation. Suboptimal code was consequently optimised in an iterative process.

Note however that this description of performance tests with coding and results passing back and forth is only applied to symmetric primitives. Performance evaluation of asymmetric primitives is mainly based on the theoretical description gathered from the template. Indeed the number of operations is low, compared to symmetric primitives and their nature and speed is well understood (for instance it is well-known how much time a modular multiplication may take). The only exceptions in this field are QUARTZ, FLASH and SFLASH, which make use of systems of polynomial equations over small finite fields (not requiring large number packages), and thus necessitate a fundamental analysis.

In the future the template will still be useful to guide us through the second phase of performance evaluation (from October 2001).

3.3 Template Results

3.3.1 Block Ciphers

Block ciphers are symmetric encryption primitives that typically encipher blocks of 64 or 128 bits. For most block ciphers, the ciphertext is obtained by repeatedly applying a relatively simple cryptographic function to the input. The simple cryptographic function is called a *round function* and the key-derived material used in the round function is called a *round key*. The round keys are computed from the key using a *key-schedule* algorithm. An *SP-network* is type of block cipher which has the effect of modifying the entire data block in each round. A *Feistel* cipher is a type of block cipher which modifies only half of the block in each round.

In the NESSIE call for primitives specified the following security levels for block ciphers.

High: Key length of at least 256 bits. Block length at least 128 bits.

Normal: Key length of at least 128 bits. Block length at least 128 bits.

Normal-Legacy: Key length of at least 128 bits. Block length 64 bits.

The following lists the block ciphers submitted to NESSIE, the sizes of the blocks, and for which security levels they were submitted.

Name of cipher	Block size	Submitted in category		
		High	Normal	Normal-Legacy
CS-Cipher	64			X
Hierocrypt-L1	64			X
IDEA	64			X
Khazad	64			X
Misty1	64			X
Nimbus	64			X
Anubis	128	X		
Camellia	128	X	X	
Grand Cru	128		X	
Hierocrypt-3	128	X	X	
Noekeon	128		X	
Q	128	X	X	
SC2000	128	X	X	
SHACAL	160		X	
NUSH	64,128	X	X	X
RC6	128	X	X	
SAFER++	64,128	X	X	X

NOTES: NUSH was designed with three different block sizes: 64 bits, 128 bits, and 256 bits. RC6 has a variable block length $4w$ bits, where $w \geq 32$ is recommended by the designers. There are two variants of SAFER++, one with 64-bit blocks and one with 128-bit blocks.

We can separate block ciphers in five different categories:

1. 64-bit block ciphers: CS-Cipher, Hierocrypt-L1, IDEA, Khazad, Misty1, Nimbus, SAFER++, NUSH,
2. 128-bit block ciphers: Anubis, Camellia, NUSH, Grand Cru, Hierocrypt-3, Noekeon, Q, SC2000, SAFER++, NUSH, RC6,
3. 160-bit block ciphers: SHACAL,
4. 192-bit block ciphers: RC6,
5. 256-bit block ciphers: NUSH, RC6.

Based on our evaluation, we concluded that optimised block ciphers code was submitted for NUSH, Safer++ and CS-Cipher (excluding key schedule). Non-optimised code was submitted for Anubis, Grand Cru, Idea and Khazad.

3.3.1.1 CS-cipher

- Submitters: Pierre-Alain Fouque, CS Communication & Systèmes, France.
- Characteristics: 64-bit block; 128-bit key, 8-round SP-network.
- Description: Each round starts with a subkey xor, followed by a layer of four non-linear transformations and a byte permutation. This is repeated twice more in each round, but with constants used instead of the subkey in the initial xor. There is a final key xor after the last round.

The permutations are constructed in such a way that any byte in the output of one round depends on all eight input bytes to that round. The non-linear transformation takes two bytes of input, and has the following property: if one takes 256 inputs that are constant in one byte and take on all values once in the other byte, then each byte value occurs once in any of the two output bytes.

- Specifications of the Code:

Name of the candidate:	CS-Cipher
Block size (bits):	64
Word size (bits):	8
Key/Subkeys size (bits):	128 / 576
Code size (kB):	7

- Basic Operations:

Basic Operation Types	Number
Rotations (8-bit)	96
Table Look-ups (8 to 8-bit)	192
XOR (8-bit)	488
AND (8-bit)	96

- Comments: In the operation count we consider an 8-bit optimised version. This is equivalent to the code provided in the submission. This version requires one 8 to 8-bit S-box.

3.3.1.2 Hierocrypt-L1

- Submitters: Kenji Ohkuma, Fumihiko Sano, Hirofumi Muratani, Masahiko Motoyama, and Shinichi Kawamura (Toshiba Corporation, Japan)
- Characteristics: 64-bit block, 128-bit key, 6-round SP-network.
- Description: Each round consists of a layer with two parallel XS-boxes each operating on a 32-bit input, followed (except in the last round) by a linear diffusion layer with a 64-bit input. An XS-box consists of an upper subkey mixing layer, an upper S-box layer, a linear diffusion layer (with a 32-bit input), a lower subkey mixing layer, and a lower S-box layer. The subkeys are xored in, and the S-box used has 8-bit inputs and outputs. After the last round an output transformation introduces another subkey.
- Specifications of the Code:

Name of the candidate:	Hierocrypt-L1
Block size (bits):	64
Word size (bits):	8
Key/Subkeys size (bits):	128 / 896
Code size (kB):	12

- Basic Operations:

Basic Operation Types	Number
Shifts (32-bit)	72
Table Look-ups (8 to 8-bit)	8
Table Look-ups (8 to 32-bit)	48
Table Look-ups (8 to 64-bit)	40
XOR (32-bit)	60
XOR (64-bit)	36
AND (32-bit)	96

- Comments: In the operation count we consider a 32/64-bit optimised version. This version requires one 8 to 8-bit S-box, four 8 to 32-bit S-boxes, and eight 8 to 64-bit S-boxes. In this way the linear diffusion layers can be implemented efficiently with three 32-bit XORs and seven 64-bit XORs respectively. The code provided in the submission is not optimised at all.

3.3.1.3 IDEA

- Submitters: Richard Straub (MediaCrypt AG, Switzerland).
- Characteristics: 64-bit block, 128-bit key, 8.5-round semi-Feistel network. IDEA is already used in many applications like SSH and PGP.
- Description: The data is divided into four 16-bit words. At the beginning of each round, 4 16-bit subkeys are added or multiplied with each data word. The output of these operations is xored with two 16-bit words. These words enter the main core of the round function, where two additional subkey words are introduced, and additions and multiplications are performed. The output of this core is then xored with the transformed data words.
- Specifications of the Code:

Name of the candidate:	IDEA
Word size (bits):	16
Key/Subkeys size (bits):	128/832
Code size (kB):	22.6

- Basic Operations:

Basic Operation Types	Number
16-bit XOR	48
16-bit Additions	34
16-bit Multiplications	34

3.3.1.4 Khazad

- Submitters: Paulo Sérgio L. M. Barreto (Scopus Tecnologia S. A., Brazil) and Vincent Rijmen (K. U. Leuven, Belgium).
- Characteristics: 64 bit block, 128-bit key, 8-round SP-network.
- Description: Each round consists of a subkey addition, eight 8-bit to 8-bit S-boxes and a linear transformation (in the form of a matrix).
- Specifications of the Code:

Name of the candidate:	Khazad
Word size (bits):	64
Key/Subkeys size (bits):	128/576
Code size (kB):	58

- Basic Operations:

Basic Operation Types	Number
Shifts/Rotations (64-bit)	64
Table Look-ups (8-bit to 64-bit)	64
Additions, XOR, AND ...(64-bit)	64

- Comments: Eight different 8-bit to 64-bit S-boxes are used, except in the last round in which 8-bit to 64-bit S-boxes are used. The algorithm can also be implemented using an additional 8-bit to 8-bit S-box instead of the seven shifts.

3.3.1.5 Misty1

- Submitters: Eisaku Takeda (Mitsubishi Electric Corp., Japan).
- Characteristics: 64 bit block, 128-bit key. 8 rounds of a Feistel network with an additional key-dependent linear function FL before every second round.
- Description: In each round, the 32-bit input is divided into two 16-bit words: the left word is combined with a subkey, a 16-bit to 16-bit confusion function FI is applied, and the output is xored with the right word. The words are swapped and this process repeated twice more. After the function FI there is additional subkey xor.

The function FI has a similar structure to the round function with a 9-bit lefthand word and a 7-bit righthand word, and S-boxes replacing the FI function.

After every two rounds of the Feistel network and after the last round, the function FL is applied to both lefthand and righthand words. The input to the FL function is divided into two 16-bit words. The left word is combined with a subkey using and, and the outcome is xored with the right word. Then the right word and another subkey are combined using or before being xored with the left word to give the output to the function.

- Specifications of the Code:

Name of the candidate:	Misty1
Word size (bits):	32
Key/Subkeys size (bits):	128/512
Code size (kB):	6.9

- Basic Operations:

Basic Operation Types	Misty1
Table Look-ups (7-bit to 7-bit)	24
Table Look-ups (9-bit to 9-bit)	48
32-bit logical AND	10
16-bit logical AND	24
32-bit logical OR	10
32-bit logical XOR	56
16-bit logical XOR	72
8-bit logical XOR	24

- Comments: We consider 9-bit XOR as 16-bit XOR, and 7-bit XOR as 8-bit XOR.

3.3.1.6 Nimbus

- Submitters: Alexis Warner Machado (Gauss Informatica, Brazil).
- Characteristics: 64 bit block, 128-bit key. 5 rounds each affecting the whole data word (but not an SP-network).
- Description: Each round consists of a subkey xor, multiplication by another subkey, and then inversion of the data word.
- Specifications of the Code:

Name of the candidate:	Nimbus
Word size (bits):	64
Key/Subkeys size (bits):	128/704
Code size (kB):	

- Basic Operations:

Basic Operation Types	Variant 1	Variant 2	Variant 3
Shifts/Rotations (64-bit)	75	35	25
Table Look-ups (4-bit to 64-bit)	80	0	5
Table Look-ups (8-bit to 64-bit)	0	40	0
Table Look-ups (12-bit to 64-bit)	0	0	25
Multiplications (64-bit)	5	5	5
64-bit XOR	86	46	36
64-bit AND	80	40	30

- Comments: Variant 1 performs the inversions using 4-bit to 64-bit S-boxes. Smaller S-boxes (4-bit to 4-bit) can be used instead of 75 shifts. Variant 2 uses 8-bit to 64-bit S-boxes to perform the inversion. In Variant 2 using smaller S-boxes (8-bit to 8-bit) costs an additional 35 shifts. Variant 3 uses 12-bit to 64-bit S-boxes (and one small 4-bit to 64-bit S box) to do the inversion. Using one 12-bit to 12-bit S-box (and 4-bit to 4-bit S-box) costs an additional 25 shifts.

3.3.1.7 NUSH legacy version

- Submitters: Anatoly Lebedev (LAN Crypto, Int., Russia).
- Characteristics: 64-, 128- or 256-bit block, 128-, 192- or 256-bit key, 9, 68 or 132 rounds depending on block size.
- Description: Each round consists of four iterations. In each iteration two of four variables are updated using a subkey, while the other two are changed in a non-linear manner. There are two different kinds of iterations, one using or, the other using and. The cipher also has a pre-whitening step and a post-whitening step where a subkey is added.
- Specifications of the Code:

Name of the candidate:	NUSH legacy version
Block size (bits):	64
Word size (bits):	16
Key/Subkeys size (bits):	128, 192 or 256/1280
Code size (kB):	10

- Basic Operations:

Basic Operation Types	Number
Rotations (16-bit)	36
ADD / XOR (16-bit)	116
AND / OR (16-bit)	36

- Comments: In the operation count we consider a 16-bit optimised version. This is equivalent to the (optimised) code provided in the submission.

3.3.1.8 SAFER++ legacy version

- Submitters: Gurgun Khachatryan (Cylink Corporation, USA), James Massey (ETH Zürich, Switzerland), Melsik Kuregian (National Academy of Sciences, Armenia).
- Characteristics: Legacy version: 64-bit block; 128-bit key; 8-round SP-network. Normal version: 128-bit block; 128- or 256-bit key; 7- or 10-round SP-network, depending on key size.
- Description: Each round consists of a subkey mixing layer, an S-box layer, a second subkey mixing layer and two diffusion layers. Subkey bytes are added partly modulo 256, partly bit-by-bit modulo 2. Two different 8-bit to 8-bit S-boxes are used. The diffusion layers apply a shuffle and four 4-PHT transformations in parallel. After the last round an output transformation adds another subkey. For the legacy version of the cipher the text is expanded to 128 bits before entering the diffusion layers, and afterwards shortened to 64 bits again. The expansion is done by spreading the text to upper half bytes and setting the lower half bytes all zero, and the shortening by dropping the lower half bytes (which are zero) and merging to form bytes.
- Specifications of the Code:

Name of the candidate:	SAFER++ legacy version
Block size (bits):	64
Word size (bits):	8
Key/Subkeys size (bits):	128 / 1152
Code size (kB):	35 (including normal version)

- Basic Operations:

Basic Operation Types	Number
Table Look-ups (8 to 8-bit)	64
XOR (32-bit)	17
ADD (8-bit)	404

- Comments: In the operation count we consider an 8/32-bit optimised version. This version requires two 8 to 8-bit S-boxes. The code provided in the submission is not optimised.

3.3.1.9 Anubis

- Submitters: Paulo Sérgio L. M. Barreto (Scopus Tecnologia S. A., Brazil) and Vincent Rijmen (K. U. Leuven, Belgium).
- Characteristics: 128 bit block, variable key length (at least 128 bits). Variable number R of round SP-network (dependent on key size, minimum 12).
- Description: Each round consists of a subkey addition, 16 S-boxes (8-bit to 8-bit) and a linear transformation (presented as a matrix transpose operation). The S-boxes were chosen so as to guarantee that encryption and decryption are identical but with the order of the subkeys reversed.
- Specifications of the Code:

Name of the candidate:	Anubis
Word size (bits):	8
Key/Subkeys size (bits):	128-320/ 128·(R+1)
Code size (kB):	32

- Basic Operations:

For 12-round Anubis:

Basic Operation Types	Number
Shifts/Rotations (128-bit)	180
Table Look-ups (8-bit to 128-bit)	192
128-bit XOR	192
128-bit AND	192

- Comments: There are 16 different 8-bit to 128-bit tables. 8-bit to 128-bit table lookups can be replaced by two 8-bit to 64-bit table lookups. The cost of each additional round is 15 shifts, 16 table look-ups, 16 128-bit XORs and 16 128-bit AND.

3.3.1.10 Camellia

- Submitters: Shiho Moriai (Nippon Telegraph and Telecommunication Corporation, Japan) and Mitsuru Matsui (Mitsubishi Electric, Japan).
- Characteristics: 128-bit block, 128-bit, 192-bit or 256-bit key. 18 rounds of a basic Feistel structure with an additional invertible function FL (affecting both left and right data words, used after round 6 and round 12). For 192-bit and 256-bit keys, an additional application of the FL function and 6 more Feistel rounds are performed.
- Description: In each round the 64-bit input is divided into eight 8-bit strings, each xored with 8-bit subkey. The result then goes into 8 separate 8-bit to 8-bit S-boxes. After 6 rounds, the FL function is applied to both the left and right words. The 64-bit input to the FL function is divided into two 32-bit words. The left word is combined with a subkey word using and, and the outcome is rotated by one bit and then xored with the right word. The right word and another subkey word are then combined using or, and the output is xored with the left word.
- Specifications of the Code:

Name of the candidate:	128-bit keys	192-/256-bit keys
Word size (bits):	8	8
Key/Subkeys size (bits):	128/1664	192,256/2176
Code size (kB):	8.2	8.2

- Basic Operations:

Basic Operation Types	Number	Number
Shifts/Rotations (64-bit)	130	174
Table Look-ups (8-bit to 64-bit)	144	192
128-bit XOR	2	2
64-bit XOR	162	216
32-bit XOR	8	12
64-bit AND	144	192
32-bit AND	4	6
32-bit OR	4	6

3.3.1.11 Grand Cru

- Submitter: Johan Borst (K.U.Leuven, Belgium).
- Characteristics: 128-bit block, 128-bit key, 10-round SP-network.
- Description: Grand Cru can be viewed as an enhanced version of Rijndael [4]. Before the first round there is a key addition and an unkeyed diffusion layer. Each round starts with the xor of a round key, followed by the application of 16 parallel 8-bit S-boxes. The S-box is identical to that used in [4]. Next there is a keyed permutation of the bytes in the ciphertext block, and a multiplication of the block with a fixed 4×4 matrix over $GF(2^8)$. This matrix is also identical to that in [4]. The round ends with a key-dependent rotation of each byte in the ciphertext block. The final round of Grand Cru does not include the matrix multiplication. After the last round, the inverse of the unkeyed diffusion layer from the beginning is applied, followed by a key addition.
- Specifications of the Code:

Name of the candidate:	Grand Cru
Block size (bits):	128
Word size (bits):	8
Key/Subkeys size (bits):	128 / 4224
Code size (kB):	16

- Basic Operations:

Basic Operation Types	Number
Shifts (8-bit)	144
Rotations (8-bit)	160
Table Look-ups (5 to 8-bit)	50
Table Look-ups (8 to 8-bit)	672
XOR (8-bit)	1372
ADD (8-bit)	32
AND (8-bit)	144

- Comments: In the operation count we consider an 8-bit optimised version. Note that the 32-bit and 64-bit optimisations used in Rijndael are not possible for Grand Cru. This version requires one 8 to 8-bit S-box and one 5 to 8-bit S-box. The code provided in the submission is not optimised.

3.3.1.12 Hierocrypt-3

- Submitters: Kenji Ohkuma, Fumihiko Sano, Hirofumi Muratani, Masahiko Motoyama, Shinichi Kawamura (all Toshiba Corporation)
- Characteristics: 128-bit block, 128-bit, 192-bit or 256-bit key, 6, 7 or 8 round (depending on key size) SP-network.
- Description: Each round consists of a layer with four parallel XS-boxes each operating on a 32-bit input, followed (except in the last round) by a linear diffusion layer, operating on a 128-bit input. An XS-box consists of an upper subkey mixing layer, an upper S-box layer, a linear diffusion layer operating on a 32-bit input, a lower subkey mixing layer, and a lower S-box layer. The subkeys are xored in, and the S-box used has 8-bit inputs and outputs. After the last round there is another subkey addition.
- Specifications of the Code:

Name of the candidate:	Hierocrypt-3 128-bit key	192-bit key	256-bit key
Block size (bits):	128	128	128
Word size (bits):	8	8	8
Key/Subkeys size (bits):	128 / 1792	192 / 2048	256 / 2304
Code size (kB):	15 (all variants)		

- Basic Operations:

Basic Operation Types	Number	Number	Number
Shifts (32-bit)	144	168	192
Table Look-ups (8 to 8-bit)	16	16	16
Table Look-ups (8 to 32-bit)	96	112	128
Table Look-ups (8 to 128-bit)	80	96	112
XOR (32-bit)	120	140	160
XOR (128-bit)	76	91	106
AND (32-bit)	192	224	256

- Comments: In the operation count we consider a 32/128-bit optimised version. This version requires one 8 to 8-bit S-box, four 8 to 32-bit S-boxes, and sixteen 8 to 64-bit S-boxes. The linear diffusion layers can be implemented efficiently in this way with three 32-bit XORs and fifteen 128-bit XORs respectively. The code provided in the submission is not optimised at all.

3.3.1.13 Noekeon

- Submitters: Joan Daemen, Michael Peeters and Gilles Van Assche (Proton World, Belgium) and Vincent Rijmen (K.U.Leuven, Belgium).
- Characteristics: 128-bit block, 128-bit key, 16-round SP-network.
- Description: Each round consists of some subfunctions (each taking a 128-bit input), a linear function (taking the key and a 128-bit block as input), three rotations, 32 parallel 4-bit S-boxes and then another three rotations. To avoid round symmetries, in each round a round constant is added to 32 bits of the ciphertext. After the final round, there is an addition of a round constant and another application of the linear function. The encryption and decryption routines are very similar.
- Specifications of the code:

Name of the candidate:	Noekeon
Word size (bits):	32
Key/Subkeys size (bits):	128/2048
Code size (kB):	10.5

- Basic Operations:

Basic Operation Types	Number
Shifts/Rotations (32-bit)	128
128-bit XOR	16
32-bit XOR	304
32-bit AND	32
32-bit NOT	32
32-bit OR	32

3.3.1.14 NUSH

- Submitters: Anatoly Lebedev (LAN Crypto, Int.)
- Characteristics: 64-bit, 128-bit or 256-bit block, 128-bit, 192-bit or 256-bit key. 9, 68 or 132 rounds depending on block size.
- Description: Each round consists of four iterations. In each iteration two of four variables are updated using a subkey, while the other two are changed in a non-linear manner. There are two different kinds of iterations, one using or, the other using and. The cipher also has a pre-whitening step and a post-whitening step where a subkey is added.
- Specifications of the Code:

Name of the candidate:	NUSH
Block size (bits):	128
Word size (bits):	32
Key/Subkeys size (bits):	128, 192 or 256/4608
Code size (kB):	15

- Basic Operations:

Basic Operation Types	Number
Rotations (32-bit)	68
ADD / XOR (32-bit)	212
AND / OR (32-bit)	68

- Comments: In the operation count we consider a 32-bit optimised version. This is equivalent to the (optimised) code provided in the submission.

3.3.1.15 Q

- Submitters: Leslie McBride (Mack One Software, USA).
- Characteristics: 128-bit block, variable key length (up to 256-bit), 8.5 rounds.
- Description: The data is divided into sixteen 8-bit words (a 4×4 matrix of bytes). At the beginning of each round, a subkey word is xored in, then an 8-bit to 8-bit S-box is applied 16 times (for each of the 16 data bytes) and an additional subkey is xored in. A 4-bit to 4-bit S-box is then applied 32 times, the round key is xored in, a byte permutation is performed, and the 4-bit to 4-bit S-box is again used 32 times. After the last round, there is additional subkey xor, an 8-bit to 8-bit S-box layer, a subkey xor and an additional subkey xor (post-whitening). Two of the three subkeys used in each round are the same, and are equal for all rounds and also for the last half round.
- Specifications of the Code:

Name of the candidate:	bit-sliced	non-bitsliced
Word size (bits):	32	32
Key/Subkeys size (bits):	1280	1280
Code size (kB):	11.3	16.7

- Basic Operations:

Basic Operation Types	Number
Shifts/Rotations (32-bit)	24
Table Look-ups (8-bit to 8-bit)	128
128-bit XOR	26
32-bit XOR	144
32-bit AND	56
32-bit OR	16

- Comments: Optimisation takes into consideration only the bit-sliced version.

3.3.1.16 RC6

- Submitters: Jakob Jonsson (RSA Laboratories, Sweden) and Burt Kaliski (RSA Laboratories, USA).
- Characteristics: 128-bit block, 128-bit, 192-bit or 256-bit keys. 20-round generalised Feistel network.
- Description: RC6 was a candidate block cipher for the Advanced Encryption Standard (AES) and one of the five finalists. RC6 is based on the block cipher RC5. Like RC5, RC6 makes essential use of data-dependent rotations. In addition RC6 makes use of four working registers instead of two as in RC5, and uses an integer multiplication as an additional primitive operation. RC6 is a parameterised family of encryption algorithms, where RC6- $w/r/b$ is the version with word size w in bits, with r rounds and with an encryption key of b bytes. Thus, RC6 has $4w$ -bit blocks, where w can be any positive integer. However the designers recommend that $w \geq 32$ in the submission to NESSIE.
- Specifications of the Code:

Name of the candidate:	RC6
Block size (bits):	128 (default version)
Word size (bits):	32 (default version)
Key/Subkeys size (bits):	128 (default) / 1408
Code size (kB):	8

- Basic Operations:

Basic Operation Types	Number
Shifts (32-bit)	40
Rotations (32-bit)	80
XOR (32-bit)	40
ADD (32-bit)	84
MUL (32-bit)	40

- Comments: In the operation count we consider a 32-bit optimised version. The code provided in the submission is not optimised.

3.3.1.17 SAFER++

- Submitters: Gurgen Khachatrian (Cylink Corporation, USA), James Massey (ETH Zürich, Switzerland), Melsik Kuregian (National Academy of Sciences, Armenia)
- Characteristics: Legacy version: 64-bit block; 128-bit key; 8-round SP-network. Normal version: 128-bit block; 128- or 256-bit key; 7- or 10-round SP-network depending on key size.
- Description: Each round consists of a subkey mixing layer, an S-box layer, a second subkey mixing layer and two diffusion layers. Subkey bytes are added partly modulo 256, partly bit-by-bit modulo 2. Two different 8-bit to 8-bit S-boxes are used. The diffusion layers apply a shuffle and four 4-PHT transformations in parallel. After the last round an output transformation adds another subkey. For the legacy version of the cipher the text is expanded to 128 bits before entering the diffusion layers, and afterwards shortened to 64 bits again. The expansion is done by spreading the text to upper half bytes and setting the lower half bytes all zero, and the shortening by dropping the lower half bytes (which are zero) and merging to form bytes.
- Specifications of the Code:

Name of the candidate:	SAFER++ 128-bit key	256-bit key
Block size (bits):	128	128
Word size (bits):	8	8
Key/Subkeys size (bits):	128 / 1920	256 / 2688
Code size (kB):	35	35

- Basic Operations:

Basic Operation Types	Number	Number
Table Look-ups (8 to 8-bit)	112	160
XOR (64-bit)	15	21
ADD (8-bit)	456	648

- Comments: In the operation count we consider an 8/64-bit optimised version. This version requires two 8 to 8-bit S-boxes. The submission provides 8-bit optimised code and 32-bit optimised code (the latter only for encryption, not decryption).

3.3.1.18 SC2000

- Submitters: Takeshi Shimoyama, Hitoshi Yanami, Kazuhiro Yokoyama, Masahiko Takenaka, Kouichi Itoh, Jun Yajima, Naoya Torii, (all Fujitsu Laboratories LTD.) and Hidema Tanaka (Science University of Tokyo).
- Characteristics: 128-bit block, 128-bit, 192-bit or 256-bit key. 6.5- or 7.5-round mix of Feistel and SP-network.
- Description: The round function of SC2000 consists of a layer of 32 parallel 4-bit S-boxes followed by two rounds of a Feistel network. The round keys are xored with the cipher block before and after the application of the S-boxes. The last half round consists of key additions and S-box look-ups. The F -function in the Feistel network consists of a layer of four 6-bit S-boxes and eight 5-bit S-boxes, multiplication by a fixed 32×32 bit matrix and finally a linear output transformation. The key schedule is a complex transformation of the key selected by the user, in such a way that every 32-bit word of the round keys depends on the whole key.
- Specifications of the Code:

Name of the candidate:	SC2000 128-bit key	SC2000 192- or 256-bit key
Block size (bits):	128	128
Word size (bits):	32	32
Key/Subkeys size (bits):	128 / 1792	192 or 256 / 2048
Code size (in kilobyte=kB):	20	20

- Basic Operations:

Basic Operation Types	Number	Number
Shifts (32-bit)	48	56
Table Look-ups (10 to 10-bit)	24	28
Table Look-ups (11 to 11-bit)	48	56
XOR (32-bit)	87	98
XOR (64-bit)	12	14
XOR (128-bit)	14	16
AND / OR / NOT (32-bit)	145	168

- Comments: In the operation count we consider a 32/64/128-bit optimised version. This version requires one 10 to 10-bit S-box and two 11 to 11-bit S-boxes. Bitslicing is used instead of a 4-bit S-box. The code provided in the submission is not optimised at all.

3.3.1.19 SHACAL

- Submitters: Helena Handschuh, David Naccache (Gemplus, France).
- Characteristics: 160-bit block, 512-bit key (or shorter, but minimum 128 bits), 4 rounds of 20 steps.
- Description: The cipher is an encryption mode of the hash function SHA-1, where the key is inserted as the message and the plaintext as the initial value. It uses four rounds, each consisting of 20 steps. In a step, one of five variables is updated using a word of the expanded key and the other four variables in a non-linear manner (the different rounds use different non-linear functions). Each step there is also a rotation of one of these other variables. Note that all five variables get updated four times in each round.
- Specifications of the Code:

Name of the candidate:	SHACAL
Block size (bits):	160
Word size (bits):	32
Key/Subkeys size (bits):	512 / 2560
Code size (kB):	10

- Basic Operations:

Basic Operation Types	Number
Rotations (32-bit)	224
XOR (32-bit)	312
ADD (32-bit)	320
AND / OR / NOT (32-bit)	100

- In the operation count we consider a 32-bit optimised version. The code provided in the submission is not optimised.

3.3.1.20 Rijndael

This brief introduction to Rijndael comes from a Draft FIPS for the AES, available to <http://csrc.nist.gov/encryption/aes/index.html>. The Advanced Encryption Standard (AES) specifies a FIPS-approved cryptographic algorithm that can be used to protect electronic data. The AES algorithm is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits. Rijndael was chosen as standard in 2001, after the AES process. Rijndael is a substitution-linear transformation network with 10, 12 or 14 rounds, depending on the key size. A data block to be processed using Rijndael is partitioned into an array of bytes, and each of the cipher operations is byte-oriented. The Rijndael round function consists of four layers. In the first layer, an 8×8 S-box is applied to each byte. The second and third layers are linear mixing layers, in which the rows of the array are shifted, and the columns are mixed. In the fourth layer, subkey bytes are XORed into each byte of the array. In the last round, the column mixing is omitted. Rijndael was submitted as AES candidate by Joan Daemen (Proton World International) and Vincent Rijmen (Katholieke Universiteit Leuven). Rijndael was selected with the following comments: “Rijndael’s combination of security, performance, efficiency, implementability, and flexibility make it an appropriate selection for the AES for use in the technology of today and in the future.” We will thus compare the NESSIE candidates working on 128-bit block size with Rijndael.

- Basic Operations:

Basic Operation Types	128-bit
Shifts/Rotations (32-bit)	30
Table Look-ups (8-bit to 32-bit)	160
128-bit XOR	11
32-bit XOR	120

3.3.1.21 DES/triple-DES

The DES is a modified version of Lucifer submitted by IBM in 1975 after a FIPS call in 1973. DES became a standard in 1977 and for security reasons was modified into Triple DES. The migration from DES to Triple DES is discussed in FIPS 46-3.

- Number of different operations

The investigation of the DES-operations was split into four parts:

1. the straightforward implementation of the DES algorithm,
2. an optimised implementation for the DES algorithm,
3. the straightforward implementation of the triple-DES algorithm,
4. the same optimisation for the triple-DES algorithm.

The optimisation is as follows: The permutation P after the S-box operation is dropped and concatenated with the expansion permutation. This requires an application of P^{-1} after the initial permutation and P before the final permutation.

Key generation process

Type of operation	Number of operations
Table lookup (8-bit × 56-bit)	8
Table lookup (8-bit × 48-bit)	112
Left circular shift 28-bit	32
XOR 48-bit	96
XOR 56-bit	7

	DES	optimised DES	triple-DES	optimised triple-DES
Type of operation	Number of operations			
Table lookup (6-bit × 4-bit)	128	128	384	384
Table lookup (8-bit × 32-bit)	64	8	192	8
Table lookup (8-bit × 48-bit)	64	64	192	192
Table lookup (8-bit × 64-bit)	16	16	16	16
XOR 32-bit	48	6	144	6
XOR 48-bit	64	64	192	192
XOR 64-bit	14	14	14	14

- Number of different tables

To obtain smaller tables, some of the tables can be split, e.g. a 32-bit × 32-bit table can be split into four 8-bit × 32-bit tables, as is done in this document. Therefore it is not sufficient to just to count the tables in the original DES-implementation, but these split tables must be taken into account.

	DES	optimised DES	triple-DES	optimised triple-DES
Table size	Number of tables			
Table (6-bit × 4-bit)	8	8	8	8
Table (8-bit × 32-bit)	4	8	4	8
Table (8-bit × 48-bit)	11	11	11	11
Table (8-bit × 56-bit)	8	8	8	8
Table (8-bit × 64-bit)	16	16	16	16

The key generation process and the S-boxes are equal in each case. The table *PC1* is split into four 8-bit × 56-bit tables and the table *PC2* is split into seven 8-bit × 48-bit tables. The S-boxes are not split. The initial and the final permutation are also equal in each case and the associated tables are split into eight 8-bit × 64-bit tables (observe that there are two 64-bit × 64-bit tables to be split). The permutation *P* is split into four 8-bit × 32-bit tables, but in optimised algorithms the permutation P^{-1} is also used, so that there are eight 8-bit × 32-bit tables. The

expansion permutation E is split into four 8-bit \times 48-bit tables and in the optimised algorithms E is concatenated with P , which does not affect the number of 8-bit \times 48-bit tables.

3.3.1.22 Conclusion

See table 1.

3.3.2 Synchronous Stream Ciphers

The stream ciphers submitted to NESSIE were as follows:

- LILI-128
- LEVIATHAN
- SOBER-t16 and SOBER-t32
- SNOW
- BMGL

The submissions BMGL and LEVIATHAN cannot be compared directly with the others. Neither of these are based on linear feedback shift registers. BMGL is specified in terms of a block cipher, so could be thought of as being a mode of operation of a block cipher, and there is a proof reducing the security of BMGL to the security of the block cipher which it uses. LEVIATHAN meanwhile is designed to directly find any position in the key stream and uses a set of binary trees based on a key-dependent permutation to specify the key stream. The remaining submissions SOBER-t16, SOBER-t32 and SNOW are based on linear feedback shift registers. We further note that only the two SOBER stream ciphers have provided a rekeying schedule.

We note that the NESSIE call for primitives specified the following security levels for stream ciphers.

- *High*. Key length of at least 256 bits. Internal memory of at least 256 bits.
- *Normal*. Key length of at least 128 bits. Internal memory of at least 128 bits.

LILI-128 and SOBER-t16 claim to offer the ‘normal’ security level and SOBER-t32 claims to offer the ‘high’ security level, while and LEVIATHAN, BMGL and SNOW claim to offer either the ‘normal’ or ‘high’ security level depending on the key length selected.

Table 1: Block ciphers template results

Theoretical Analysis	Block size (bits)	Word size (bits)	Key size (bits)	Subkey size (bits)	Table lookups/Table size (bits×bits)	Shifts/ Rotations + multiplications	XOR,ADD (bit size)	AND,OR, NOT	Total table lookups	Total logical operations (8-bit)	Total multiplications (8-bit)	Code size (kB)	Optimised in submission?
Cs-cipher	64	8	128	576	192(8x8)	96(8bit)	488(8bit)	96(8bit)	192	584	0	7	yes
Hierocrypt-L1	64	8	128	896	8(8x8), 48(8x32), 40(8x64)	72(32bit)	60(32bit), 36(64bit)	96(32bit)	96	912	0	12	no
Idea	64	16	128	832		42+34 32-bit multiplications	388 (32bit)		0	1552	136	22.6	no
Khazad	64	8	128	576	64(8x64)	64	64(64bit)		64	512	0	58	no
Misty1	64	32	128	512	24(7x7), 48(9x9)	0	56(32bit), 72(16bit), 24(8bit)	10(32bit), 24(16bit), 10(32bit)	72	390	0	6.9	no
Nimbus6/1	64	64	128	704	80(4x64)	75+5 multiplications	86(64bit)	80(64bit)	80	1328	40		no
Nimbus6/2	64	64	128	704	40(8x64)	35+5 multiplications	46(64bit)	40(64bit)	40	688	40		no
Nimbus6/3	64	64	128	704	5(4x64), 25(12x64)	25+5 multiplications	36(64bit)	30(64bit)	30	528	40		no
Nush64 (legacy)	64	16	128,192, 256	1280		36(16bit)	116(16bit)	36(16bit)	0	304	0	10	yes
Safer++ (legacy)	64	8	128	1152	64(8x8)	128	17(32bit), 404(8bit)		64	472	0	35	no
DES	64	32	56	768	128/8(6x4), 8/8(8x32), 64/11(8x48), 16/16(8x64), 0/8(8x56)		6(32bit), 64(48bit), 14(64bit)		216	520	0		n. a.
Triple DES	64	32	56		384/8(6x4), 8/8(8x32), 192/11(8x48), 16/16(8x64), 0/8(8x56)		6(32bit), 192(48bit), 14(64bit)		600	1288	0		n. a.
Anubis (on R rounds)	128	8	128-320	128 * ($R+1$)	192+16*($R-12$) (8x128)	180+15*($R-12$)	192+16*($R-12$) (128bit)	192+16*($R-12$) (128bit)	192+16*($R-12$)	6144+512*($R-12$)	0	32	no
Camellia	128	8	128	1664	144(8x64)	130	2(128bit), 162(64bit), 8(32bit)	144(64bit), 4(32bit), 4(32bit)	144	2544	0	8.2	no
Camellia	128	8	192/256	2176	192(8x64)	174	2(128bit), 216(64bit), 12(32bit)	192(64bit), 6(32bit), 6(32bit)	192	3392	0	8.2	no
Grandru	128	8	128	4224	50(5x8), 672(8x8)	144(8bit), 160(8bit)	1372(8bit), 32(8bit)	144(8bit)	722	1548	0	16	no
Hierocrypt-3	128	8	128	1792	16(8x8), 96(8x32), 80(8x128)	144(32bit)	120(32bit), 76(128bit)	192(32bit)	182	2464	0	15	no
Hierocrypt-3	128	8	192	2048	16(8x8), 112(8x32), 96(8x128)	168(32bit)	140(32bit), 91(128bit)	224(32bit)	224	2912	0	15	no
Hierocrypt-3	128	8	256	2304	16(8x8), 128(8x32), 112(8x128)	192(32bit)	160(32bit), 106(128bit)	256(32bit)	256	3360	0	15	no
Noekeon	128	32	128	2048		128	16(128bit), 304(32bit)	32(32bit), 32(32bit), 32(32bit)	0	1856	0	10.5	no
Nush128	128	32	128,192, 256	4608		68(32bit)	212(32bit)	68(32bit)	0	1120	0	15	yes
Q (bit-sliced)	128	32		1280	128(8x8)	24	26(128bit), 144(32bit)	56(32bit), 16(32bit)	128	1280	0	11.3	no
RC6 (default)	128	32	128	1408		40(32bit), 80(32bit) +40(32bit) multiplications	40(32bit), 84(32bit)		0	496	160	8	no
Safer++/128	128	8	128	1920	112(8x8)		15(64bit), 456(8bit)		112	576	0	35	yes, for 8-bit and 32-bit
Safer++/256	128	8	256	2688	160(8x8)		21(64bit), 648(8bit)		160	816	0	35	yes, for 8-bit and 32-bit
SC2000 (128)	128	32	128	1792	24(10x10), 48(11x11)	48(32bit)	87(32bit), 12(64bit), 14(128bit)	145(32bit)	72	668	0	20	no
SC2000 (192-256)	128	32	192-256	2048	28(10x10), 56(11x11)	56(32bit)	98(32bit), 14(64bit), 16(128bit)	168(32bit)	84	760	0	20	no
Rijndael	128	8	128	1408	160(8x32)	30	11(128bit), 120(32bit)		160	656	0		n. a.
Shacal	160	32	512	2560		224(32bit)	272(32bit), 320(32bit)	180(32bit)	0	3088	0	8	no

3.3.2.1 LEVIATHAN

- Submitters: David McGrew (Cisco Systems, USA).
- Characteristics: Key size of 128 bits or 256 bits.
- Description: LEVIATHAN is a synchronous stream cipher. It is defined by a set of binary tree structures of height 16. Each node of each tree is associated with a triple of words (each of four bytes) $z|y|x$. The triple at the root of the j th tree is $1|0|j$. Key-dependent functions a and b map the triple s at a node to a triple at each of its two descendants, so that its lefthand descendant is $a(s)$ while its righthand descendant is $b(s)$.

We apply a function c to the triple at each leaf of each tree to give a word and use the words thus obtained as the key stream. The ‘key schedule’ for determining a permutation (which is used to define the key-dependent function) from the key is similar to that of the (alleged) RC4 stream cipher.

- Period: No claims made in the submission.

There is a distinguishing attack on LEVIATHAN [3, 10]. Even if there is no known attack faster than exhaustive key search that obtains the secret key, we decided not to analyse LEVIATHAN’s performance because of its security level lower than required by the NESSIE call.

3.3.2.2 LILI-128

- Submitters: E. Dawson, W. Millan, L. Penna and L. Simpson (all Queensland University of Technology, Australia) and J. Golić (University of Belgrade, Yugoslavia).
- Characteristics: Key size of 128 bits and internal memory of 128 bits.
- Description: LILI-128 is a synchronous stream cipher with a 128-bit key. It consists of two LFSRs each with primitive feedback polynomials. The first, $LFSR_c$, has length 39 while the second, $LFSR_d$, has length 89. Two bits from $LFSR_c$ (which is clocked regularly) are used to determine an integer c in the range $\{1, 2, 3, 4\}$ and $LFSR_d$ is clocked c times. A nonlinear filter f_d is then applied to ten bits of the output of $LFSR_d$ to produce the key stream.
- Period: The designers show that the cipher has period of approximately 2^{128} .

There are several attacks on LILI-128 as a result of the small size of the state space of the cipher. The best attack is that of Jönsson and Johansson [5] with an overall complexity of 2^{71} bit operations, requiring key stream of length 2^{30} bits and a precomputation of 2^{79} table lookups. In view of these results, it was decided not to analyse the performance of LILI.

3.3.2.3 SOBER-t16 and SOBER-t32

- Submitters: Philip Hawkes and Gregory Rose (Qualcomm International, Australia).
- Characteristics: Key size of 128 bits for SOBER-t16 and 256 bits for SOBER-t32, internal memory is 272 bits for SOBER-t16 and of 544 bits for SOBER-t32.
- Description: SOBER-t16 and SOBER-t32 are synchronous additive stream ciphers. The stream ciphers are constructed from a linear feedback shift register (*LFSR*), a non-linear filter (*NLF*) and a form of irregular decimation, called stuttering. SOBER-t16 outputs the key stream as 16-bit blocks and SOBER-t32 emits 32-bit blocks. The *LFSR* of the ciphers are of length 17 and operate over $GF(2^{16})$ for SOBER-t16 and $GF(2^{32})$ for SOBER-t32, respectively. The *NLF* is specified in the submission. The stuttering decimates the output of the *NLF* in an irregular fashion. For the stuttering it can be shown that there is an average of $\frac{6}{13}$ key stream output per clock of the *LFSR* for SOBER-t16 and $\frac{12}{25}$ for SOBER-t32.
- Period: By considering the period of the LFSR and the effect of the NLF and stuttering on this, it is likely that the average key stream period is $\frac{6}{13} \cdot (2^{272} - 1)$ for SOBER-t16 and $\frac{12}{25} \cdot (2^{544} - 1)$ for SOBER-t32.

Sober-t16 128 bit key:

- Keysetup:

Name of the candidate	Sober-t16 with 128 bit key
Table-lookups (16 bits)	371
Shift (16 bits)	70
Rotation (16 bits)	0
OR/AND/XOR (16 bits)	189
Add	212

- Key stream generation (12 bytes):

Name of the candidate	Sober-t16 with 128 bit key
Table-lookups (16 bits)	138
Shift (16 bits)	39
Rotation (16 bits)	0
OR/AND/XOR (16 bits)	156
Add	63

- Encrypt/Decrypt(1 byte):

Name of the candidate	Sober-t16 with 128 bit key
Table-lookups (8 bit word)	3
Shift (8 bits)	0
Rotation (8 bits)	0
OR/AND/XOR (8 bits)	1
Add	1

Sober-t32 128 bit key:

- Keysetup:

Name of the candidate	Sober-t32 with 128 bit key
Table-lookups (32 bit word)	371
Shift (32 bits)	78
Rotation (32 bits)	0
OR/AND/XOR (32 bits)	253
Add	242

- Key stream generation (48 bytes):

Name of the candidate	Sober-t32 with 128 bit key
Table-lookups (32 bit word)	288
Shift (32 bits)	111
Rotation (32 bits)	0
OR/AND/XOR (32 bits)	262
Add	119

- Encrypt/Decrypt (1 byte):

Name of the candidate	Sober-t32 with 128 bit key
Table-lookups (8 bit word)	3
Shift (8 bits)	0
Rotation (8 bits)	0
OR/AND/XOR (8 bits)	1
Add	1

3.3.2.4 SNOW

- Submitters: Patrick Ekdahl and Thomas Johansson (Lund University, Sweden).
- Characteristics: Key size of 128 or 256 bits, internal memory of 576 bits.
- Description: SNOW is a synchronous word-oriented stream cipher based on ideas from the classical summation generator [7].

The cipher consists of a length 16 linear feedback shift register (*LFSR*) over $GF(2^{32})$ feeding a Finite State Machine (*FSM*). The first symbol from the *LFSR* enters the *FSM* and the output of the *FSM* is bitwise added to the last entry of the *LFSR* producing 32 key stream bits. The *FSM* consist of two 32 bit registers, called *R1* and *R2*, and some operations to determine the output of the *FSM* and the new contents of the registers *R1* and *R2*.

- Period: The period of SNOW is likely to be $2^{16 \cdot 32} - 1$.
- Key setup:

Name of the candidate	Snow 128 bit	Snow 256 bits
Table-lookups (32 bits)	256	256
Shift (32 bits)	268	280
Rotation (32 bits)	64	64
OR/AND/XOR (32bits)	736	768
Add	158	174

- Key stream generation:

Name of the candidate	Snow 128 bit	Snow 256 bits
Table-lookups (32 bits)	64	64
Shift (32 bits)	112	112
Rotation (32 bits)	16	16
OR/AND/XOR (32bits)	224	224
Add	77	77

- Encrypt/Decrypt (1 byte):

Name of the candidate	Snow 128 bit	Snow 256 bits
Table-lookups (8 bits)	3	3
Shift (8 bits)	0	0
Rotation (8 bits)	0	0
OR/AND/XOR (8 bits)	1	1
Add	1	1

3.3.2.5 BMGL

- Submitters: Johan Håstad (Royal Institute of Technology, Stockholm, Sweden) and Mats Näslund (Ericsson Research, Sweden).
- Characteristics: Key sizes are as for Rijndael (128, 192 and 256 bits).
- Description: The submission BMGL is a pseudo-random number generator with the block cipher Rijndael as cryptographic core. Rijndael is considered as a one-way function from the key to the ciphertext. BMGL iterates this function and extracts a few pseudo-random bits in each iteration using hard-core predicates. The construction uses an optimization of earlier work on pseudo-random generators [1, 6].

The template analysis was carried out on a version of BMGL based on Rijndael with block size 128 bit, inducing algorithm parameters $m = 16$, $n = 128$.¹

- Keysetup:

Name of the candidate	BMGL
Table-lookups (8 bit word)	560
Shift (8 bits)	0
Rotation (8 bits)	0
OR/AND/XOR (8 bits)	0
Add	544

- Key stream generation (2 bytes): operations for 1 Rijndael key setup (with blocksize 128 and 128 key bits) and operations for 1 Rijndael block encryption (with blocksize 128 and 128 key bits)

Name of the candidate	BMGL
Table-lookups (32 bit word)	160
Shift (32 bits)	80
Rotation (32 bits)	0
OR/AND/XOR (32 bits)	224
Add	20

- Encrypt/decrypt (1 byte):

Name of the candidate	BMGL
Table-lookups (8 bit word)	3
Shift (8 bits)	0
Rotation (8 bits)	0
OR/AND/XOR (8 bits)	1
Add	1

¹For implementation, we will consider the parameters suggested by the submitters, namely $m = 40$, $n = 256$.

3.3.2.6 Arcfour

- Description: Arcfour is the alleged RC4 as available from the OpenSSL package.
- Number of different operations:
 - Initialisation Process:

Type of operation	Number of operations
Table lookup ($\lceil \log_2(\text{keylength_in_byte}) \rceil$ -bit \times 8-bit)	256
Table lookup (8-bit \times 8-bit)	512
Table store (8-bit \times 8-bit)	512
Addition 8-bit mod256	512

- Encryption for 1 byte
For variable keylength an additional operation $\text{mod}(\text{keylength})$ on the key index is necessary. The input size for the table is then $\lceil \log_2(\text{max_keylength_in_byte}) \rceil$ bits.

Type of operation	Number of operations
Table lookup (8-bit \times 8-bit)	3
Table store (8-bit \times 8-bit)	2
Addition 8-bit mod256	3

The Exclusive-Or-Operation with the plaintext is NOT counted.

- Number of different tables: two different tables are needed, namely the state array S with 8-bit input and output and the key array with $\lceil \log_2(\text{keylength_in_byte}) \rceil$ -bit input and 8-bit output. Observe that the size of the latter array depends on the choice of the key size.

3.3.2.7 Conclusion

See table 2 and table 3.

Table 2: Stream ciphers template results: key setup

Key Setup	Table Lookups	Shifts	Rotations	And, Or, Xor	Add
BMGL (8bit word)	560	0	0	0	544
Snow128 (32bit word)	256	268	64	736	158
Snow256 (32bit word)	256	280	64	768	174
Sober-t16/128 (16bit word)	371	70	0	189	212
Sober-t32/128 (32bit word)	371	78	0	253	242
Arcfour	$256(\log_2(\text{byte-keylength}) \times 8\text{bit}),$ 512(8x8bit), 512 table storages (8x8bit)	0	0	0	512

Table 3: Stream ciphers template results: key stream generation

Key stream Generation	Table lookups	Shifts	Rotations	And, Or, Xor	Add	Bits output/LSFR cycle
BMGL (32bit word)	160	80	0	224	20	32
Snow128 (32bit word)	64	112	16	224	77	32
Snow256 (32bit word)	64	112	16	224	77	32
Sober-t16/128 (16bit word)	138	39	0	156	63	16
Sober-t32/128 (32bit word)	288	111	0	262	119	32
Arcfour	3(8x8bit), 2(8x8bit) table storages	0	0	0	3	8^1

¹This number measures bits/stream cipher cycle since there is no LFSR for Arcfour.

3.3.3 Message Authentication Codes

There are two candidates in this category, Two-Track-MAC and UMAC.

3.3.3.1 Two-Track-MAC

- Submitters: Bart van Rompay (K. U. Leuven, Belgium) and Bert den Boer (deBIS).
- Characteristics: 160-bit message authentication code with 160-bit key. Message divided into 512-bit blocks (last one padded if necessary).
- Description: Two-Track-MAC is based on the unkeyed hash function RIPEMD-160. First, the message to be authenticated is padded with a 1-bit, and then 0-bits until its length is $448 \bmod 512$. Then the binary representation of the length of the original message ($\bmod 2^{64}$) is appended, so the length of the message becomes a multiple of 512 bits. Each 512-bit block is split into a set of sixteen 32-bit words, M_1, \dots, M_{16} . The secret key is a set of five 32-bit words, K_1, \dots, K_5 . The algorithm works by iterating a round function as follows. Two sets of five 32-bit words X_1, \dots, X_5 and Y_1, \dots, Y_5 and a set of 16 message words are input to two different functions f_L and f_R that output five 32-bit words each.

$$A_1, \dots, A_5 = f_L(X_1, \dots, X_5, M_1, \dots, M_{16})$$

$$B_1, \dots, B_5 = f_R(Y_1, \dots, Y_5, M_1, \dots, M_{16})$$

These two functions are identical to the ones used in RIPEMD-160. Then $C_i = A_i - X_i$ and $D_i = B_i - Y_i \bmod 2^{32}$ are computed, for $1 \leq i \leq 5$. To finish the round, the ten words C_i and D_i are mixed with two linear transformations g_L and g_R .

$$E_1, \dots, E_5 = g_L(C_1, \dots, C_5, D_1, \dots, D_5)$$

$$F_1, \dots, F_5 = g_R(C_1, \dots, C_5, D_1, \dots, D_5)$$

E_i and F_i together with the next set of message words form the inputs to the next round. In the first round, the five secret keywords are input as both X_i and Y_i , $1 \leq i \leq 5$. In the final round, f_L and f_R swap places. At the end of the final round the subtractions $E_i = C_i - D_i \bmod 2^{32}$ are carried out instead of g_L and g_R . The results from these subtractions form the output of Two-Track-MAC.

- Specifications of the Code:

Name of the candidate:	Two-Track-MAC
Block size (bits):	512
Word size (bits):	32
Key/Subkeys size (bits):	160
Code size (kB):	13

- Basic Operations:

Basic Operation Types	Number
Rotations (32-bit)	320
XOR (32-bit)	256
ADD / SUB (32-bit)	613
AND / OR / NOT (32-bit)	192

- In the operation count we consider a 32-bit optimised version. The code provided in the submission is not optimised.

3.3.3.2 UMAC

- Submitters: Ted Krovetz (Intel Corporation, USA), John Black, Shai Halevi, Hugo Krawczyk and Phillip Rogaway.
- Characteristics: Message authentication code supporting key lengths of 128 and 256 bits, and output lengths of 32 through 256 bits (by multiples of 32).
- Description: UMAC is based on a hash scheme, called UHash, which first compresses the message to be authenticated to a fixed length. The resulting value is then xored with a nonce which has been enciphered with the AES (Rijndael) block cipher. UHash consists of three different layers, each based on a universal hash function family: the first layer uses the fast NH hash family to compress the message by a fixed ratio; the second layer uses the RP hash family, which is not as fast as NH but generates an output of fixed length; the third layer uses the IP hash family, which reduces the length of its input to a more appropriate size, and also strengthens the universality properties. The PRG which computes (from the user key) the key material needed in the internal operation of UHash is based on the AES in output-feedback mode. Many options are available in an implementation of UMAC but two named parameter sets have been specified: UMAC16 and UMAC32.
- Specifications of the Code:

Name of the candidate:	UMAC16	UMAC32
Block size(bits):	8192	8192
Word size (bits):	16	32
Key/Subkeys size (bits):	128 or 256 / 8192	128 or 256 / 8192
Code size (kB):	146 (both variants)	

- Basic Operations:

Basic Operation Types	Number	Number
ADD (16-bit)	2048	
ADD (32-bit)	1024	512
ADD (64-bit)		256
MUL (32-bit)	1024	
MUL (64-bit)		256

- Comments: In the operation count we only consider the first level of hashing where the different message blocks are compressed by a fixed ratio. This ratio is 256 for UMAC16 and 128 for UMAC32. This is repeated four times for UMAC16 and two times for UMAC32 to obtain (after the next two levels of hashing and a final xor with a Rijndael-encrypted nonce) 64-bit MAC results with forgery probability about 2^{-60} for both versions. It is difficult to describe the next two levels of hashing here because they do not work on separated message blocks. The code provided in the submission is optimised.

3.3.4 Collision Resistant and one-way Hash Functions

3.3.4.1 Whirlpool

- Submitters: Paulo Sérgio L. M. Barreto (Scopus Tecnologia S. A., Brazil) and Vincent Rijmen (K. U. Leuven, Belgium).
- Characteristics: 512-bit hash function, operating on 512-bit blocks (if necessary the last block is appended by 1 and as many 0's as required).
- Description: Whirlpool is an iterated 512-bit hash function with a 512-bit block size. The 512 bits are regarded as a matrix of 8×8 bytes. The basic function W runs in 10 rounds using a 512-bit key. Each round is composed of a non-linear function, an 8-bit to 8-bit S-box executed 64 times in parallel, then a permutation π shifting the j th column by j cells down cyclically, followed by a linear diffusion layer (a multiplication by a 8×8 byte MDS matrix). A key is introduced at this point, and the round is iterated. The above structure is used in a Miyaguchi-Preneel hashing scheme. Therefore $IV = 0 \dots 0$ is defined, and $H_t = W[H_{t-1}](m_i) \oplus H_{t-1} \oplus m_i$, where the last H_t is the hash value.
- Specifications of the Code:

Name of the candidate:	Whirlpool
Word size (bits):	8
Key/Subkeys size (bits):	512, 256/5120
Code size (kB):	65

- Basic Operations:

Basic Operation Types	Number
Shifts/Rotations	1188
8-bit to 64-bit Table Look-Ups	1280
64-bit XOR	1175
64-bit AND	1268

- Comments: The results are based on the submitted code which seems optimal for 64-bit processors. Note that the word size is actually 512 bits, thus, a 64-bit implementation is the best way of implementing this cipher (a 128-bit implementation is not optimal).

In the following we give the descriptions of standards hash functions chosen by the National Institute of Standards and Technology (NIST). This brief introduction of American standard hash functions is extracted from FIPS 180-2, announcing the Secure Hash Standard, which supersedes FIPS 180-1 by adding three algorithms that are capable of producing larger message digests (see: <http://csrc.nist.gov/publications/>). The SHA-1 algorithm specified herein is the same algorithm that was specified previously in FIPS 180-1, although some of the notation has been modified to be consistent with the notation used in the SHA-256, SHA-384, and SHA-512 algorithms.

The result of SHA-function is an output called a message digest. The message digests range in length from 160 to 512 bits, depending on the algorithm. Secure hash algorithms are typically used with other cryptographic algorithms, such as digital signature algorithms and keyed-hash message authentication codes, or in the generation of random numbers (bits).

All four of the algorithms are iterative, one-way hash functions that can process a message to produce a condensed representation called a message digest. These algorithms enable the determination of a message integrity: any change to the message will, with a very high probability, result in a different message digest. This property is useful in the generation and verification of digital signatures and message authentication codes, and in the generation of random numbers (bits). Each algorithm can be described in two stages: preprocessing and hash computation. The four algorithms differ most significantly in the number of bits of security that are provided for the data being hashed; this is directly related to the message digest length.

Algorithm	Message size (bits)	Block size (bits)	Word size (bits)	Output size (bits)	Security (bits)
SHA-1	$< 2^{64}$	512	32	160	80
SHA-256	$< 2^{64}$	512	32	256	128
SHA-384	$< 2^{128}$	1024	64	384	192
SHA-512	$< 2^{128}$	1024	64	512	256

3.3.4.2 SHA-1

SHA-1 may be used to hash a message having a length of l bits, where $0 \leq l < 2^{64}$.

- Specifications of the Code:

Name of the candidate:	SHA-1
Block size (bits):	512
Word size (bits):	32
Code size (kB):	

- Basic Operations:

Basic Operation Types	Number
Rotations (32-bit)	224
XOR (32-bit)	312
ADD (32-bit)	325
AND / OR / NOT (32-bit)	100

- Comments: In the operation count we consider a 32-bit optimised version.

3.3.4.3 SHA-256

SHA-256 may be used to hash a message, M , having a length of l bits, where $0 \leq l < 2^{64}$.

- Specifications of the Code:

Name of the candidate:	SHA-256
Block size (bits):	512
Word size (bits):	32
Memory requirement (kB):	
Code size (kB):	

- Basic Operations:

Basic Operation Types	Number
Shifts (32-bit)	96
Rotations (32-bit)	576
XOR (32-bit)	640
ADD (32-bit)	600
AND / NOT (32-bit)	384

- Comments: In the operation count we consider a 32-bit optimised version.

3.3.4.4 SHA-512 and SHA-384

SHA-512 and SHA-384 may be used to hash a message having a length of l bits, where $0 \leq l < 2^{128}$. In the following, SHA-512 is described with SHA-384. This is because the SHA-384 algorithm is identical to SHA-512, with the exception of using a different initial hash value and truncating the final hash value to 384 bits.

- Specifications of the Code:

Name of the candidate:	SHA-512 or SHA-384
Block size (bits):	1024
Word size (bits):	64
Code size (kB):	

- Basic Operations:

Basic Operation Types	Number
Shifts (64-bit)	128
Rotations (64-bit)	736
XOR (64-bit)	816
ADD (64-bit)	760
AND / NOT (64-bit)	480

- Comments: in the operation count we consider a 64-bit optimised version.

3.3.4.5 Conclusion

See table 4.

Table 4: Message authentication schemes and hash functions template results

Theoretical Analysis	Block size (bits)	Word size (bits)	Key size (bits)	Subkey size (bits)	Table lookups/ Table size (bits×bits)	Shifts/ Rotations + multi- plications	XOR, ADD, MULT (bit size)	AND, OR, NOT	Total table lookups	Total logical operations (8-bit)	Total multi- plications (8-bit)	Code size (kB)	Optimized in submission?
Umac16	8192	16	128, 256	8192			2048(16bit), 1024(32bit), 1024(32bit mult.)		0	8192	4096	146	no
Umac32	8192	32	128, 256	8192			512(32bit), 256(64bit), 256(64bit mult.)		0	4096	2048	146	no
Two-Track-Mac	512	32	160	160		320	128(32bit), 613(32bit)	384 (32bit)	0	4500	0	13	no
Whirlpool	512	8	512	5120	1280(8x64)	1188	1175(64bit)	1268 (64bit)	1280	19544	0	65	no
SHA-1	512	32				0/224 (32bit)	312 (32bit), 325 (32bit)	100 (32bit)	0	2948	0		
SHA-256	512	32				96 (32bit)/ 576 (32bit)	640 (32bit), 600 (32bit)	384 (32bit)	0	6496	0		
SHA-384/-512	1024	64				128 (64bit)/ 736 (64bit)	816 (64bit), 760 (64bit)	480 (64bit)	0	16448	0		

In the following section, we consider the theoretical performance of the asymmetric submissions to NESSIE. We do not take into account the concatenation operations and the conversions of integers into byte strings or bit strings as these operations are important for the implementation but not for performance and we can just consider the other operations such as modular exponentiation, modular multiplication and solving polynomial equations. Note that in the second phase of the project we shall consider larger sizes of the candidates' parameters.

3.3.5 Asymmetric Encryption Schemes

3.3.5.1 ACE Encrypt

- Submitters: Thomas Schweinberger and Victor Shoup (IBM Zurich).
- Description: A probabilistic encryption scheme based on the Diffie-Hellman problem over a finite field of prime order. It is a variant of the Cramer-Shoup encryption scheme.
- Key generation

The scheme uses a random prime P of length k , at least 128 bytes and a prime q of length 32 bytes. A random integer $g_1 \in \{2, \dots, P-1\}$ of order q and five random integers w, x, y, z_1, z_2 in Z/qZ are generated. Then the following integers are computed:

$$\begin{aligned} - g_2 &= g_1^w \pmod{P} \\ - c &= g_1^x \pmod{P} \\ - d &= g_1^y \pmod{P} \\ - h_1 &= g_1^{z_1} \pmod{P} \\ - h_2 &= g_1^{z_2} \pmod{P} \end{aligned}$$

Then the key generation performs 5 modular exponentiations modulo a k bytes prime, with $k \geq 128$.

- Parameters: Here is a summary of the parameter lengths:
 - maximal message length: at most 2^{64} bytes
 - the public key consists of $(P, q, g_1, g_2, c, d, h_1, h_2, k_1, k_2)$ where k_1 and k_2 are two random byte strings of length 124 and 296 bytes respectively, when P has length 128 bytes. Then the public key length is 1348 bytes when P is a 128 bytes integer.
 - The secret key consists of (w, x, y, z_1, z_2) , its length is 160 bytes.
 - size of the ciphertext: the ciphertext is given by (s, u_1, u_2, v, e) , where s is a random 16 bytes string, u_1, u_2 and v lie in $(Z/PZ)^*$ and e is the cryptogram whose length depends on the size of the message. In fact, we have $|e| = |M| + 16 \cdot \lceil |M|/1024 \rceil$. Therefore the length of the ciphertext is 400 bytes plus the length of the cryptogram e . Note that this length is not fixed, but depends on the length of the message.
- Encryption
 - six modular exponentiations modulo P . Note that some computations can be improved since the product of two powers has to be made.

- one modular multiplication modulo P : this multiplication is the one that can be mixed with the two exponentiations.
 - one UOWHF (Universal One Way Hash Function). It is based on the SHA-1 hash function. This function is fully described in the submission. We just note here that when this function is called, with $|P| = 128$ in bytes, then 8 calls to the SHA-1 function are made, and 4 called to the core SHA-1 function.
 - one ESHF (Entropy Smoothing Hash Function). This function is based on two computations of the simplified SHA-1 function.
 - a symmetric encryption scheme, composed with a MAC. It uses many mask values generated with the MARS block cipher used in counter mode.
- Decryption
 - five modular exponentiations modulo P .
 - one modular multiplication modulo q .
 - one UOWHF (Universal One Way Hash Function).
 - one ESHF (Entropy Smoothing Hash Function).
 - a symmetric decryption scheme, composed with a MAC. It uses a lot of mask values generated with the MARS block cipher used in counter mode.

3.3.5.2 ECIES

- Submitters: Don B. Johnson (Certicom Corp., USA) and Simon Blake-Wilson (Certicom Corp., Canada).
- Description: A probabilistic encryption scheme based on the elliptic curve Diffie-Hellman problem.

We shall suppose that both parties have agreed to the use of a specific elliptic curve, with given cryptographic parameters— n , a large prime dividing $\#E(\mathbb{F}_q)$, G a point on the curve of order n — that is the setup of the scheme will not be taken into consideration, since it can be done once and for all.

Input: a message M (byte string) to be encrypted. The notations are detailed in Appendix A.

Output: a byte string C consisting of a key \overline{R} , the ciphertext EM , and a tag D .

- Key generation: (generations of 2 random integers less than $n + 2$ elliptic scalar multiplications)
 - Public and secret keys of receiver (d, Q) : receiver chooses a random $d \in [1, n - 1]$ and computes $Q = dG$.
 - Public and secret keys of sender (k, R) : receiver chooses a random $k \in [1, n - 1]$ and computes $R = kG$.
- Encryption:
 - Convert R into a byte string \overline{R} using EO.
 - Key sharing: elliptic Diffie-Hellman (1 elliptic scalar multiplication) to produce $z \in \mathbb{F}_q$.
 - Convert z into a byte string using FO.
 - Use the specified key-derivation function (concatenation of the same hash function applied to the seed concatenated with a counter) to produce the encryption key EK and the MAC key MK.
 - Use the specified symmetric encryption scheme with key EK (triple-DES, XOR) on the message.
 - Use the specified MAC scheme on the encrypted message EM with key MK to get a tag D .
 - Output $\overline{R}||EM||D$.
- Decryption:
 - Recognize the sender's output form and produce R from \overline{R} using OE.
 - Retrieve key z using Diffie-Hellman. Everything then proceeds as before, except that symmetric encryption is replaced by symmetric decryption.

3.3.5.3 EPOC-1

- Submitters: Tatsuaki Okamoto (NTT, Japan).
- Description:

EPOC-1 is mainly designed for key distribution. It uses two random primes p and q of the same length k , at least 48 bytes. The modulus n is equal to p^2q and hence of length $3k$, at least 144 bytes. The scheme needs also a random element $g \in (Z/nZ)^*$ such that the order of $g_p = g^{p-1} \bmod p^2$ is p , h is a n -th power modulo n . The message length $mLen$ and the length $rLen$ of the random used in the scheme are such that $mLen + rLen \leq pLen - 1$. A hash function is used, with an identifier denoted by HID . We consider HID and all the lengths $mLen$, $rLen$, $pLen$ and $hLen$ (length of hash function output) to be system parameters; they are thus not included in the public key.

The security of this scheme is based on the p -subgroup assumption (cf. section 5.2.2 of [8]).

- Key Generation:
 - two primes p and q are generated, of length k , then the modulus $n = p^2q$ is calculated
 - a random element $g \in (Z/nZ)^*$ is generated, such that the order of $g_p = g^{p-1} \bmod p^2$ is p
 - a random element h_0 in $(Z/nZ)^*$ is generated, and $h = h_0^n \bmod n$ is computed
- Parameters: we recall that $k = pLen$ is the length of the primes p and q . In general, we have $k = 48$ bytes.
 - maximal message length: the plaintext has length $mLen$ such that $mLen + rLen \leq k - 1$. Furthermore, the input of the hash function is $mLen + rLen$.
 - public key size: the public key consists of (n, g, h) . Its length is $9k$ bytes, that is 432 bytes for $k = 48$ bytes.
 - secret key size: The secret key consists of (p, g_p) . Its length in bytes is $3k$, that is 144 bytes.
 - system parameters: they consist of

$$(HID, pLen, hLen, rLen, mLen)$$

The length is 24 bytes if we take 20 bytes for the hash function identifier.

- ciphertext size: $3k$ bytes, 144 bytes for $k = 48$.
- Encryption

- two modular exponentiations are performed modulo n . The exponents are at most $kLen$ bytes for the first one and $hLen$ bytes for the second one.
 - one modular multiplication modulo n .
 - one hash computation is made with an input of length at most k bytes.
- Decryption
 - three modular exponentiations, one modulo p^2 with an exponent of length k and two exponentiations performed modulo n with exponents of length at most k bytes.
 - one modular multiplication modulo n
 - two modular subtractions modulo p
 - one modular inversion and one modular multiplication modulo p
 - one hash function is computed with an input of length at most k bytes.

3.3.5.4 EPOC-2

- Submitters: Tatsuaki Okamoto (NTT, Japan).
- Description: EPOC-2 is a hybrid scheme using a symmetric encryption which can be an additive stream cipher for short messages. The scheme uses two random primes p and q of the same length $k = pLen$, at least 48 bytes. The modulus n is equal to p^2q and hence of length at least 144 bytes. The scheme needs also a random element $g \in (Z/nZ)^*$ such that the order of $g_p = g^{p-1} \pmod{p^2}$ is p , h is a n -th power modulo n . The length $rLen$ of the random used in the scheme is such that $rLen \leq pLen - 1$. Two hash functions are used, with an identifier denoted by HID and GID . We consider here that HID , GID , and all the lengths $mLen$, $rLen$, $pLen$, $gLen$ and $hLen$ (length of hash functions outputs) are system parameters and are not then included in the public key. A pair of symmetric encryption and decryption scheme is used, with an identifier denoted $SEID$. For an additive stream cipher, this identifier equals 1.

The security of this scheme is based on the factoring assumption (cf. section 5.2.2 of [8]).

- Key Generation: the operations are the following:
 - two primes p and q are generated, of length k , then the modulus $n = p^2q$ is calculated
 - a random element $g \in (Z/nZ)^*$ is generated, such that the order of $g_p = g^{p-1} \pmod{p^2}$ is p
 - a random element h_0 in $(Z/nZ)^*$ is generated, and $h = h_0^n \pmod{n}$ is computed
- Parameters: we recall that $k = pLen$ is the length of the primes p and q . In general, we have $k = 48$ bytes.
 - maximal message length: the plaintext has length $mLen$ such that the input of the hash function is $mLen + rLen$.
 - public key size: the public key consists of (n, g, h) . Its length is $9k$ bytes, that is 432 bytes for $k = 48$ bytes.
 - secret key size: The secret key consists of (p, g_p) . Its length in bytes is $3k$, that is 144 bytes.
 - system parameters: they consist of

$$(HID, GID, SEID, pLen, hLen, rLen, mLen)$$

The length is 64 bytes if we take 20 bytes for each hash function identifier and for the symmetric scheme identifier.

- ciphertext size is $3k$ plus the length of the output of the symmetric encryption scheme, $mLen$ in case of an additive stream cipher.

- Encryption

- two modular exponentiations performed modulo n with exponents of length $hLen$ and $rLen$.
- one modular multiplication modulo n .
- two hash computations are made with inputs of length at most k bytes for H and $rLen$ bytes for G .
- one symmetric encryption is made, with key of length $gLen$ and with the message of length $mLen$. When $mLen$ equals $gLen$, an additive stream cipher can be used.

- Decryption

- three modular exponentiations, one modulo p^2 with exponent of length k bytes, and two modulo n with exponents of length $hlen$ and k .
- one modular multiplication modulo n
- two modular subtractions modulo p
- one modular inversion and one modular multiplication modulo p
- two hash functions are computed with inputs of length at most k bytes for H and $rlen$ for G .
- one symmetric decryption function is made, with key of length $gLen$.

3.3.5.5 EPOC-3

- Submitters: Tatsuaki Okamoto (NTT, Japan).
- Description: EPOC-3 is a hybrid scheme using a symmetric encryption which can be an additive stream cipher for short messages for example. The scheme uses two random primes p and q of the same length k , at least 48 bytes. The modulus n is equal to p^2q and hence of length at least 144 bytes. The scheme needs also a random element $g \in (Z/nZ)^*$ such that the order of $g_p = g^{p-1} \pmod{p^2}$ is p , h is a n -th power modulo n . The length of the random used in the scheme is such that $RLen \leq pLen - 1$. Two hash functions are used, with an identifier denoted by HID and GID . We consider here that HID , GID , and all the lengths $mLen$, $rLen$, $pLen$, $gLen$ and $hLen$ (length of hash functions outputs) are system parameters and are not then included in the public key. A pair of symmetric encryption and decryption scheme is used, with an identifier denoted $SEID$. For an additive stream cipher, this identifier equals 1.

The security of this scheme is based on the gap-factoring assumption (cf. section 5.2.2 of [8]).

- Key Generation: the operations are the following:
 - two primes p and q are generated, of length k , then the modulus $n = p^2q$ is calculated
 - a random element $g \in (Z/nZ)^*$ is generated, such that the order of $g_p = g^{p-1} \pmod{p^2}$ is p
 - a random element h_0 in $(Z/nZ)^*$ is generated, and $h = h_0^n \pmod{n}$ is computed
- Parameters: we recall that $k = pLen$ is the length of the primes p and q . In general, we have $k = 48$ bytes.
 - maximal message length: the plaintext has length $mLen$ such that the input of the hash function is $3k + kLen + RLen + mLen$.
 - public key size: the public key consists of (n, g, h) . Its length is $9k$ bytes, that is 432 bytes for $k = 48$ bytes.
 - secret key size: The secret key consists of (p, g_p) . Its length in bytes is $3k$, that is 144 bytes.
 - system parameters: they consist of

$$(HID, GID, SEID, pLen, hLen, rLen, mLen, RLen)$$

The length is 65 bytes if we take 20 bytes for each hash function identifier and for the symmetric scheme identifier.

- ciphertext size is $3k + hLen$ plus the length of the output of the symmetric encryption scheme, denoted $kLen$, which equals $mLen$ in case of an additive stream cipher.

- Encryption

- two modular exponentiations performed modulo n with exponents of length $RLen$ and $rLen$.
- one modular multiplication modulo n .
- two hash computations are made with inputs of length at most $3k + kLen + RLen + mLen$ bytes for H and $RLen$ bytes for G .
- one symmetric encryption is made, with key of length $gLen$ and with the message of length $mLen$. When $mLen$ equals $gLen$, an additive stream cipher can be used.

- Decryption

- one modular exponentiation, one modulo p^2 with exponent of length k bytes
- two modular subtractions modulo p
- one modular inversion and one modular multiplication modulo p
- two hash functions are computed with inputs of length at most $3k + kLen + RLen + mLen$ bytes for H and $Rlen$ for G .
- one symmetric decryption function is made, with key of length $gLen$.

3.3.5.6 PSEC-1

- Submitters: Tatsuaki Okamoto (NTT, Japan).
- Description: PSEC-1 is mainly designed for key distribution. This scheme uses some system parameters composed with the elliptic curve domain parameters (k, F_q, a, b, p, P) : q is defined to be at least a 20-byte prime number in the implementation. a and b are two elliptic curve coefficients, elements of F_q . They define an elliptic curve E . p is a prime at least 20 bytes long, and divides the number of points on the curve E . P is a point on the curve of order p , P is represented by its affine coordinates in F_q^2 . The schemes needs also a hash function with identifier HID and the length of some parameters $mLen$, length of the message, $pLen$, length of the prime p , $hLen$, length of the hash function output, $rLen$ length of the random number during the encryption and $qLen$ the length of the prime q . Note that $mLen$ and $rLen$ are such that $mLen + rLen \leq qLen$. All these parameters are considered to be system parameters. The security of this scheme is based on the elliptic curve decision Diffie-Hellman assumption (cf. section 5.2.2 of [8]).
- Key generation: the key generation accepts as input the elliptic curve parameters (k, F_q, a, b, p, P) . Then a random element $s \in (Z/pZ)^*$ is chosen and the point $W = sP$ is computed.
- Parameters: Let k be the size in bytes of the prime number p , dividing the number of points on the curve. Typically $k = 20$ bytes.
 - maximal message length: the plaintext has length $mLen$ such that $mLen + rLen \leq qLen$. Furthermore, the input of the hash function is $mLen + rLen$.
 - system parameters: they consist of

$$(k, F_q, a, b, p, P, HID, mLen, hLen, rLen, qLen)$$

The length can be considered as 366 bytes for the parameters used in the scheme.

- public key size: the public key is given by W , a point on the curve. Its length is $2k$ bytes, that is 40 bytes for the particular case with $k = 20$ bytes.
 - secret key size: the secret key is just given by s , a random integer in F_q , used to calculate the point W . Its length is k bytes, that is 20 bytes in the example.
 - the ciphertext is given by (C_1, c_2) where C_1 is a point on the curve, that is an element of F_q^2 , and c_2 has length $qLen$, 20 bytes in general.
- Encryption
 - one hash function computation with input length at most $qLen$ bytes. The output is $hLen$ bytes.

- 2 computations of points on the curve. These are of the form *constant · point* where the constants has length *hLen*.
- 1 XOR between two values of length *qLen*.
- Decryption
 - 2 computations of points on the curve, with constants of length *hLen* and *pLen*.
 - 1 XOR between two values of length *qlen*
 - one hash computation, with input of length at most *qLen* and output of length *hLen*

3.3.5.7 PSEC-2

- Submitters: Tatsuaki Okamoto (NTT, Japan).
- Description: PSEC-2 is an hybrid scheme using a symmetric encryption which can be an additive stream cipher for short message for example. This scheme uses some system parameters composed with the elliptic curve domain parameters (k, F_q, a, b, p, P) : q is defined to be at least a 20 bytes prime number in the implementation. a and b are two elliptic curve coefficients, elements of F_q . They define an elliptic curve E . p is a prime at least 20 bytes long, and divides the number of points on the curve E . P is a point on the curve of order p , P is represented by its affine coordinates in F_q^2 . The schemes needs also two hash functions with identifier HID and GID , and a pair $(SymEnc, SymDec)$ of symmetric key encryption and decryption algorithms, with an identifier $SEID$. The output of the encryption scheme is denoted $kLen$. The scheme uses the length of some parameters $mLen$, length of the message, $pLen$, length of the prime p , $hLen$ and $gLen$, lengths of the hash functions outputs, $rLen$ length of the random number during the encryption and $qLen$ the length of the prime q . Note that $rLen$ is such that $rLen \leq qlen$. All these parameters are considered as system parameters.

The security of this scheme is based on the elliptic curve Diffie-Hellman assumption (cf. section 5.2.2 of [8]).

- Key generation: the key generation accepts as input the elliptic curve parameters (k, F_q, a, b, p, P) . Then a random element $s \in (Z/pZ)^*$ is chosen and the point $W = sP$ is computed.
- Parameters: Let k be the size in bytes of the prime number p , dividing the number of points on the curve. Typically $k = 20$ bytes.

- maximal message length: the plaintext has length $mLen$ such that the input of the hash function is $mLen + rLen$.
- system parameters: they consist of

$$(k, F_q, a, b, p, P, HID, GID, SEID, hLen, gLen, rLen, qLen)$$

of length 387 bytes.

- public key size: the public key is given by W , a point on the curve. Its length is $2k$ bytes, that is 40 bytes for the particular case with $k = 20$ bytes.
- secret key size: the secret key is just given by s , a random integer in F_q , used to calculate the point W . Its length is k bytes, that is 20 bytes in the example.
- the ciphertext is given by (C_1, c_2, c_3) where C_1 is a point on the curve, that is an element of F_q^2 , c_2 has length $qLen$, 20 bytes in general and c_3 is the output of the symmetric encryption scheme, $kLen$ bytes.

- Encryption

- 2 hash functions computations with input length at most $qLen$ bytes for the H and $rLen + mLen$ for G . The output of these functions is respectively $hLen$ and $gLen$ bytes.
- 2 computations of points on the curve. These are of the form $constant \cdot point$ where the constants has length $hLen$.
- 1 XOR between two values of length $qLen$.
- one symmetric encryption computation, with key length $gLen$ and output $kLen$. Note that if $gLen = mLen$ the symmetric scheme can be an additive stream cipher.

- Decryption

- 2 computations of points on the curve, with constants of length $hLen$ and $pLen$.
- 1 XOR between two values of length $qlen$
- two hash computations, with input of length respectively at most $qLen$ and $rLen + mLen$. The output of H is $hLen$ bytes, and $gLen$ bytes for G .
- one symmetric decryption computation, with key length $gLen$.

3.3.5.8 PSEC-3

- Submitters: Tatsuaki Okamoto (NTT, Japan).
- Description: PSEC-3 is an hybrid scheme using a symmetric encryption which can be an additive stream cipher for short message for example. This scheme uses some system parameters composed with the elliptic curve domain parameters (k, F_q, a, b, p, P) : q is defined to be at least a 20 bytes prime number in the implementation. a and b are two elliptic curve coefficients, elements of F_q . They define an elliptic curve E . p is a prime at least 20 bytes long, and divides the number of points on the curve E . P is a point on the curve of order p , P is represented by its affine coordinates in F_q^2 . The schemes needs also two hash functions with identifier HID and GID , and a pair $(SymEnc, SymDec)$ of symmetric key encryption and decryption algorithms, with an identifier $SEID$. The output of the encryption scheme is denoted $kLen$. The scheme uses the length of some parameters $mLen$, length of the message, $pLen$, length of the prime p , $hLen$ and $gLen$, lengths of the hash functions outputs, $rLen$ length of the random number during the encryption and $qLen$ the length of the prime q . Note that $rLen$ is such that $rLen \leq qLen$. All these parameters are considered as system parameters.

The security of this scheme is based on the elliptic curve gap Diffie-Hellman assumption (cf. section 5.2.2 of [8]).

- Key generation: the key generation accepts as input the elliptic curve parameters (k, F_q, a, b, p, P) . Then a random element $s \in (Z/pZ)^*$ is chosen and the point $W = sP$ is computed.
- Parameters: Let k be the size in bytes of the prime number p , dividing the number of points on the curve. Typically $k = 20$ bytes.

- maximal message length: the plaintext has length $mLen$ such that the input of the hash function is $8 + 4qLen + 2mLen$.
- system parameters: they consist of

$$k, F_q, a, b, p, P, HID, GID, SEID, hLen, gLen, qLen,$$

of length 387 bytes.

- public key size: the public key is given by W , a point on the curve. Its length is $2k$ bytes, that is 40 bytes for the particular case with $k = 20$ bytes.
- secret key size: the secret key is just given by s , a random integer in F_q , used to calculate the point W . Its length is k bytes, that is 20 bytes in the example.
- the ciphertext is given by (C_1, c_2, c_3, c_4) where C_1 is a point on the curve, that is an element of F_q^2 , c_2 has length $qLen$, 20 bytes in general, c_3 is the output of the symmetric encryption scheme, $kLen$ bytes and c_4 is $hLen$ bytes long.

- Encryption

- 2 hash functions computations with input length at most $qLen$ bytes for the G and $8 + mLen + 4qLen + kLen$ for H . The output of these functions is respectively $gLen$ and $hLen$ bytes.
- 2 computations of points on the curve. These are of the form $constant \cdot point$ where the constants has length $pLen$.
- 1 XOR between two values of length $qLen$.
- one symmetric encryption computation, with key length $gLen$ and output $kLen$. Note that if $gLen = mLen$ the symmetric scheme can be an additive stream cipher.

- Decryption

- 1 computation of point on the curve, with constant of length $pLen$.
- 1 XOR between two values of length $qlen$
- two hash computations, with input of length respectively at most $qLen$ for G and $8 + mLen + 4qLen + kLen$ for H . The output of H is $hLen$ bytes, and $gLen$ bytes for G .
- one symmetric decryption computation, with key length $gLen$.

3.3.5.9 RSA-OAEP

- Submitters: Jakob Jonsson and Burt Kaliski (RSA Laboratories Europe and RSA Security Inc.).
- Description: The scheme uses one hash function, which output is denoted $hLen$. It uses a Mask Generation Function too, based on the hash function, and fully described in the submission. It accepts as input a seed and an integer, corresponding to the expected output.

The security of this scheme is based on the RSA assumption (cf. section 5.2.2 of [8]).

- Key generation: as input, this algorithm takes L the desired bit length for the modulus and the public exponent e which is predetermined. To generate a random prime p of length $L/2$ bits, the algorithm generate a random odd integer in the interval, makes some trial divisions by the first 2000 primes, verifies that $\gcd(p - 1, e) = 1$ and performs some Miller-Rabin tests, such that the probability that a composite number passes the test is less than 2^{100} . When two primes p and q are generated, $n = pq$ is computed and the parameters necessary for the secret key are computed, depending on the variant used (inversion of e modulo $\varphi(n)$ or three inversion modulo $L/2$ bits integers).
- Parameters: Let k be the size in byte of the RSA modulus, typically 128 bytes. The scheme uses some encoding parameters P , which can be empty. We denoted their length by $PLen$, which is at most the input limitation for the hash function.
 - maximal message length: $k - 2 - 2hLen$ bytes, where $hLen$ means the length of the output of the hash function (generally 20 bytes). In general, the message cannot exceed 106 bytes.
 - the public key consists of (n, e) , where n is the modulus of length k bytes, and e is the public exponent, which can be small. Hence the public key size is $k + 3$ bytes, 131 bytes in the general case.
 - Secret key size
 - * variant 1: for this variant, the secret key consists of (n, d) , where n is the modulus and d the secret exponent. The size of the secret key is then $2k$ bytes, that is 256 bytes in the general case.
 - * variant 2: for this variant, the secret key consists of the quintuple $(p, q, dP, dQ, qInv)$, where p and q are the factors of the modulus, dP is the inverse of e modulo $p - 1$, dQ is the inverse of e modulo $q - 1$, and $qInv$ is the inverse of q modulo p . This variant is the most used in practice. Its length is $5k/2$ bytes, 320 bytes in general.
 - The ciphertext is an element of $(Z/nZ)^*$, therefore its size is l bytes, that is 128 bytes in general.

- Encryption

- one modular exponentiation. The exponent is the public one, which can be small in practice; we can consider that its length is at most 3 bytes. The modulus is k bytes long.
- one hash computation is made with an input of length $PLen$ bytes.
- 2 MGF computations are made: for the first computation, the seed is 20 bytes long, and the output is $k - hLen - 1$ bytes long. For the second computation, the seed is $k - hLen - 1$ bytes long and the output is $hLen$ bytes long.
- two XORs are made, between values of length $k - hLen - 1$ bytes for the first one and $hLen$ bytes for the second one.

- Decryption

- Variant 1: one modular exponentiation with an exponent and a modulus of size k bytes.
- Variant 2:
 - * two modular exponentiations, with exponent and modulus with $k/2$ byte length.
 - * one modular multiplication modulo an integer of length $k/2$ bytes, between two integers of the same length, $k/2$ bytes.
 - * one multiplication/division by an integer.
 - * one hash function.
 - * two XORs.
 - * mask generation functions based on hash
- 2 MGF are computed: one with seed of size $k - hLen - 1$ and output $hLen$ bytes, the other with seed of size $hLen$ and output of size $k - hLen - 1$ bytes.
- 2 XOR between values of length $k - hLen - 1$ and $hLen$ bytes.
- 1 hash computation with input of length $PLen$ bytes.

3.3.5.10 Summary

We present two tables, tables 5 and 6, containing the practical parameter length and the ciphertext length for each scheme. For the first one, the parameters are chosen such that the security of all the schemes are considered equivalent. For the schemes based on the RSA problem, the modulus is a 128-byte integer, for schemes based on the Diffie-Hellman problem, the finite cyclic group is Z/PZ^* for P of length 128 bytes, for those based on the elliptic curve Diffie-Hellman problem, the field size and the size of the order of the base point are 20 bytes.

Table 5: Usual parameters lengths (in bytes)

Scheme	Public Key Length	Secret key Length
ACE-Encrypt	1348	160
ECIES	40	20
EPOC-1	240	144
EPOC-2	240	144
EPOC-3	240	144
PSEC-1	40	20
PSEC-2	40	20
PSEC-3	40	20
RSA-OAEP	129	320

Table 6: Ciphertext Length (in bytes) (message length = 16 bytes)

ACE-Encrypt	432
ECIES	47, 57, 67, 77
EPOC-1	144
EPOC-2	160
EPOC-3	180
PSEC-1	60
PSEC-2	76
PSEC-3	96
RSA-OAEP	128

The second table contains the ciphertext size depending on the message length (here 16 bytes). We consider that the output of the hash functions is 20 bytes (160 bits), the output

of the symmetric encryption scheme used in the EPOC and PSEC schemes is considered as 16 bytes (128 bits), the length of the random string (salt) is 16 bytes.

Remark. The ciphertext length for EPOC1 and PSEC1 may be surprising. However, the condition imposed on the maximal message length $mLen$ is very constraining. Then, to encrypt a large message, we have to partition it into $mLen$ bytes messages and encrypt each separately (ECB mode). The CBC mode should also be used, providing ciphertext of the same length. However, these schemes are impractical for long messages. That's why these schemes are mainly designed for key encapsulation.

3.3.6 Asymmetric Digital Signature Schemes

3.3.6.1 ACE Sign

- Submitters: Thomas Schweinberger and Victor Shoup (IBM Zurich).
- Description: This scheme makes use of a size parameter m such that $1024 \leq m \leq 16384$ to generate the keys. This parameter is the length in bits of the modulus. Two safe prime numbers p and q are generated, of the same size $m/2$. We have $p = 2p' + 1$ and $q = 2q' + 1$ where p' and q' are prime numbers. Then $N = pq$ is computed. The scheme needs also a random 20 bytes prime number, a random element $h \in QR_N$ of maximal order $p'q'$, and x , the $a - th$ power of h modulo N . Two random byte strings k' and s are generated, of length 184 bytes and 32 bytes respectively. The scheme needs also two UOWHF (Universal One Way Hash Function) fully described in the submission. These hash functions are based on the SHA-1 hash function and the core SHA-1 state transformation. In the most general case, the size parameter m , length of the modulus, is fixed to 1024, i.e. a modulus length of 128 bytes.

The security of this scheme is based on the strong RSA assumption (cf. section 5.2.2 of [8]).

- Key generation: to generate safe primes, the authors proposed an algorithm described in [2]. This algorithm is claimed to yield easily a factor of 10 speed-up over the naive method. Two safe primes have to be generated in the key generation. The product $N = pq$ is then computed. A random 20 bytes prime e' is generated. Then a random element $h \in QR_N$ of maximal order $p'q'$ is generated, using h' a random element in $1, \dots, N - 1$ such that $\gcd(h', N) = 1$ and $\gcd(h' \pm 1, N) = 1$. Then h is defined as $h'^2 \pmod N$.
- Parameters
 - Maximal message length: at most 2^{64} bytes, the input limitation of the UOWHF used in the scheme.
 - The public key consists of (N, h, x, e', k', s) . Its length is then $3m/8 + 236$ bytes, that is 620 bytes in the general case (m is the length in bits of the modulus, $m/8$ the length in bytes).
 - The private key is given by (p, q, a) . Its length is $2m/8 = m/4$ bytes, that is 256 bytes.
 - The signature is given by (d, w, y, y', \tilde{k}) , where (d, w) is a certificate of correctness for a prime number e , its length is 85 bytes. y and y' are two elements in Z/NZ and \tilde{k} is a random byte string of length $20l + 64$ bytes, where l is the length in bits of $\lceil (|M| + 8)/64 \rceil$. Then the length of the signature is $2m/8 + 149 + 20l = m/4 + 149 + 20l$ bytes. For a 20 bytes message and a 128 bytes modulus, the signature is a 425 bytes string.

- There are no system parameters since all the hash functions are fixed by the system and there is no flexibility with them.
- Signature generation
 - one UOWHF computation, with key of length $20l + 64$, where l is the length in bits of $\lceil (|M| + 8)/64 \rceil$. The message to hash has length at most 2^{64} bytes.
 - one modular square is made modulo N , for an element in Z/NZ .
 - two modular exponentiations and a modular multiplication are performed modulo N . The two exponents have length 20 bytes. This step can be improved and can be then faster than two independent exponentiations.
 - a random 20 bytes prime e is generated, with its certificate (w, d) of correctness. The algorithm to generate this prime is fully described in the submission. Briefly, it uses MARS in sum counter mode to generate random elements P and R of length 52 bytes and 107 bytes respectively, with P prime (verified by trial divisions and Miller-Rabin tests). To test the primality of $e = 2PR + 1$, some trial divisions are made, then some Miller-Rabin tests are performed. Finally some congruences on P and R have to be satisfied. In average, 384 MARS have to be performed to generate the prime e .
 - a UOWHF is performed, using the second function. However, it is equivalent to use the first one with key length 184 bytes and a message size $85 + 20l$ bytes, where l is the length in bits of $\lceil (|M| + 8)/64 \rceil$.
 - one inversion modulo $p'q'$ is computed, for a 20 bytes element, a modular subtraction and a modular multiplication modulo $p'q'$ are performed.
 - an exponentiation modulo N is made.
- Verification of the Signature
 - one prime certificate verification is made. This needs 4 MARS computations, some Miller-Rabin tests and some congruences verifications.
 - one UOWHF is performed with a key of size $20l + 64$ and a message of at most 2^{64} bytes.
 - two modular exponentiations and a modular multiplication modulo N are performed. This step can be improved using the standard algorithmic techniques.
 - one UOWHF is performed, using a 184 bytes key and a $85 + 20l$ bytes message.
 - two exponentiations and a multiplication modulo N are made, subject to standard optimisations.

3.3.6.2 ECDSA

- Submitters: Don B. Johnson (Certicom Corp., USA) and Simon Blake-Wilson (Certicom Corp., Canada).
- Description: A digital signature scheme based on the elliptic curve discrete logarithm problem.

We shall suppose that both parties have agreed to the use of a specific elliptic curve, with given cryptographic parameters— n , a large prime dividing $\#E(\mathbb{F}_q)$, G a point on the curve of order n — that is the setup of the scheme will not be taken into consideration, since it can be done once and for all. In addition to the above general assumption, one should also specify a hash function to use when generating signatures. Also, the sender (signer) should have generated an elliptic curve key pair (d_U, Q_U) and the receiver (verifier) should possess Q_U . The notations are detailed in appendix.

Input: a message M (byte string) to be signed.

Output: a pair of integers (r, s) as signature on M .

- Key Generation: selection of (d_U, Q_U) and of an ephemeral elliptic curve key pair (k, R) .
- Signature
 - Convert the x -coordinate x_R of R into an integer $\overline{x_R}$ using FI and reduce modulo n to get r .
 - Let H be the hashed message. Using OB one retrieves, after BO, a byte string E (a truncation of H) and applying BI and integer e .
 - Compute s . This requires two multiplications, one addition and one inversion modulo n .
 - Output (r, s) .
- Verification
 - Find e from H , the hashed message, as above.
 - Compute u_1 and u_2 . This requires one inversion and two multiplications modulo n .
 - Retrieve R . This requires two scalar multiplications and one elliptic curve point addition.
 - Convert the x -coordinate x_R of R into an integer $\overline{x_R}$ using FI and reduce modulo n to get v .
 - Compare v to r and check that they are equal.

3.3.6.3 ESIGN

- Submitters: Tatsuaki Okamoto (NTT, Japan).
- Description: A digital signature scheme based on the approximate e -th root problem in a group of order $N = p^2q$. Thus, very roughly, a signature for a message M is deemed to be valid if, given the hash $H(m)$ of a message m , $s^e \pmod n$ lies in the interval $[H(m), H(m) + pq]$.

The scheme needs a hash function with an identifier HID of length fixed here to 20 bytes.

- Key generation: The key generation algorithm generates two random prime p and q of the same size denoted k , typically 48 bytes. The modulus N equals p^2q and has length $3k$. An integer $e \leq 8$ is selected.
- Parameters
 - the system parameters consist of (HID, k) of length 22 bytes (we consider here that the primes p and q are less than 2^{16})
 - the maximal message length is limited to the input limitation of the hash function, 2^{61} in case of SHA-1.
 - The public key consists of (n, e) . Its length is $3k + |e|$ bytes, 145 bytes in the general case, with $e = 8$.
 - the private key consists of (p, q) , of length $2k$.
 - The signature consists of s of length $3k$ bytes, 144 bytes in the general case.
- Generation of the signature
 - one modular exponentiation modulo N , with the exponent e given in the public key.
 - one modular subtraction modulo N
 - one hash computation with the message to sign as input.
 - one division of an integer of length $3k$ by the product pq of length $2k$ (the product pq is precalculated since it is used many times for each signature generation).
 - one multiplication and one subtraction with integers of length $3k$.
 - one modular exponentiation with exponent $e - 1$ modulo the prime p , one modular multiplication modulo p with e and the previous result, one inversion modulo p of the obtained integer, and finally one multiplication of this result of length k with an other integer of the same length.
 - one modular multiplication modulo N with an integer of length k and the product pq , and one addition of this result with an integer of length $3k$.

- Verification of the signature
 - one modular exponentiation with the public exponent e
 - one hash function computation with the message as input

3.3.6.4 RSA-PSS

- Submitters: Jakob Jonsson and Burt Kaliski (RSA Laboratories Europe and RSA Security Inc.).
- Description: This scheme uses two random primes p and q of roughly the same length $k/2$, typically 128 bytes. The modulus N equals pq . The scheme uses one hash function, which output is denoted $hLen$. It also uses a Mask Generation Function, based on the hash function, and fully described in the submission. It accepts as input a seed and an integer, corresponding to the attended output. $sLen$ is the salt length and is considered as a system parameter.

The security of this scheme is based on the RSA assumption (cf. section 5.2.2 of [8]).

- Key generation: as input this algorithm takes L the desired bit length for the modulus and the public exponent e which is predeterminate. To generate a random prime p of length $L/2$ bits, the algorithm generate at random an odd integer in the interval, makes some trial divisions by the first 2000 primes, verify that $\gcd(p-1, e) = 1$ and performs some Miller-Rabin tests, such that the probability that a composite number passes the test is less than 2^{100} . When two primes p and q are generated, $n = pq$ is computed and the parameters necessary for the private key are computed, depending on the variant used (inversion of e modulo $\varphi(n)$ or three inversion modulo $L/2$ bits integers).
- Parameters
 - maximal message length: input limitation of the hash function, 2^{61} bytes for SHA-1.
 - system parameters: the identifier of the hash function, the identifier of the Mask Generation Function and the salt length $sLen$. The length of these parameters is 60 bytes.
 - The public key consists of (n, e) , where n is the modulus and e the public exponent, which can be small. Its length is $k + |e|$ bytes, 129 bytes in the general case.
 - the private key can have two different form.
 - * variant 1: for this variant, the private key consists of (n, d) , where n is the modulus and d the private exponent. The size of the private key is then $2k$ bytes, that is 256 bytes in the general case.
 - * variant 2: for this variant, the private key consists of $(p, q, dP, dQ, qInv)$, where p and q are the factors of the modulus, dP is the inverse of e modulo $p-1$, dQ is the inverse of e modulo $q-1$, and $qInv$ is the inverse of q modulo p . This variant is the most used in practice. Its length is $5k/2$ bytes, 320 bytes in general.

- Signature size: the signature consists of s which is an element of Z/NZ . Its length is l bytes, i.e. 128 bytes in general.
- Signature Generation
 - Variant 1: one modular exponentiation with the private exponent d of length k .
 - Variant 2:
 - * two modular exponentiations with exponent of size $k/2$ and modulo p and q .
 - * one subtraction modulo p
 - * one multiplication modulo p
 - * one multiplication with two integers of length $k/2$
 - * one addition with integers of length $k/2$ and k .
 - two hash computations with the message as input for the first one and a $8 + hLen + sLen$ byte string for the second.
 - one MGF computation with a seed of size $hLen$ and an output of size $\lceil k/8 \rceil - hLen - 1$.
 - one XOR with integers of length $\lceil k/8 \rceil - hLen - 1$.
- Verification of the signature: the verification is the same for the two variants
 - one modular exponentiation with e as exponent
 - one hash function computation with the message as input
 - one MGF computation with seed of length $hLen$ and output length $\lceil k/8 \rceil - hLen - 1$
 - one XOR with integers of length $\lceil k/8 \rceil - hLen - 1$
 - one hash with input of length $8 + hLen + sLen$
- Comments: the signature needs 384 modular multiplications with CRT and 1536 without CRT.

3.3.6.5 FLASH

- Submitters: Jacques Patarin, Nicolas Courtois and Louis Goubin (BULL CP8).
- Description: Flash is a deterministic signature scheme belonging to the family of multivariate public key schemes. More precisely, Flash is a C^{*-} scheme.
- Key generation: to generate the private key, composed with two affine bijection s and t from F_{256}^{37} to F_{256}^{37} and with a 80-bit secret string, two methods can be used: for both, each affine bijection is described with a square invertible matrix 37×37 and a column matrix 37×1 with coefficients in F_{256} . With the first method, each coefficient is generated at random, and a test is made to verify that the square matrix is invertible. The second method is based on the LU decomposition. A lower triangular matrix and an upper triangular matrix 37×37 are generated (each coefficient is generated at random in F_{256}), then the product is computed. For both methods, each coefficient of the column matrix is generated at random. The public key is deduced from the private one. It is composed with a function G from $(F_{256})^{37}$ to F_{256}^{26} defined by the following relation:

$$G(X) = [t(\varphi^{-1}(F(\varphi(s(X)))))]_{0 \rightarrow 207},$$

where the notation $[\lambda]_{r \rightarrow s}$ is the string of bits of λ defined by $(\lambda_r, \dots, \lambda_s)$. The function F is defined by:

$$\forall A \in F_{256}^{37}, F(A) = A^{256^{11}+1}.$$

It is easy to see that G can be represented by 26 quadratic polynomials of a special form, with coefficients in F_{256} .

- Parameters:
 - maximal message length: input limitation of the SHA-1 hash function (2^{61} bytes).
 - the private key size is 2.75 kBytes since each coefficient in the matrix is a one byte integer.
 - the public key size is 18 kBytes, since we have 26 quadratic polynomials, each with $\frac{38 \times 37}{2} + 37 + 1$ coefficients in F_{256} .
 - there is no system parameters
 - signature length: the signature is a 37-byte (296-bit) integer.
- Signature generation: Two hash functions are computed, with input the message and a 20 bytes string.

The affine function t^{-1} is computed on an input in $(F_{256})^{37}$. This computation can be made with a matrix multiplication of size 37×37 with coefficients in F_{256} with a

column vector with inputs in F_{256} too, and an addition of two columns matrix 37×1 . We assume here that the inverse of t has already been precomputed.

Then the function F^{-1} is computed. The method is fully explained in the submission. Three can be used. However, the one used in the submission implementation needs the following operations: a linear transformation in $(F_{256})^{37}$, an exponentiation in $(F_{256})^{37}$ and a multiplication in $(F_{256})^{37}$. One exponentiation using the square and multiply principle, with element in $(F_{256})^{37}$. Finally 18 linear transformation in $(F_{256})^{37}$ and 18 multiplication in $(F_{256})^{37}$ are computed.

Then the function s^{-1} is used with an entry in $(F_{256})^{37}$. Again, this can be done with a multiplication between a square matrix and a column vector, and an addition with two column vectors.

- Verification: To verify a signature, two hash functions are computed, one with the message as input and the other with the previous result. Then the function G , the public key, is computed with the signature as input. Finally a test is made to verify the equality of two 26 bytes (208 bits) strings.

3.3.6.6 SFLASH

- Submitters: Jacques Patarin, Nicolas Courtois and Louis Goubin (BULL CP8).
- Description: SFlash is a deterministic signature scheme belonging to the family of multivariate public key schemes. More precisely, SFlash is a C^{*-} scheme with a special choice of parameters. SFlash is a constrained version of Flash.
- Key generation: let F be the field F_2 . The private key is composed with two affine bijection s and t from F_{128}^{37} to F_{128}^{37} and with a 80-bit secret string. Each affine bijection is described with a square invertible matrix 37×37 and a column matrix 37×1 with coefficients not in F_{128} as for Flash, but in a subfield, isomorphic to F_2 . To generate it, two methods can be used: with the first method, each coefficient is generated at random, and a test is made to verify that the square matrix is invertible. The second method is based on the LU decomposition. A lower triangular matrix and an upper triangular matrix 37×37 are generated (each coefficient is generated at random in the subfield of F_{128}), then the product is computed. For both methods, each coefficient of the column matrix is generated at random. The public key is deduced from the private one. It is composed with a function G from $(F_{128})^{37}$ to F_{128}^{26} defined by the following relation:

$$G(X) = [t(\varphi^{-1}(F(\varphi(s(X)))))]_{0 \rightarrow 181}$$

where the notation $[\lambda]_{r \rightarrow s}$ is the string of bits of λ defined by $(\lambda_r, \dots, \lambda_s)$. The function F is defined by:

$$\forall A \in F_{128}^{37}, F(A) = A^{128^{11}+1}$$

It is easy to see that G can be represented by 26 quadratic polynomials of a special form, with coefficients in F_{128} .

- Parameters:
 - maximal message length: input limitation of the SHA-1 hash function (2^{61} bytes).
 - the private key size is 0.35 kBytes since each coefficient in the matrix is one bit integer.
 - the public key size is 2.2 kBytes, since we have 26 quadratic polynomials, each with $\frac{38 \times 37}{2} + 37 + 1$ coefficients in F_2 .
 - there is no system parameters
 - signature length: the signature is a 33 bytes (precisely 259 bits) integer.
- Signature generation: Two hash functions are computed, with input the message and a 20 bytes string.

The affine function t^{-1} is computed on an input in $(F_{128})^{37}$. This computation can be made with a matrix multiplication of size 37×37 with coefficients in F_2 with a column vector with inputs in F_{128} , and an addition of two columns matrix 37×1 , one with inputs in F_2 and the other with inputs in F_{128} . We assume here that the inverse of t has already been precomputed.

Then the function F^{-1} is computed. The method is fully explained in the submission. Three can be used. However, the one used in the submission implementation needs the following operations: a linear transformation in $(F_{128})^{37}$, an exponentiation in $(F_{128})^{37}$ and a multiplication in $(F_{128})^{37}$. One exponentiation using the square and multiply principle, with element in $(F_{128})^{37}$. Finally 18 linear transformation in $(F_{128})^{37}$ and 18 multiplication in $(F_{128})^{37}$ are computed.

Then the function s^{-1} is used with an entry in $(F_{128})^{37}$. Again, this can be done with a multiplication between a square matrix and a column vector, and an addition with two column vectors.

- Verification: To verify a signature, two hash functions are computed, one with the message as input and the other with the previous result. Then the function G , the public key, is computed with the signature as input. Finally a test is made to verify the equality of two 182 bits strings.

3.3.6.7 QUARTZ

- Submitters: Jacques Patarin, Nicolas Courtois and Louis Goubin (BULL CP8).
- Description: Quartz is a deterministic signature scheme belonging to the multivariate public key algorithms. More precisely it is a HFEV⁻ scheme. Quartz is mainly designed to provide very short signatures, only 128 bits.
- Key generation: The private key consists of two affine bijection s and t respectively in F_2^{107} and F_2^{103} . Each bijection is described with two matrices, one square invertible matrix 107×107 for s and 103×103 for t , and one column matrix 107×1 and 103×1 respectively. The coefficients of these matrices are in F_2 . The private key contains also a family (F_V) of secret polynomials from $F_{2^{103}}$ to $F_{2^{103}}$. These polynomials have a special form, and coefficients in $F_{2^{103}}$. To generate the private key, each coefficient of each polynomial is generated at random. Then each coefficient of the matrices are generated at random. A test allows to verify if the square matrices are invertible. Finally, the private key, a 80-bit secret string, is generated at random.

The public key can be deduced from the private one. Indeed, it consists of a function G from F_2^{107} to F_2^{100} , defined from the private functions F_V , t and s . Equivalently, G can be viewed as 100 quadratic polynomials P_i with coefficients in F_2 .

- Parameters
 - maximal message length: input limitation of the SHA-1 hash function, i.e. 2^{61} bytes.
 - private key length: 3 kBytes
 - public key length: 71 kBytes
 - signature length: 128 bits i.e. 16 bytes
 - there is no system parameters

- Signature generation:

To generate a signature, 3 hash computations are made, the first one with the message as input and the two others with a 20 bytes string. Then 4 iterations of the following are executed:

- Y is the XOR with 100 bits values.
- W is the SHA-1 with input $Y||\Delta$ of length 180 bits
- one affine transformation $B = t^{-1}(Y||R)$. This step needs one multiplication of a square matrix 103×103 with a column vector 103×1 , and an addition of two column vectors 103×1 , all with coefficients in F_2 . B is then a column vector 103×1 .

- solving one polynomial equation $F_V(A) = Z$. This step is the most time consuming. The method to use is explained in the submission. First, if this equation has not a unique solution, the value W is replaced by $\text{SHA-1}(W)$ and the previous step is executed again. To verify the number of solutions of this equation, one gcd can be computed, and this needs the computation of Z^{2^i} for $0 \leq i \leq 103$ modulo $F_V(Z) - B$. The solution, if it exists, can be viewed as a column vector 103×1 . The probability of having a unique solution is approximately $1/e$. Then, with two trials in average, the equation is solved.
- Then $X = s^{-1}(A||V)$ is computed. This can be done with a multiplication of a square matrix 107×107 with a column vector 107×1 , and an addition of two column vectors 103×1 . All the coefficients are in F_2 .
- \tilde{S} is defined as the 100 first bits of X , and X_i as the last 7 bits.

The signature is defined as $S = \tilde{S}||X_4||X_3||X_2||X_1$ of length 128 bits. We remark that the equation $F_V(A) = Z$ may not have a unique solution, with probability $1 - 1/e$ approximately. Since the algorithm has to solve this equation four times, the probability that the algorithm fails and the message doesn't have a signature is about 2^{-83} .

- Verification: unlike the signature, the verification step is very fast. 3 hash computations are made, the first with the message as input, and the two others with a 160 bits string. Then, the signature S is written as $\tilde{S}||X_4||X_3||X_2||X_1$. U is initialized to \tilde{S} and the following is executed 4 times:
 - $Y = G(U||X_i)$. This computation is very fast, since this can be done with XORs of 100 bits integers. In average, we have 1472 such XORs.
 - $U = Y \oplus H_i$ is the XOR with 100 bits strings.

The last step is the verification that the value U returned equals the 100 bits string with 0.

3.3.6.8 Summary

Table 7 and figure 1 lists the sizes of the data handled by the algorithms. The most important are the signature size which can cause an overhead for a message transmission, and the public key size which has to be handled by the public key infrastructure. When (part of) the public key is needed to sign, we include it in the private key length.

Most submitted signature scheme have their security and performance depend on the choice of some parameter. For this table, the choices have been 1024 bits moduli for two factors based schemes (ACE Sign and RSA-PSS) 1152 bits moduli for three factors based schemes (ESIGN) and 163 bits base field for discrete logarithm based schemes (ECDSA). RSA public exponent is fixed to 3.

Those schemes use a hash function which gives a limitation for the message length e.g. 2^{61} bytes for SHA-1. Only ACE-Sign has signature length depending on the message length. We consider that the message are between 20 bytes and 1 Mbyte.

Table 7: Data lengths

Scheme	Public Key	Private key	Signature
ACE-Sign	620 bytes	748 bytes	425 to 705 bytes
ECDSA	48 bytes	24 bytes	48 bytes (can be reduced to 326 bits)
ESIGN	145 bytes	96 bytes	144 bytes
RSA-PSS	128 bytes	320 bytes	128 bytes
Flash	19266 bytes	2822 bytes	37 bytes
SFlash	2409 bytes	362 bytes	37 bytes (can be reduced to 259 bits)
Quartz	71 to 90 kB	3914 bytes	16 bytes

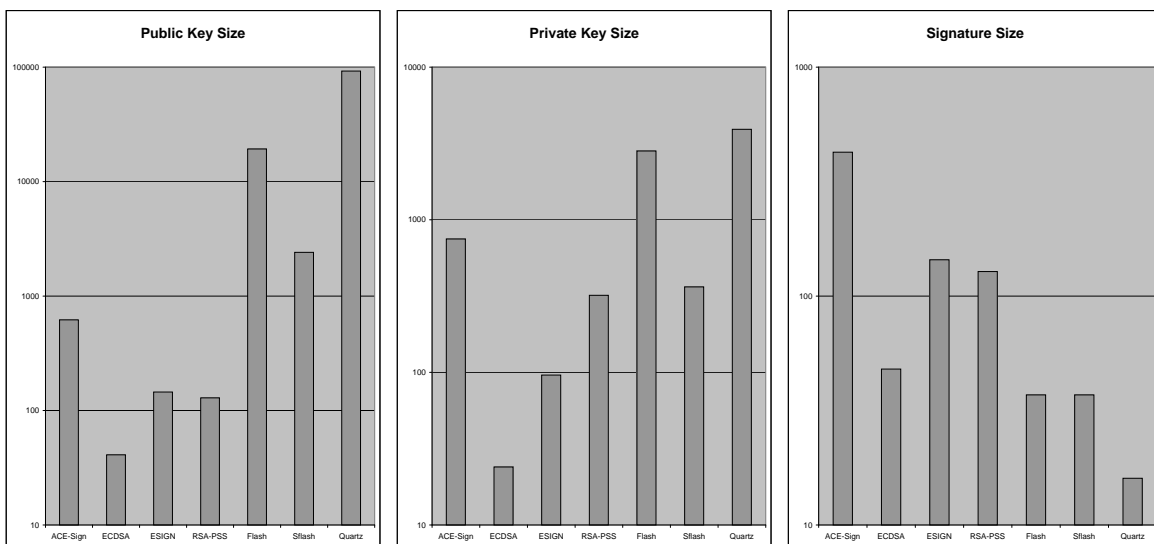


Figure 1: Data lengths

3.3.7 Asymmetric Identification Schemes

There is only one candidate in this category: GPS.

3.3.7.1 GPS

- Submitters: Guillaume Poupard, Olivier Baudron, Fabrice Boudot, Philippe Bourel, Emmanuel Bresson, Johann Corbel, Nicolas Courtois, Laurent Frisch, Henri Gilbert, Marc Girault, Louis Goubin, Jean-François Misarsky, Phong Nguyen, David Pointcheval, Jacques Patarin, Jacques Stern and Jacques Traoré, (BULL CP8, France-Télécom and La Poste).
- Description: A public key identification scheme based on the discrete logarithm problem. It is a revised version of the Schnorr scheme.
- Parameters: A modulus n of either 768 bits or 1024 bits, and integers g (same number of bits as n) B (suggested 32 bits), S (at least 140 bits) and A (at least 252 bits but depends on the size of B and S).
- Generation of public and private keys
 $n = pq$, with $\log_2 n = k$ and $\log_2 p = \log_2 q = \frac{k}{2}$. s a random number in $[0, S]$. And the prover had to compute $I = g^{-s} \pmod n$. The cost of keys generation is one inversion modulo n , one exponentiation modulo n , generation of two prime numbers of fixed length and generation of a random number.
- Number of operations for one round by the Prover
The prover has to generate a random number r and to compute $g^r \pmod n$ and finally to compute $y = r + cs$. The cost of this part consist of one exponentiation modulo n and one addition and one multiplication in \mathbb{Z} .
- Number of operations for one round by the Verifier
The verifier has to generate a random number c in $[0, B[$, and to compute $g^y I^c \pmod n$ and to verify that y is really in $[0, A + (B - 1)(S - 1)[$
- Prover communicates two numbers, one with less than k bits and one in \mathbb{Z} . Verifier sends one value to the prover with size $\log_2 B$.
- Previous operations for verifier and prover has to be repeated l time.

4 Claimed Performance

4.1 Block Ciphers and MACs

We present here performance results, for 128 bit key when possible, as claimed by the submitters. Since the first phase on the Nessie project only looks at performance using C code, we can only value those implementations written in ANSI C.

4.1.1 CS-Cipher

Processor	Encryption (1) (cycles/block)
Pentium	4053.12

(1) efficiency results for standard C implementation on a Pentium 133MHz.

4.1.2 Hierocrypt-L1

Processor	Encryption/Decryption (1) (cycles)	Encryption/Decryption (cycles) with Key Setup
Pentium	199	374

(1) best efficiency results in 10 trials to carry out 1000000 block encryptions in ECB mode for 128-bit key version on Pentium III 650 MHz, running MS Windows 98 CE, and MSVC++ 6.0. Data obtained from Proceedings of the 2nd NESSIE Workshop.

4.1.3 IDEA

We give here the performances claimed by the submitters for their candidates.

Processor	Mbits/s
90 MHz Pen- tium	4.2
366 MHz Pen- tium II	31
600 MHz Pen- tium III	61

In the following we present the results given by Lipmaa.

Processor	Block size	Cycles	MBytes/s
Pentium III	4x64	440	55.5
Pentium MMX	4x64	543	45.0
Pentium MMX	64	358	17.0

4.1.4 Khazad

Khazad is an involutinal cipher, so the effort required for encryption and decryption is the same. Because of this, decryption is not obtained by applying the encryption components in reversed order, and so the key schedules for encryption and decryption are not identical.

Processor	Language	Key Scheduling (cycle)	Encryption Decryption (cycle)	Encryption Decryption (Mbps)
Pentium III (1)	ANSI C	717(encrypt) 1206(decrypt)	536	65.7

(1) IBM PC/ AT compatible PC, Intel Pentium III, 550 MHz.

4.1.5 MISTY1

Processor	Language	Key Scheduling (cycle)	Encryption (cycle)	Encryption (Mbps)
Pentium II (1)	ANSI C	170	450	71.1
Pentium II (2)	ANSI C	300	300	106.7

(1) IBM PC/ AT compatible PC, Intel Pentium II, 500 MHz, 128 MB memory, Windows 98, straightforward implementation.

(2) IBM PC/ AT compatible PC, Intel Pentium II, 500 MHz, 128 MB memory, Windows 98, optimised implementation requiring more memory.

4.1.6 NUSH

Processor	Block Size (bits)	Encryption (cycles/block)	Key Setup (cycles/key)
Pentium	64	180	64

(1) efficiency results for C implementation.

4.1.7 DES

The DES performance had been well studied in the past, we give here some results to compare with the NESSIE candidates.

Origin	Processor	Block size	Cycles	MBit/s	Mbyte/sec
Bosselaers	90 MHz Pentium	64	340	16.9	2.12

4.1.8 Anubis

Anubis is an involutinal cipher, so the effort required for encryption and decryption is the same. Because of this, decryption is not obtained by applying the encryption components in reversed order, and so the key schedules for encryption and decryption are not identical.

Processor	Language	Key Scheduling (cycle)	Encryption Decryption (cycle)	Encryption Decryption (Mbps)
Pentium II (1)	ANSI C	3352(encrypt) 4527(decrypt)	589	119.5

(1) IBM PC/ AT compatible PC, Intel Pentium III, 550 MHz.

4.1.9 Camellia

Processor	Language	Key Scheduling (cycle)	Encryption (cycle)	Decryption (Mbps)
Pentium III (1)	Assembly	160	371	241.5
Pentium II (2)	ANSI C	263	577	66.6

(1) IBM PC/ AT compatible PC, Intel Pentium III (700MHz), 256KB on L2 cache, FreeBSD 4.0R, 128MB main memory.

(2) IBM PC/ AT compatible PC, Intel Pentium II (300MHz), 512KB L2 cache, Windows95, 160MB main memory.

4.1.10 Grand Cru

Processor	Encryption (1) (cycles/block)	Decryption (cycles/block)	Key Setup (cycles/setup)
Pentium XEON	45000 65000	65000 90000	200000 300000

(1) efficiency results of reference ANSI C implementation for 128-bit key version using gcc on an Intel Pentium 200 MHz, running Linux 2.0.38, and on an Intel XEON 550 MHz, running Linux 2.2.14.

4.1.11 Hierocrypt-3

Processor	Encryption/Decryption (1) (cycles)	Encryption/Decryption (cycles) with Key Setup
Pentium	404	726

(1) best efficiency results in 10 trials to carry out 1000000 block encryptions in ECB mode for 128-bit key version on Pentium III 650 MHz, running MS Windows 98 CE, and MSVC++ 6.0. Data obtained from Proceedings of the Second NESSIE Workshop.

4.1.12 Noekeon

Note that Noekeon has two key schedules. In direct mode the user-selected key is used directly, and so requires no operations before encryption or decryption can start. In indirect mode the user-selected key is encrypted once using the all-zero key, and the ciphertext becomes the working key. This key schedule thus takes the same time as one encryption.

Processor	Language	Key Scheduling (cycle)	Encryption Decryption (cycle)	Encryption Decryption (Mbps)
Pentium II (1)	ANSI C	0(direct) 525(indirect)	525	48.8

(1) IBM PC/ AT compatible PC, Intel Pentium II, 200 MHz, Windows NT 4.0, Microsoft Visual C/C++ 6.0 compiler.

4.1.13 NUSH

Processor	Block Size (bits)	Encryption (cycles/block)	Key Setup (cycles/key)
Pentium	128	340	112

(1) efficiency results for C implementation.

4.1.14 RC6

We present the performances claimed by the submitters of RC6, in cycles/byte for 20-round and 128 bit-block version, for C code.

Processor	Compiler	cycles/byte
Pentium II	Borland	39
Pentium Pro	MSVC	30
Pentium Pro	MSVC + opt.	16
Pentium Pro	GCC	26
Pentium Pro	GCC + opt.	23

4.1.15 SAFER++

Processor	Encryption (1) (clock cycles/block)	Key Schedule (2) (clock cycles/block)
Pentium	640	2333

(1) efficiency results for 128-bit key version on Pentium III 667 MHz, running MS Windows 2000, and MSVC++ 6.0 (optimized source code).

(2) efficiency of reference implementation (not optimized).

4.1.16 SC2000

Processor	Encryption (1) (clock cycles/block)	Decryption (clock cycles/block)	Key Schedule (clock cycles/block)
Pentium	528	539	524

(1) best efficiency results for 128-bit key version in MSVC++ 6.0 + Assembly (inline) on an Intel Pentium III 550 MHz, running Microsoft Windows NT 4.0; key schedule in C language.

4.1.17 Rijndael

Some performance results exist for this block cipher, which was well studied in the AES process. We give some significant results to compare with NESSIE candidates, in addition to our own tests.

Origin	Processor	Block size	Cycles	MBytes/s
Lipmaa	Pentium II/III	128	229	53.3

4.1.18 Shacal

The Shacal submitters estimate the computational efficiency at 2800 cycles per 20 bytes block encryption (block size) , 2330 per 20 byte block decryption and 3200 per 64 byte key setup, on PC platform AMD K6 processor at 233 MHz.

4.1.19 Two-Track MAC

Processor	MAC (1) (cycles/64-byte block)	Key setup (cycles/64-byte block)
Pentium	6135	10

(1) efficiency results for non-optimized reference C code.

4.1.20 UMAC

Processor	Message Size (bytes) (1)			
	43	256	1500	$256 \cdot 2^{10}$
Pentium	16.3	3.8	2.1	1.9

(1) efficiency results in cycles/byte on a 700 MHz Pentium III under gcc 2.95, mixed C/Assembly. Figures for UMAC32 submitted to NESSIE and called simply UMAC. Data obtained from Proceedings of First NESSIE Workshop.

4.2 Asymmetric Primitives

The performances announced by the submitters are not very different from the results of tables 14, 15 and 17.

4.2.1 Asymmetric Encryption Schemes

Chinese remainders are used for RSA. The parameters for EPOC are close to Type B.

Table 8: Claimed performance of asymmetric encryption schemes.

Scheme	Key Setup	Encryption	Decryption	Architecture
ACE-Encrypt	14 sec.	230 ms	97 ms	Pentium @ 266
PSEC-1		4.4 ms	4.4 ms	Pentium @ 700
PSEC-2		4.4 ms	4.4 ms	Pentium @ 700
PSEC-3		4.4 ms	2.4 ms	Pentium @ 700
ECIES		5.8 ms	4.4 ms	Pentium @ 200
EPOC-1		13 ms	20 ms	Pentium @ 700
EPOC-2		10 ms	17 ms	Pentium @ 700
EPOC-3		10 ms	7 ms	Pentium @ 700
RSA-OAEP		1 ms	27 ms	Celeron @ 450

4.2.2 Asymmetric Digital Signature Schemes

Table 9: Claimed performance of digital signature schemes

Scheme	Key Setup	Signature	Verification	Architecture
ACE-Sign	36 sec.	62 ms	73 ms	Pentium @ 266
ECDSA — $GF(2^{163})$	1.5 ms	2.1 ms	4.1 ms	Pentium @ 400
ECDSA — $GF(p)$	2.1 ms	2.6 ms	6.5 ms	Pentium @ 400
ESIGN		3.4 ms	0.4 ms	Pentium @ 700
RSA-PSS	1.9 sec.	27 ms	3 ms	Celeron @ 450
Flash		49 ms	50 ms	Pentium @ 500
SFlash		44 ms	50 ms	Pentium @ 500
Quartz		30 sec.	50 ms	Pentium @ 500

4.2.3 Asymmetric Identification Schemes

Table 10: Claimed performance of asymmetric identification schemes

Scheme	Key Setup	Commitment	Answer	Verification	Architecture
GPS	0.7 sec.	10 ms	1 μ s	12 ms	Pentium @ 450

5 Practical Software Implementation

5.1 Measurements for Symmetric Primitives

5.1.1 Description of Tools

5.1.1.1 Block Ciphers Tests For several block cipher candidates, performance was skipped because of security level did not seem to match up to the requirements imposed by the NESSIE call (cf. D13 [8]). We only tested the candidates which passed the first security evaluation tests (cf. *ibidem*). The tool for block ciphers is made up of two parts. In a first time we generated 100 000 random keys, we did the key schedule for each of them and we measured the time. The second part of the test consisted of generating 100 random keys and expanding each of them; we generated 100 000 plaintexts that we encrypted and decrypted using these keys. We measured the encryption and decryption times separately. We verified that the decrypted text is equal to the plaintext. The following performance results are reported: mean time for key expansion, encryption time for each of the 100 keys as well as mean encryption time (over all keys), and finally mean decryption time.

5.1.1.2 Stream Ciphers Tests The first part of tests consists of measuring time for expansion of 100 000 random keys. In a second time, we generated 1000 random keys, we expanded them, and we measured the encryption time as follows: we only measured time for successive keys generations because the second part of encryption is a xor between key stream and plaintext. Differences between candidates in this part of encryption consist of size of output and the size of words xored. In smart card, we have to do this part of computation because according to bit size of processor used the xor of two numbers do not take the same time. For example, on a 32-bit processor a xor between two words of 32 bits takes one clock round, but on an 8-bit processor, a xor between the same words takes four times more clock round. To do the same tests for each candidate, we introduce the constant “KEYSTR_BITS_ROUND”, which is the bit size of output of the function “NESSIEkeystream”. For each candidate we complete the value of “KEYSTR_BITS_ROUND” that indicates the bit size of output. Finally we invoke “NESSIEkeystream” $100\ 000 * 64 / \text{KEYSTR_BITS_ROUND}$ times and measure the time per key and key stream generation mean.

We only measure the time for encryption and not for decryption, because all stream ciphers work in the same way for encryption and decryption. Results are written in a result file.

5.1.1.3 Tests for MAC and Hash Functions For these candidates, we propose at first time to measure time to key expansion, with 100 000 random key. Secondly, we measure authentication time for 1000 random keys and for each of them 10 000 random plaintexts.

For the hash function Whirlpool we do the same test, except that there is no key.

5.1.1.4 Conclusion We had proposed tools for Performance Evaluation I (WP4), in this technical report, we gave explanations of our tools. These tools will complete the template for Performance Evaluation (NES/DOC/UCL/WP4/002/c) which was a theoretical approach.

5.1.2 Block Ciphers Testing

We compare Block Cipher Schemes on a PC Dell Optiplex GX115, Pentium III, 850 MHz, 256MB RAM os Windows 2000 with Visual C++ (denoted PIII), on a PC DELL Powerege 6300 2 processors PIII Xeon @550 MHz (but we only use 1), 1MB cache Windows NT Server-Client Microsoft, with Visual C++ 6.0 (denoted Xeon), and finally on an UltrasparcII processor 450 MHz, 4MB cache, Solaris 2.8, Sun SWC 5.0 compiler (denoted Sparc). See Table 11.

Nimbus and Q have not been included because of their security weaknesses. Also Safer++ was optimised for encryption only.

Table 11: Block ciphers performance results

Name	Key setup (K cycles/key)			Encryption (cycles/byte)			Decryption (cycles/byte)		
	Sparc	Xeon	PIII	Sparc	Xeon	PIII	Sparc	Xeon	PIII
Cs-cipher	4.2	5.3	5.6	236	684	733	218	468	501
Hierocrypt-L1	39	43	116	95	95	120	118	117	120
Idea	0.6 ¹	0.9 ¹	1.2 ¹	141	110	149	148	110	149
Khazad	0.76	1.2	0.85	90	81	62	90	81	62
Misty1	0.2	0.18	0.17	104	101	107	106	94	102
Nush64 (legacy)	2.2	1.5	1.4	87	111	123	86	97	113
Safer++ (legacy)	2.9	3	4.7	220	157	240	204	232	350
DES	15	16	17	130	103	117	125	98	111
Anubis (on 12 rounds)	5.3	4.2	5.5	88	50	53	84	51	54
Camellia/128	1.8	3.0	4.2	154	172	161	151	238	212
Grandcru	110	150	181	1620	3400	2600	2160	5400	3900
Hierocrypt-3/128	150	183	605	97	63	150	120	70	170
Noekeon	1	1.7	2.2	104	135	108	106	136	108
Nush128	4	2.8	2.3	79	45	110	80	42	90
RC6 (default)	3.4	2.9	6.1	127	49	53	126	49	52
Safer++/128	2.0	2.3	7.3	142	89	260	158	93	314
SC2000 (128)	3.4	1.2	1.5	75	475	440	80	480	450
Rijndael	0.27/0.85 ²	0.53/2.1 ²	0.68/2.0 ²	72	59	54	81	69	56
Shacal	1.1	1	2.7	95	62	66	n. a.	n. a.	n. a.

¹For encryption.

²Rijndael has a separate key schedule algorithm for encryption and decryption. The former is indeed much faster than the latter.

5.1.3 Stream Ciphers Testing

We compare stream ciphers on three different platforms: a Pentium III, 850 MHz, 128 MB RAM os: windows 2000 (noted PIII Siemens), sun4u sun Ultra 30 (UltraSparc-II 248 MHz), 128 MB RAM os: SUNOS 5.6 (noted Sparc Siemens) and a Pentium III, 850 MHz, 256MB RAM os Windows 2000 (noted PIII UCL). See table 12.

Lili-128 and Leviathan have not been included because of their security weaknesses.

Table 12: Stream ciphers performance results

Streamciphers	Key setup (K cycles/key)			Key stream generation (cycles/byte output)		
	PIII UCL	Sparc Siemens	PIII Siemens	PIII UCL	Sparc Siemens	PIII Siemens
BMGL (40bit word) ¹	26	37	13	2900	1100	1200
Snow/128 (32bit word)	8.4	2.7	2.0	38	10	8.1
Snow/256 (32bit word)	8.2	3.6	2.3	42	10	8.1
Sober-t16/128 (16bit word)	3.7	2.2	1.6	75	21	29
Sober-t32/128 (32bit word)	3.4	1.9	0.93	22	9.7	7.3
Arcfour	16	3.3	3.8	73	19.7	24

¹We used the reference Rijndael code written by Brian Gladman in his evaluation of the AES candidates. Also, we used $n = 256$ and $m = 40$ as parameters, as suggested by the submitters.

5.1.4 MAC and Hash Functions Testing

See Table 13.

Table 13: MACs' and hash functions performance results

Algorithm	Key setup (K cycles/key)			Performance (cycles/(authenticated or hashed) byte)		
	PIII UCL	Sparc Siemens	PIII Siemens	PIII UCL	Sparc Siemens	PIII Siemens
Two-Track MAC	0.5	0.05	0.34	84	35	36
UMAC	113	103	90	20	67	14
Whirlpool	n. a.	n. a.	n. a.	110	567	200
SHA-1	n. a.	n. a.	n. a.	44	27	23

5.2 Measurements for asymmetric primitives

We chose the smallest parameters that by the submissions claimed to have a security of 2^{80} , i.e. 1024 or 1152 bits for factorisation or discrete log based schemes and 160 bits for elliptic curves. RSA public exponent is 3, Type B parameters are chosen for EPOC, elliptic curves are over the binary prime field $\text{GF}(2^{163})$ for ECIES and ECDSA and over a prime field for PSEC. Note that OEF (Optimal extension fields) are more efficient but are believed to weaken the discrete logarithm assumption.

5.2.1 Asymmetric Encryption Schemes

Due to lack of resources in phase I, the performance comparison has not been actually measured on computers. Table 14 shows estimations of performance for encryption and decryption of 20 bytes messages on a typical Pentium desktop at 500 MHz, based on the theoretical analysis of the previous section.

Table 14: Estimated performance of asymmetric encryption schemes

Scheme	Encryption	Decryption
ACE-Encrypt	50 ms	45 ms
PSEC-1	5 ms	5 ms
PSEC-2	5 ms	5 ms
PSEC-3	5 ms	2.5 ms
ECIES	5 ms	2.5 ms
EPOC-1	15 ms	20 ms
EPOC-2	10 ms	15 ms
EPOC-3	10 ms	7 ms
RSA-OAEP	1 ms	11 ms
Rabin-SAEP	0.5 ms	11 ms

5.2.2 Asymmetric Digital Signature Schemes

We compared the results of asymmetric digital signature schemes candidates on two different platforms: a PC (Pentium III, Katmai core at 550 MHz with 512KB cache, running FreeBSD 4.2 and gcc 2.95.2) and an Ultra 5 workstation (UltraSparc III at 333 MHz with 2MB cache, running Solaris 2.7 in 32 bits mode and the best of gcc 2.95.2 and Sun SWC 5.0). Tests were also made on an Alpha running Linux and a G4 running MacOS X, but some code was not directly portable on those architectures. By the way, figures below show that signature schemes performance depends only slightly on the workstation used.

We ran the key generation, signature and verification algorithms an adequate number of time at took the mean of the cpu user time. We took the code sent by the submitters, with minor modifications, and we measured the time for the inner functions and not the time to access data files.

We ran Quartz signature with various messages because its running time depends on the message signed. Usually it is between a third and twice the mean time.

We tested the signature and verification with three sizes of messages : 20 bytes, 64 Kbytes and 1 Mbyte. Message length of 20 bytes shows the cost of the core algorithm. For a message length of 1 Mbyte the major cost is often the computation of the hash of the message. RSA-PSS, ACE, ECDSA, FLASH, SFLASH and QUARTZ use only one SHA-1 of the whole message, ESIGN uses four SHA-1 (note that P1363 discussions may change ESIGN to reduce this cost to three or one SHA-1).

Table 15 show the main results. Figures 7, 8 and 9 show the influence of message length and architecture.

Please note that all the submitted code was already well optimised, with the exception of RSA-PSS which timings can be greatly improved by using an optimised library for large numbers. We give below the timings using the openssl 0.9.4 library.

Table 15: Time needed in seconds on the Pentium III for a 20 bytes message.

Scheme	Key generation	Signature	Verification
ACE-Sign	14 sec.	29 ms	20 ms
ECDSA	1.6 ms	1.9 ms	4.9 ms
ESIGN	0.20 sec.	5.5 ms	0.69 ms
RSA-PSS	4 sec. 0.73 sec. using openssl	110 ms 11 ms using openssl	2.2 ms 0.7 ms using openssl
Flash	3.5 sec.	6.5 ms	0.93 ms
SFlash	4.0 sec.	4.6 ms	0.49 ms
Quartz	4.0 sec.	14 sec.	0.42 ms

5.2.3 Asymmetric Identification Schemes

The GPS C implementation uses the GMP library, available for example at <http://www.swox.com/gmp>. Submitters proposed C, java and 6805 microprocessor based

on smart card implementations. GPS is set up in five parts: the generation of the parameters, the generation of public and private keys, the commitment, the answer, and finally the verification. Submitters delivered a demonstration program, available on the internal web site of NESSIE, which measures the time needed to compute commitments, answers, and to perform verifications. We used the same parameters as suggested in the submission which guarantee enough security, namely $|S| = 160$, $|B| = 35$, and $|A| = 275^2$. We ran the tests on three platforms:

Table 16: Description of GPS Test Platforms

Platform	Processor	Frequency (MHz)	RAM (MB)	SWAP (MB)
Ultra Sparc 10	Ultra SPARC II-i	440	1024	1024
Ultra Sparc 2a	2x Ultra SPARC II	2 x 296	512	515
Ultra Sparc 2b	2x Ultra Sparc	2 x 168	128	377

We got the following results:

Table 17: Performance times for GPS

Platform	Generation of parameters	Public/Private keys generation	Commitment	Answer	Verification
Ultra Sparc 10	1031 <i>ms</i>	9.94 <i>ms</i>	16.69 <i>ms</i>	1.09 μs	19.55 <i>ms</i>
Ultra Sparc 2a	1542 <i>ms</i>	14.77 <i>ms</i>	24.82 <i>ms</i>	1.64 μs	29.06 <i>ms</i>
Ultra Sparc 2b	2575 <i>ms</i>	25.18 <i>ms</i>	42.35 <i>ms</i>	2.83 μs	49.50 <i>ms</i>

GPS submitters claim that GPS was well designed for smart card applications, in view of speed measurements and code size (only 300 bytes). We obtained very high speed measurements: for smart card applications, the card only need compute the answer which on our platforms takes between 1 and 3 μs . If we use a much larger number for $|A|$, the speed for the answer is quite the same and only the commitment and the verification are slower. In the future, we will realise a smart card implementation to measure the speed in this precise context, but we can predict that it will be very fast.

²With the notations of the submission, A/BS must be large enough to guarantee the statistical zero knowledge property, $|A| \geq |S| + |B| + 80$, which is the case with the chosen values.

6 Comparisons Between Candidates

The results of performance evaluation will be taken into account in the selection process for the second phase.

6.1 Block Ciphers

See figures 2 et 3. Note that we do not present the block ciphers' key setup time since we are mostly interested in the encryption/decryption feature of block ciphers at this stage.

- **Overall fastest.** RC6 (128-bit), Anubis (128-bit), Khazad (64-bit) and Shacal (160-bit), except that the first two are slower on Sun machines. On Sun machines fastest are Nush (128- and 64-bit). A good blend of the two would be Misty1 (64-bit) or Shacal (160-bit).
- **Overall slowest.** Grand Cru (128-bit), CS-cipher (64-bit) and SC2000 (128-bit) on all machines.
- **General comment.** Encryption/decryption perform similarly except for Safer++ (64-bit) and Camellia (128-bit), where decryption is slower than encryption.

6.2 Stream Ciphers

See figures 4 (key setup) and 5 (key stream generation).

- **Overall fastest.** For key setup: Sober-t32 and Sober-t16. For key stream generation: Sober-t32 and Snow (128- and 256-bit key size).
- **Overall slowest.** For key setup: BMGL by a factor of 10. For key stream generation: BMGL.
- **General comment.** All ciphers seem to be more performing than Arcfour, with the exception of Sober-t16 which is a trifle slower. Discrepancies appear in the two Pentium III measurements probably due to different optimisation variables. However the comparison between performance produces the same results, since the ratio between measurements on these two machines is almost constant.

6.3 Message Authentication Codes and Hash Functions

See figures 4 (key setup for MACs') and 6 (authentication/hashing performance).

- **Overall fastest.** For key setup: Two-Track-Mac is very fast. For message authentication: Umac is very fast on the PCs, slow on the Sun.
- **Overall slowest.** For key setup: Umac is exceedingly slow. For message authentication: Two-Track-Mac is slower on PCs but faster on the Sun.

- **Whirlpool.** Quite slow, especially on the UltraSparc, when compared to SHA-1.
- **General comment.** One can expect a benchmark value of about 50 cycles/byte for CBC-MAC with Rijndael, giving Umac the upper edge on PCs and Two-Track-Mac on the Sun machines. The Whirlpool implementation works strangely and because it is a pure 64-bit implementation it should be run on a 64-bit machine.

6.4 Asymmetric Encryption Schemes

Table 14 shows estimations of performance.

- **Overall fastest.** PSEC-3/ECIES are the fastest scheme. Rabin-SAEP and RSA-OAEP with exponent 3 have the fastest encryption but slow decryption.
- **Overall slowest.** ACE-Encrypt.
- **General comment.** All primitives submitted to NESSIE have similar performance and this will not make a difference for encryption of medium to long messages.

6.5 Asymmetric Digital Signature Schemes

Table 15 show the main results. Figures 7, 8 and 9 show the influence of message length and architecture. We discuss below some results on the comparison of the candidates.

- **Overall best.** It is obvious that the schemes with the best performance are ECDSA and ESIGN.
- **FLASH vs. SFLASH.** SFLASH is only a bit faster than FLASH, because the field \mathbb{F}_{2^7} is slightly more difficult to handle than \mathbb{F}_{2^8} .
- **QUARTZ vs. others.** From the performance point of view, QUARTZ is similar to FLASH and SFLASH, with two major changes: a *very* slow signature generation and a *very* short signature.
- **ESIGN vs. FLASH and SFLASH.** They have similar performance, with a faster key setup for ESIGN. FLASH and SFLASH could have an advantage on smart card, an implementation is needed to evaluate this.
- **ESIGN vs. ECDSA.** ECDSA has faster key generation, faster signature generation and shorter signature and keys, but slower signature verification. Usage and comparison on smart card will help to choose.
- **ACE-Sign vs. RSA-PSS.** The implementation of ACE-Sign uses some “bits to word” conversion that perform very bad on big-endian machines, so only the timings on Pentium are meaningful. The implementation of RSA-PSS was not optimised so the number measured cannot help to compare those algorithms, but the operations

they make are very similar (cf. the theoretical analysis) and a comparison can be made. ACE-Sign is definitely slower than RSA-PSS.

- **RSA-PSS vs. ESIGN.** Similar care have to be taken because of the non-optimisation of the implementation of RSA-PSS. Still, RSA-PSS has slower key setup and signature generation than ESIGN. This can be improved using a three-prime modulus, but will still be slower than ESIGN.

Since it is possible to adapt the algorithms such that all of them will use the same hash function on the message, their performance will be the same for long messages (more than 1 Mbyte).

References

- [1] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal of Computing*, 15(2):850–864, 1986.
- [2] Ronald Cramer and Victor Shoup. Signature Schemes Based on the Strong RSA Assumptions. *6th ACM Conf on Computer and Communications Security*, 1999.
- [3] P. Crowley and S. Lucks. Bias in the leviathan stream cipher. In *Proceedings of FSE 2001*, 2001.
- [4] J. Daemen and V. Rijmen. AES proposal: Rijndael. Selected as the Advanced Encryption Standard. Available from <http://www.nist.gov/aes>.
- [5] T. Johansson F. Jönsson. A fast correlation attack on lili-128. Technical report, Lund University report, 2001.
- [6] O. Goldreich and L. A. Levin. A hard core predicate for any one way function. In *Proceedings, 21st ACM STOC*, pages 25–32, 1989.
- [7] A. Menezes, P. Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.
- [8] NESSIE partners. Security Evaluation I. Technical report, NES/DOC/RHU/WP3/D13/1, 2001.
- [9] D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13:361–396, 2000.
- [10] Marcus Schafheutle. Statistical attacks on the stream cipher LEVIATHAN. Technical report, NES/DOC/SAG/WP3/027/1, 2001.

A Elliptic Curve Cryptosystem

This is a list of all routines used in the schemes ECIES and ECDSA. They are all very rapidly implemented on a PC except the last two.

1. Bit string to octet string conversion (BO). Cost is nought.
2. Octet string to bit string conversion (OB). Cost is nought.
3. Integer to octet string conversion (IO). octet: byte length m len bytes compute the integer x in base 2 and group coefficients into octuplets, padding to zeros on the left. Total cost is $O(\log x)$ or nought on a PC.
4. Octet string to integer conversion (OI). Cost is nought on a PC.
5. Field element to integer conversion (FI). Cost is nought on a PC.
6. Octet String to field element conversion (OF). Cost is nought on a PC.
7. Field element to octet string conversion (FO). Cost is nought on a PC.
8. Elliptic curve point to octet string conversion (EO). Without point compression: cost is nought on a PC. With point compression: cost is nought on a PC for prime fields, one field (in \mathbb{F}_{2^m}) inversion and one multiplication for binary fields (cost is in this case $O(m^3)$).
9. Octet string to elliptic curve point conversion (OE). Without point compression: checking point lies on curve requires $O(\log^3 q)$ operations. With point compression: compute an element of the field ($O(\log^3 q)$) then solve a quadratic equation.

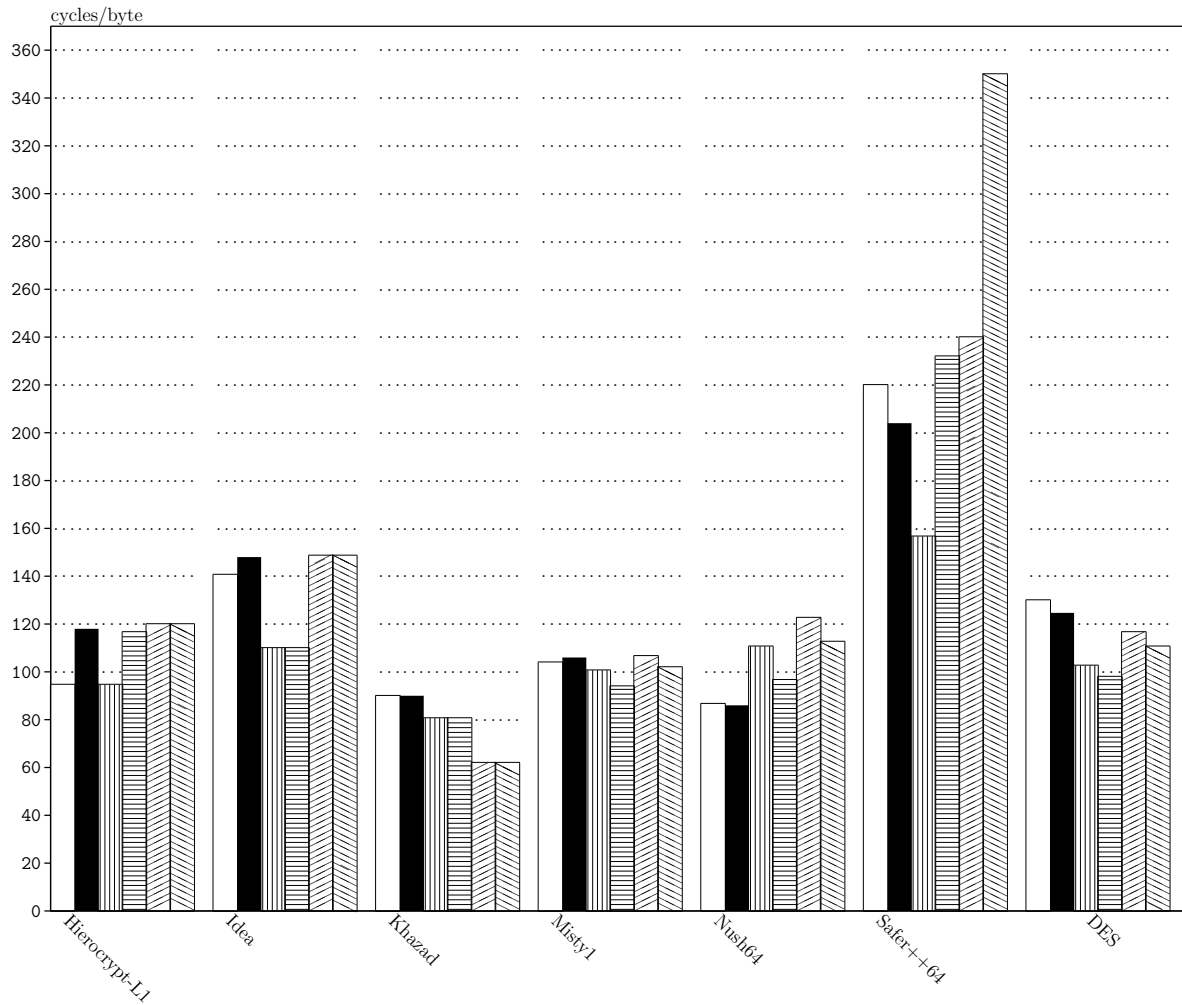


Figure 2: Block ciphers (64-bit), without CS-cipher and Hierocrypt-L1: encryption/decryption results

Legend:

Sparc (encryption),
 Xeon (encryption),
 PIII (encryption),
 Sparc (decryption),
 Xeon (decryption),
 PIII (decryption).

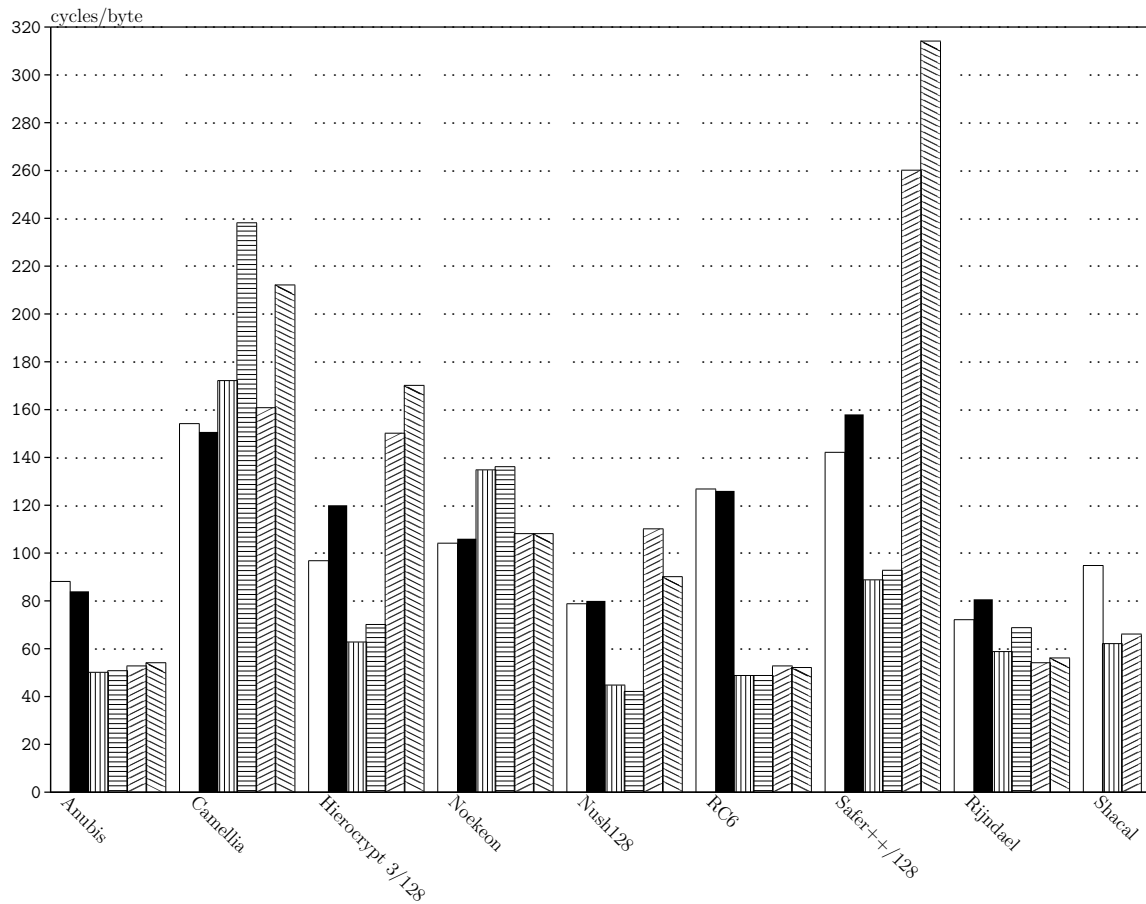


Figure 3: Block ciphers (128- and 160-bit) without Grand Cru and SC2000: encryption/decryption results

Legend:

Sparc (encryption),	Xeon (encryption),	PIII (encryption),
Sparc (decryption),	Xeon (decryption),	PIII (decryption).

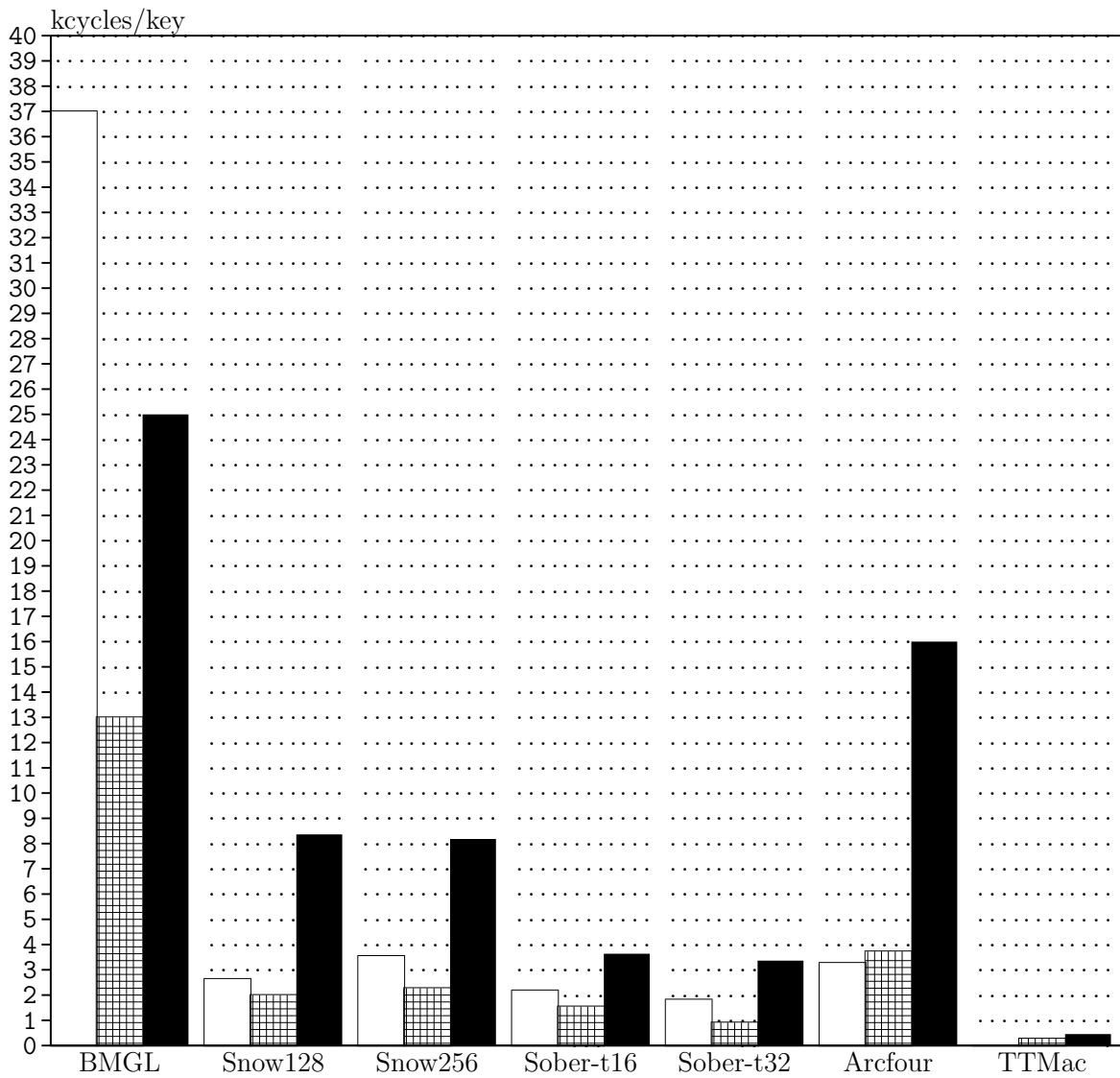


Figure 4: Stream ciphers and MACs' performance (without UMAC): key setup

Legend:

- UltraSparc 248 MHz @ Siemens, ▤ PIII 850 MHz @ Siemens,
- PIII 850 MHz @ UCL.

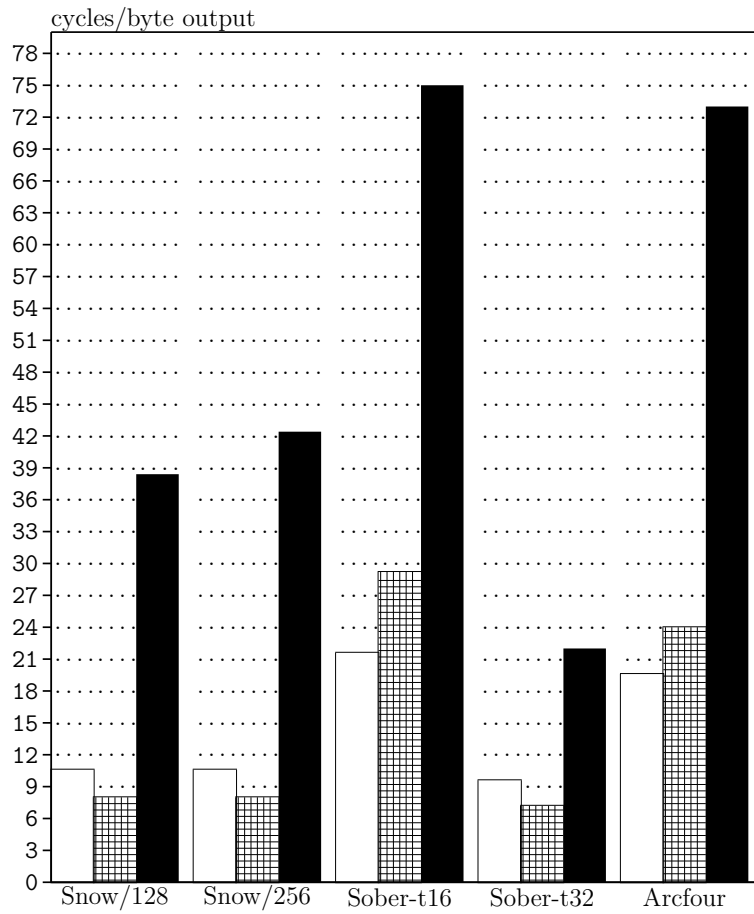


Figure 5: Stream ciphers performance (without BMGL): key stream generation

Legend:

- UltraSparc 248 MHz @ Siemens, ▨ PIII 850 MHz @ Siemens,
- PIII 850 MHz @ UCL.

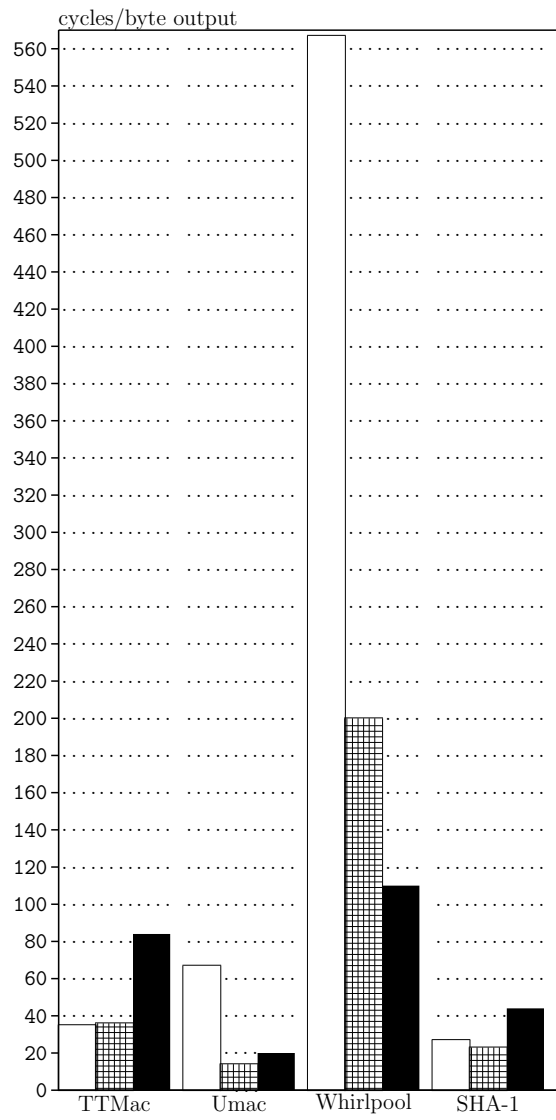


Figure 6: MACs' and hash functions performance

Legend:
 □ UltraSparc 248 MHz @ Siemens, ▤ PIII 850 MHz @ Siemens,
 ■ PIII 850 MHz @ UCL.

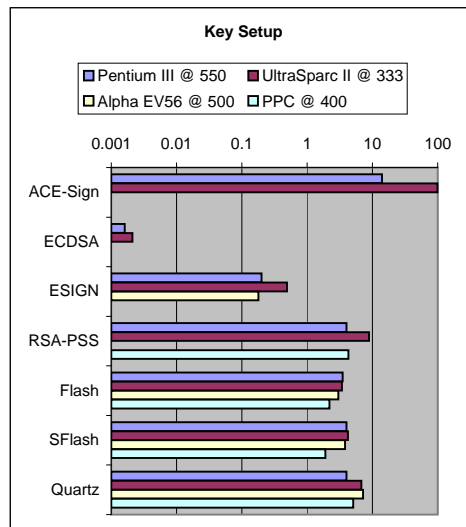


Figure 7: Digital signature : time for key setup in seconds

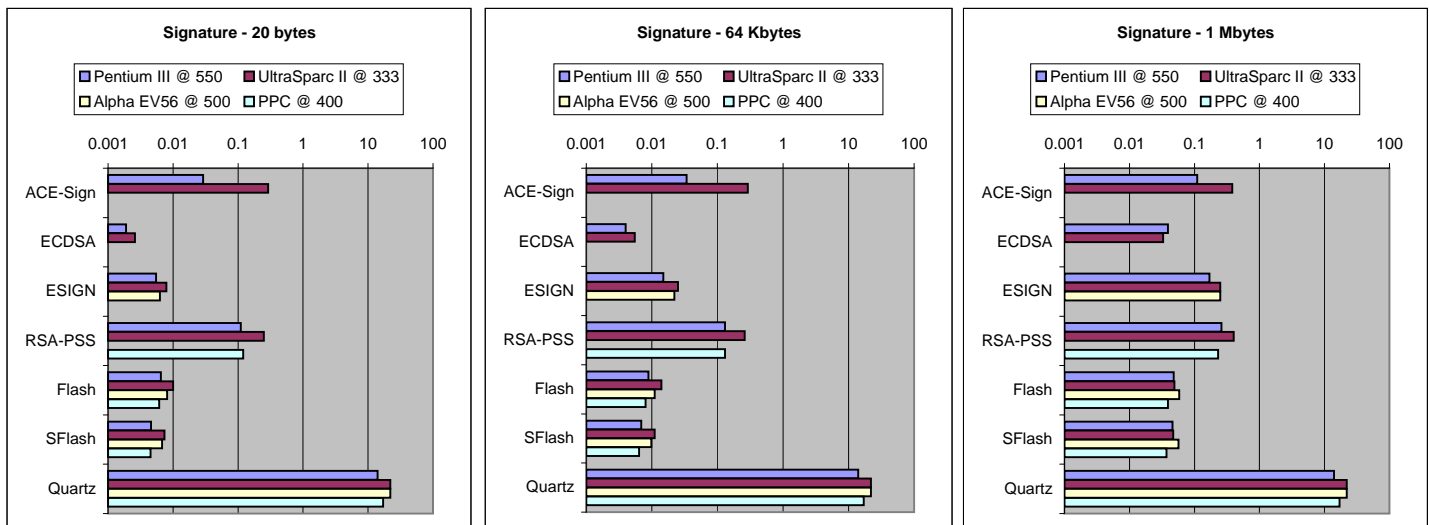


Figure 8: Digital signature : time for signature in seconds

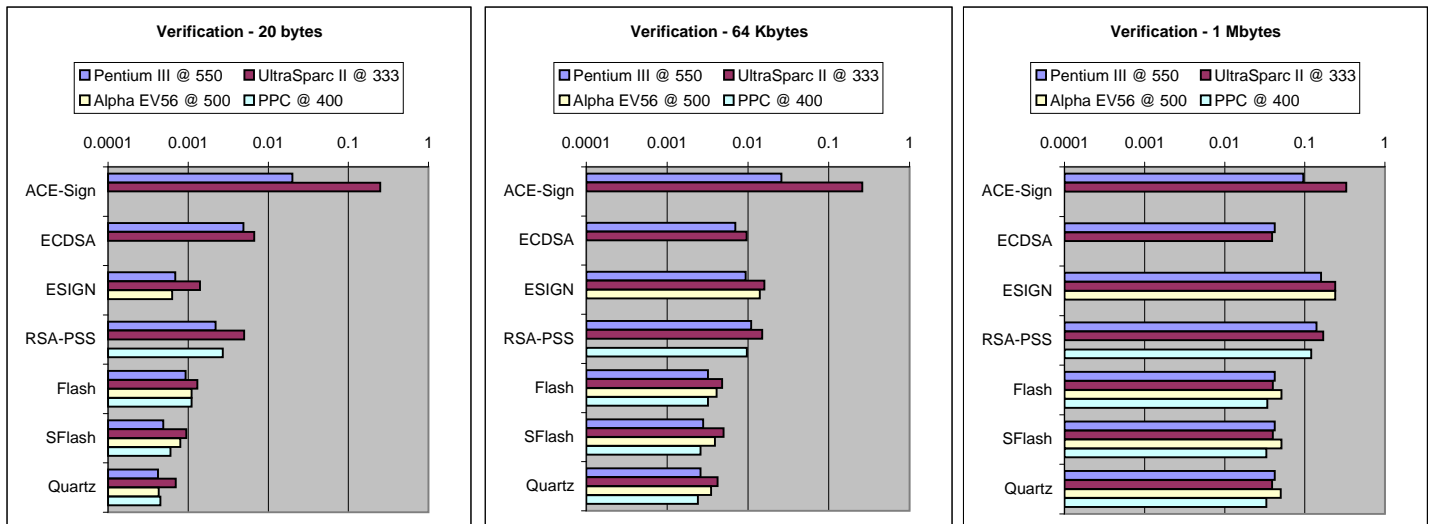


Figure 9: Digital signature : time for verification in seconds