Global Information Assurance Certification Paper

A study on hash functions for cryptography

Vladimir Omar Calderón Yaksic

Abstract

In cryptography, there are two tools needed almost for each and every system and/or protocol: random number generators and one-way functions. Their use is then vital. This work covers a detailed description of hash functions, as one-way functions, involved in different cryptographic processes. The work begins defining these functions followed by the description of their classification.

Also, some possible attacks on hash functions have been studied. Finally, the work tries to emphasize the importance of these functions giving examples of their use in cryptography for authentication, virus checking, digital signatures, timestamped documents, secure socket layer connections, etc. For all this, the work states the relevance of these functions and the need to understand them in depth.

GSEC Practical Assignment version 1.4b (amended August 29, 2002)

Contents

1 Overview

There exists two main tools used almost in each and every cryptographic system: random number generators and one-way functions. One-way functions are a huge family of functions used for confidentiality, authentication, integrity and non-repudiation. This work explores an important type of these functions which are the hash functions.

The most famous hash functions are the MD family: MD2, MD4, MD5 and SHA[12]; but we will see there are plenty of other hash functions with many different purposes. The main objectif of this work is to show a detailed description of hash functions and to state the great importance we should give to these functions, just like any other element of a cryptographic system.

The work begins describing where one-way functions are used, and that they are an essential element in cryptography. And after giving some important concepts about the classfication of cryptographic systems, it continues with a full description of the concepts and properties of one-way functions.

At that point, hash functions are ready to be introduced. The work describes the concepts and properties of these functions and also gives a first classification of hash functions as of [6].

Then the work discusses different properties and possible attacks on hash functions, considering the Shannon theorem for entropy and the Birthday Paradox in section 8.2.

Finally, various applications of hash functions are discussed showing their importance and every-day use. The work ends with a conclusion emphasizing the relevance of hash functions and their future; discussing also the fact that if quantum machines were to exist, hash functions also could be brokem (indeed, Simon's Problem and some hard-problems have been solved with quantum algorithms years ago, see [13]).

2 Why does cryptography need one-way functions?

Confidentiality is one of the 'must have' features of cryptographic systems. The idea is then to transmit a message that can not be understood by other but the receiver. Following this idea, we need functions that transform the message to make it unreadable, and, if somebody 'hears' the message, should not able to understand or interprete it. This means that we need functions that can be computed one way but not the inverse: one-way functions.

Usually, when talking about these functions, the conversation turns into subjects related to RSA, El Gamal, etc. But, an important set of one-way functions are hash functions with one-way functions' properties. Hash functions are now used in cryptography for authentication, integrity and non-repudiation especially.

So, cryptography needs these functions because of the main property of being hard to compute the inverse of the function. We have to be very careful with these matters because it involves mathematical issues which are not fully solved or known. Especially for RSA, and the cryptography systems based on hard-problems, as long as mathematicians are not sure if hard-problems can be reduced to 'easier' problems, and as long as physicians don't build quantum machines; we can be pretty sure these one-way functions will still be the key element in cryptography systems.

3 Classification of cryptographic systems

Before the description of one-way functions, we will describe some concepts that classify the different cryptographic systems. These concepts will especially be mentioned when discussing the possible attacks on the studied functions. That's why it's important to define them right from the beginning.

3.1 Unconditional security (perfect secrecy)

This category is based on the information theory published by Shannon back in the 40s. To have perfect secrecy on a cryptographic system means that if the attacker has the plaintext and the cypher text, it's of no use for him to cryptanalyze it. It means that these values are independent random variables.

To achieve this, we need a third element which will be the secret information known by the sender (the key to crypt) and the receiver (the key to decrypt). The problem, stated also in Shannon's publication, is the impracticability of this solution, because it needs the key to be as big as the message. Also, the key can not be used to exchange different messages, meaning that we should use different keys every time.

The classic example of perfect secrecy is One-Time Pad, invented in 1917 by J. Mauborgne and G. Vernam.

3.2 Provable security

Cryptographic systems are said to be provably secure if the fact to break it is as difficult as to break a generic basic hard-problem (i.e. factoring big numbers, calculate square roots in modulo a composite, calculation of discrete logarithms on a finite group, etc).

The most famous cryptographic system in this category is Rabin's cryptosystem. Notice that RSA is not proven to be provable secure, just in the random oracle model.

3.3 Computational security

Most of the cryptographic systems are in this category. Computational security means that the effort needed to break it is not available to possible attackers; or that the potential attackers don't have the sufficient amount of resources to break it.

4 One-way functions

As it has already been said in section 2, one-way functions are important. A function is a one-way function if:

$$\forall x \in X, f(x) = y$$

is easy computed, but it is virtually impossible to find a function g where $g(y) = x$

The only problem in this definition is the sentence '*is virtually impossible*'; it is important to define the impossibility of a solution. There is a whole theory behind one-way functions based on hard problems.

One-way functions are most used in asymmetric cryptography. Difficulty to find the inverse of the function depends on the size of the key used. So, that's why we feel safer using a 2048bit key than a 512bit key, because we are actually in safer ground, of course, in terms of confidentiality of the crypted messages.

Some examples of one-way functions are:

- Calculation of square roots modulo a composite:

$$f(x) = x^2 \, mod \, n \qquad \text{where } n = pq \text{ with } n, q \text{ unknown}$$

- A one-way function can be easily built based on a block cryptography system E (like DES):

$$y = f(x) = E_k(x) \oplus x \quad \text{where } k \text{ is fixed and known; } E \text{ is the function that crypts}$$

4.1 Hard problems

Complexity theory has long studied problems like: How do we know if a program will stop and give an answer? At first, this seems difficult, but it turns that it is not only a difficult, but an impossible problem. That's why we still don't have a program that tells us if a program will hang or not in a reasonable time.

Mathematicians have then classified the hard problems. There are the problems that can be solved in polynomial time (or sub-exponential[1]), it means, that the time it would take to solve them is known, and this time grows in a polynomial way as the problem becomes more complicated (these are called P problems). And there are the other problems, called NP that are problems whose solutions can be checked in polynomial time.

The question of the millenium (for which you could win a nice prize[2]) is to know whether $P = NP$ or not.

There is still another type of problems, called NP-complete. When we can reduce any problem $B \in NP$ to a problem $A \in NP$ then we can say A is NP-complete. Some NP-complete problems are the Traveling Salesman problem and the Knapsack problem [5].

So, basically, in order to solve the big question $P = NP$? We should find a NP-complete problem that can be solved in polynomial time. Then we would be able to reduce all other NP problems to our solution and solve them. A lot of work has been made to find an answer with little results. If sometime $P = NP$ should be proven, we would be able to calculate very difficult problems in polynomial time without having to develop the computing technology.

[1] But, sub-exponential doesn't mean polynomial

[2] http://www.claymath.org/Millennium_Prize_Problems/

5 Hash functions

A hash function is a function of the form:

$$h(x) = y \quad \text{where } x \in Z \text{ and } y \in Z_n$$

which has some interesting properties:

- The output of the functions is usually smaller than the input (Z_n).The input can be any size while the output is usually of a fixed size.
- To calculate the function is an easy and fast operation

The result of the hash function is also called the *digest*. The following figure shows these two properties of hash functions:

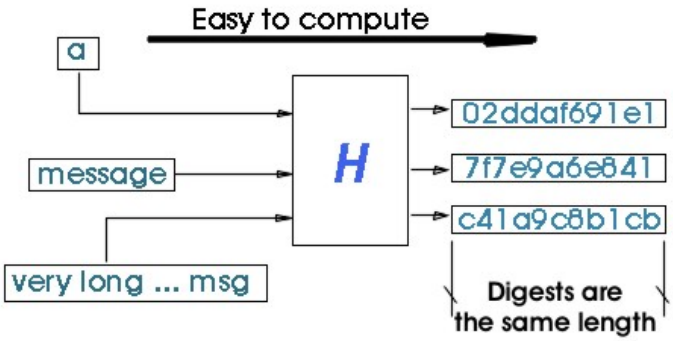

Figure 1: Input, output and properties of hash functions

These functions are used in many contexts, this work will explore the use of hash functions in general and their use in cryptography later in a more detailed way.

5.1 Cyclic redundancy checks

A CRC is an additional information given in a message that is being transmitted, Because of the redundancy added to the message, it is possible to recover some errors that might have appeared while transmission [7]. But, this does not mean that we can check integrity, because malicious agents can change the information in a way the CRC would not be affected.

5.2 Reed Solomon

These are hash functions used to detect errors while transmitting information over a channel. These are very powerful error detection codes, they are in fact an extension of cyclic codes. They are not built on binary values but a large set of symbols [8].

5.3 Properties

As we have seen, hash functions tend to reduce the input we give. This means that, potentially, there is a possibility that we find two inputs with the same output. The important thing here is to state the word '*potentially*' in a way we can use it for cryptographic issues. This means that more properties have to be given in order to have a hash function ready to use in the cryptographic world.

5.3.1 Preimage resistance

This property means that if we are in possesion of the resulting digest y, it is computationally impossible for us to find the x that was the input of the hash function $h(x) = y$.

5.3.2 Weak collision resistance

This property means that, given two values x and its image y, then it is computationally impossible to find a $x' \neq x$ where $h(x) = h(x')$. So, we do not choose the message nor the digest, and we have to search a different message with the same digest.

5.3.3 Collision resistance

It means that it is computationally impossible to find x and x' with $x \neq x'$ where $h(x) = h(x')$. Notice that now we can choose the messages, that's why this property is also called strong collision resistance.

5.4 Keyed and unkeyed hash functions

There are two main types of hash functions, the ones depending not only on the input message but also a message called the key. This key is well known for the one calculating the result of the hash function. And the others which only need the message to be *hashed*. These two types are called: keyed and unkeyed hash functions respectively, see [6].

An application of the keyed hash functions are the message authentication codes, also called MACs; which are widely used in cryptography. And, an application of unkeyed hash functions is the modification detection codes; also very used in cryptography matters. The work explores these two types in the following sections.

6 Modification Detection Codes (MDC)

These are hash functions that do not use any key to be computed. Depending on the properties they have, they can be subdivided in:

1. **One Way Hash Function**: These are functions that provide *preimage resistance* and *weak collision resistance* properties.

2. **Collision Resistant Hash Function**: These are hash functions with *weak collision resistance* and *collision resistance* properties. Notice that if the function is collision resistant then it is also *weak* collision resistant.

An important thing to notice here is that the fact to have a one-way function does not guarantee us to have a one-way hash function, this is because the latter imposes supplementary restrictions on source and image domaines.

In [6], the author gives the following classification for MDCs:

1. **Based on block cyphers**: The idea is to use an already existing block cypher system to add new properties (hashing) without affecting the performance significantly.
2. **Customized hash functions**: These are hash functions created from scratch. These will be discussed in section 9.
3. **Based on modular arithmetic**: These functions use the modulo operation to reduce the result and form a hash function.

7 Message Authentication Codes (MAC)

These are a family of hash functions dedicated to authenticate messages. The function depends on the message and a secret key. The important properties of MACs, as stated in [6] are:

- **Compression**: Like hash functions, but applied to H_k, where k is the secret key.
- **Easy to compute**: The nice property of almost all hash functions.
- **Computation resistance**: Without knowing the secret key, it is computationally impossible to get to know what $H_k(x)$ corresponds to a chosen x, even if we have got samples of the algorithm results.

The third property implies *key non-recovery* (the key k is safe). But, it is important to notice that, viceversa, this is not true.

Just like MDCs, MACs can be classified in three groups:

1. **MACs based on block cyphers**: Mostly based on DES-CBC.
2. **MACs based on MDCs**: Natural way to build MACs, but, careful analysis is needed, see examples in section 9.1 and [6].
3. **Customized MACs**: Algorithms especially created for authentication. Examples: MAA (Message Authenticator Algorithm), MD5-MAC.

A schema showing the described classification can be seen in the following figure. Notice that the figure does not include the other applications given to hash functions which can be keyed or unkeyed.

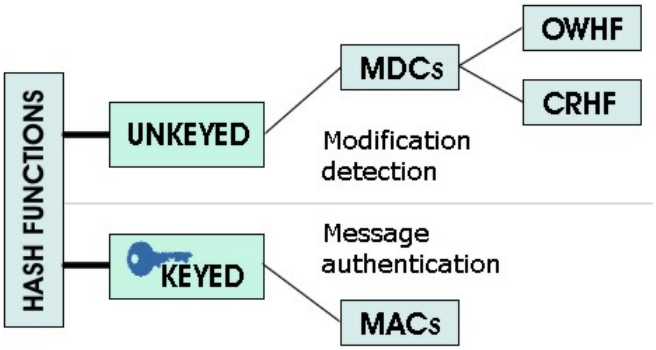

Figure 2: Classification of hash functions

8 Possible attacks on hash functions

In this section we will describe shortly how a possbile attack can be made on hash functions. First, we have to choose a property of hash functions and try to break it. This proceeding is valid because most of the properties stated have not yet been proven, or, involve the word *probable*. Here we show the way to attack the weak collision resistance property.

8.1 Attack on weak collision resistance

The problem is the following: given a digest and the message that produced it, find another message that will produce the same digest.

We will try to compute the number of times we should try, to have a good probability to find the answer to the problem. Good probability will be given by a probability of 0.5.

Let's say we have x and $h(x)$, and choose a value x_1, the possibility that we have $h(x) = h(x_i)$ is $\frac{1}{n}$. So, for m values, the possibility of not having a collision is:

$$(1-\frac{1}{n}) \cdot ... \cdot (1-\frac{1}{n}) = (1-\frac{1}{n})^m$$

$$(1-\frac{1}{n})^m = 1 - m\frac{1}{n} + \binom{m}{2}\frac{1}{n^2} - \binom{m}{3}\frac{1}{n^3} + ...$$

$$(1-\frac{1}{n})^m \approx 1 - \frac{m}{n} \quad \text{for big values of } n$$

We are looking to have a value of 0.5, so, we would have:

$$\frac{1}{2} \approx 1 - \frac{m}{n}$$

$$m \approx \frac{n}{2}$$

We have that for a digest of n bits (2^n possible values), we would need to test $2^n/2 = 2^{n-1}$ values to have a probability of 0.5 to solve the problem. That's why we have to be aware of the type of resources the attackers have to cryptanalyze our messages. If we consider 128 bits will be enough, or if it's needed 256 bits for our digest's size.

8.2 Birthday paradox

It's a simple principle that comes from a simple problem. How many people have to be in a room, to have a probability bigger than 0.5 that two persons have been born the same day? The calculation, after some mathematic work, gives us the formula:

$$m = \sqrt{2(ln2)n} \cong 1.17\sqrt{n}$$

where n is the number of results in the image (for the birthday paradox, n would be 365). And m would be the number of persons we need. This gives us $m = 22.3$. So, if we have 23 people in a room there is a good chance that we find two born the same day.

For cryptography, we can do the calculation with $m \approx \sqrt{n}$, and replace n for the number of possible results of the digest. If the digest has b bits, then $n = 2^b$. And, to have a good probability of having the same digest with two different messages, we would need to try $m = 2^{b/2}$ times.

9 Implementation of MDCs and MACs

In this section we will describe some implementations of hash functions already in every-day use. Some of them have already been broken and shouldn't be used like MD4, but are described anyway because of their historic importance.

9.1 Dedicated Modification Detection Codes

As we have seen, implementations of MDCs can be built on block cyphers, from scratch or modular aithmethic. Here it continues the description of the most famous implementations of hash functions. Other implementations would be: Matyas-Meyer-Oseas, MDC-2, MDC-4 and MASH (Modular Arithmethic Secure Hash).

9.1.1 The MD family

These are MDCs created especially for 32 bits machines. The digest is at least of 128 bits. These are MD which are computed this way:

- Depending on the size of the message, we pad some bits if necessary, and divide the message in blocs.
- A set of constants is applied to add dispersion to the message.
- A set of rounds are sequentially applied to the blocs.

- The result of the round is reinserted as input for the next round

MD2

The Message Digest Algorithm, it is described in RFC-1319 [1] in detail. The size of blocs is 16 bytes. Of course, the description given in [1] was made on 1992. This algorithm has been broken and should not be used for new applications. Maybe only for compatibility purposes. The digest is 128 bits long.

MD4 and MD5

MD4 and MD5 are described in RFC-1320[10] and RFC-1321[11] respectively. These algorithms include 4 constants as input for the first block[3] to hash. So, basically the input is always 512 bits from the block and 128 bits from constants. The result is a digest of 128 bits, which is included as input to hash the next bloc.

MD4 has 3 rounds while MD5 has 4 rounds. MD4 should not be used because it has been broken. And MD5 should not be considered for future applications.

9.1.2 SHA1

The Secure Hash Algorithm is based on MD4 and it is described on RFC-3174[4]. It works on blocs of 512 bits also, but, instead of using 4 constants, it uses 5 constants. So, if each constant was 32 bits long (128 bits in total), for SHA-1 the length of the result is 5x32 bits long (160 bits).

SHA-1 is considered to be sure for the moment.

9.1.3 RIPEMD

RIPEMD is a MDC created in the framework of the EU project RIPE. The digest is of 160 bits long (RIPEMD-160), which is considered to be sure until 2005 approximately. There exists also versions of RIPEMD to get digests of 128 bits, just a possible substitute for MD4 and MD5; 256 and 320 bits, used mostly for applications that need larger digests.

It was developed by Dobbertin, Bosselaers and Preneel. Dobbertin is responsible for finding flaws on MD4.

The following table shows different hash functions in action:

Hash function	Message	Digest
MD5	GSEC Security Essentials	abdf287462660fa9439116eb0f3b942c
SHA1	GSEC Security Essentials	b21e96c49176798b34ca13d2c3f82abbf6c26502
RIPEMD-128	GSEC Security Essentials	ced73690a0b56a6750803b1fa3d59da3
RIPEMD-160	GSEC Security Essentials	ced73690a0b56a6750803b1fa3d59da3c8670140

Table 1: Examples of messages and digests of hash functions

[3] blocks are 512 bits size

9.2 Dedicated Message Authentication Codes

MACs are widely used today, the most used are the ones created from already built MDCs, because they are computed faster than the ones based on DES-CBC. However, there exist lots of others MACs, reflecting the large family they form, for example: MAA, MD5-MAC and CRC-based MAC.

9.2.1 CBC-MAC

This is a hash function based on DES-CBC, by eliminating the intermediate cyphertexts. The key's size is 56 bits (112 using the optional part). The digest is 64 bits long (which is not considered safe anymore).

9.2.2 HMAC-MD5, HMAC-SHA1 and HMAC-RIPEMD

The HMAC family are MAC functions based on MDCs. The digest's size is then of the underlying hash function. The key used is not larger than 64 bytes (a 512 bits block). It uses two constants `ipad` and `opad` to compute the hash function in conjunction with the key and XOR operations. Basically, the HMAC function is as secure as the hash function used to compute it.

Also, this algorithm accepts the truncation of the digest down to 80 bits. The performance of this algorithm, when the message is large, tends to be as fast as the underlying hash function.

9.3 Performance

Performance measures have been made with the standard command `speed` of the command `openssl`, detailed in [9]. The execution of this command gives the number of calculations the computer can make with a determined algorithm. The command performs the calculations for 3 seconds[4].

All measures are in 1000s of bytes per second processed. The following are two tables showing the performance of different hash functions in two different machines. All timings are compared with the one performed by MD2.

HF/block size	64 bytes		256 bytes		1024 bytes		8192 bytes	
MD2	1754.39k	1.0	2464.39k	1.0	2725.55k	1.0	2818.05k	1.0
MD4	26959.42k	15.3	77394.81k	31.4	145022.45k	53.2	193320.28k	68.6
MD5	22334.63k	12.7	66966.36k	27.2	131811.67k	48.4	184606.72k	65.5
HMAC(MD5)	14093.50k	8.0	46245.97k	18.8	108043.95k	39.6	177919.32k	63.1
SHA1	20952.26k	11.9	51399.42k	20.9	82104.32k	30.1	99304.03k	35.2
RIPEMD160	17604.76k	10.0	39678.38k	16.1	57750.87k	21.2	66701.99k	23.7

Table 2: Performance on AMD Athlon 1050Mhz, 256Mb RAM running Linux RH9

[4] In the case of hash functions only, for cypher algorithms it uses 10 or 15 seconds

HF/block size	64 bytes		256 bytes		1024 bytes		8192 bytes	
MD2	2945.83k	1.0	4026.75k	1.0	4389.94k	1.0	4598.09k	1.0
MD4	36987.28k	12.6	98147.80k	24.4	160098.78k	36.5	193180.01k	42.0
MD5	38200.07k	13.0	104595.93k	26.0	184259.52k	42.0	233499.77k	50.8
HMAC(MD5)	21001.75k	7.1	68866.59k	17.1	151759.20k	34.6	227412.65k	49.5
SHA1	22561.54k	7.7	52878.08k	13.1	81529.17k	18.6	97498.45k	21.2
RIPEMD160	22330.28k	7.6	45506.56k	11.3	64100.69k	14.6	72015.87k	15.7

Table 3: Performance on Pentium IV 2000Mhz, 512Mb RAM running Linux RH9

We can observe here that MD2 is a very slow algorithm, even if it belongs to the MD family, it has a very different algorithm for its computation, see section 9.1.1.

An important observation is that MD5 is faster (in table 3 only) than the obsolete (and already broken) MD4. Also, it turns that the difference of speed is bigger as the algorithms work on bigger blocks. So, normally, MD5 is much faster than MD4 when computing the digest of big messages (think of files for example). This can also be because of the MD4 and MD5 implementations in OpenSSL.

An interesting observation is that, while all algorithms get a better performance in the fast machine[5], SHA-1 gets similar performance. This means that if we get to have much better machines, RIPEMD160 (goes faster on faster machines) would get to be as fast as SHA-1, because it seems its performance does not depend significantly on the speed of the machine.

10 Other applications of hash functions

In the following sections there are examples of hash functions executing as one of the elements for cryptographic systems. This list is non-exhaustive but it lets us perceive the relevance and wide usage we can and should give to hash functions.

10.1 The UNIX example

UNIX keeps the passwords in a file that can be accessed by anyone. The information (the password), is not the password itself but the result of a hash function. The problem is that, we will always get the same result for the same input (*brute force* attacks can easily be accomplished with a password dictionary file).

For this reason, UNIX adds a random number, called *salt*, to the password, before the computation of the hash function, see [14] for details on specification. These are called *randomized* hash functions because they depend on a random number, the *salt*.

[5]The one used for table 3

10.2 A word about One-Time Pad

The one-time pad is the perfect secrecy system for *cyphertext only* attacks. It means that it is of no use for the cryptanalyst to have the cyphertexts. The problem is that it uses the Shannon theorem of information, which states that, if a secret key is used, it needs to be at least the same size than the message itself. Also, the key can only be used once. Normally, the cyphertext is obtained with:

$$E_k(x) = y = x \oplus k$$

which is a hash function based on k (cyphertext is y and message is x). This system is interesting because of its property of perfect secrecy, but it's not suitable for real applications because of the potentially large sizes of keys to manage.

10.3 Digital signatures

Digital signatures are a huge subject in cryptography. These are used for signing a document (of course), but with the good properties that the signer can tell when something has not been signed by him AND that he can not deny his signature in a signed document. A performance issue is the fact of signing documents of different sizes: for little documents it will be faster than for large documents. As long as we would like to be able to sign all documents in the same time, what is usually done is to hash the message first with a hash function[6].

10.3.1 Timestamps

Timestamped documents are at the heart of the legal use of electronic documents. To know when a document has been signed for example, the timestamp also is included in the message to be hashed [3].

10.4 Virus checking

To know if a file has been infected with a virus means to know if it has been, even in a slightly way, modified. That's why we can use the digest of the unmodified and safe file to compare it to the digest of the possible infected file. If it turns that both digest are equal then we can say that the file has not been modified, then that it is virus-free.

This mechanism is used also by developers and software companies when distributing their software. It is simple and it is robust (depending on the hash function used).

10.5 Secure Socket Layer connections

Secure Socket Layer (SSL) provides server authentication to clients. It's the facto standard for communication on the Internet. During the SSL handshake (detailed in [2]), once the parts have negotiated the protocol, these have to select also the hash methods to use for

[6] In formal, we say the space of the messages is reduced with a hash function

authentication. The client can choose the type of hash function to use, usually MD5 or SHA-1 (which are MACs).

11 Conclusion

As we have seen, hash functions form a wide family of functions, although their use has many years with CRC, etc; these functions play a major role in cryptography. Mathematicians have showed us that we can rely on functions with properties involving the words 'probably' and 'very difficult', like the properties involving collision resistance and weak collision resistance.

Cryptography has then been able to use the properties of hash functions and specially MDCs and MACs to create the widely used MD5, SHA, RIPEMD-160, CBC-MAC, MD5-MAC. And also, their use is extended, to authentication, virus checking, digital signatures, etc.

Performance tests of the most famous hash functions used in cryptography have been made with the `speed` option of the OpenSSL command. We can now process 233Mb using MD5 in blocks of 8kb. But, we have seen that the use of a better machine does not increase that much the performance (compare tables 2 and 3), and in some cases it's even similar (SHA-1).

In this work, hopefully, we have stated the importance of hash functions in cryptography, usually left aside because of the famous RSA, DES, etc. Among the many uses of hash functions, maybe the most important would be digital signatures, because of the economic and social impact (in many countries, digitally signed documents are already taken as legal documents). A good thing to notice is that we have not seen many new hash functions proposed the last years, this is a good thing, because it means that old hash functions are still robust enough for our purposes.

For the future, maybe cryptography is the most anxious field of science to see a physical quantum machine. Many of the algorithms we use should be re-structured simply because, with quantum machines, there already exist algorithms that can solve *hard problems*. Quantum machines also promise to have secure channels for key exchange of unlimited size.

References

[1] Kaliski B. RFC-1319 The MD2 Message-Digest Algorithm, 1992. http://www.ietf.org/rfc/rfc1319.txt.

[2] Johnson Brad and Gossels Jonathan. The ssl handshake, 2001. http://www.systemexperts.com/tutors/The_SSL_Handshake_V1.5f.pdf.

[3] Ahto Buldas, Peeter Laud, Helger Lipmaa, and Jan Villemson. Time-stamping with Binary Linking Schemes. In Hugo Krawczyk, editor, *Advances on Cryptology — CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 486–501, Santa Barbara, USA, August 1998. Springer-Verlag.

[4] Jones Eastlake. US Secure Hash Algorithm 1 (SHA1), 2001. http://www.ietf.org/rfc/rfc3174.txt.

[5] RSA Laboratories. Rsa laboratories' frequently asked questions about today's cryptography, version 4.1, 2000. http://www.rsasecurity.com/rsalabs/faq/2-3-1.html.

[6] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., 1996.

[7] National Institute of Standards and Technology. Cyclic redundancy check, 2002. http://www.nist.gov/dads/HTML/cyclicRedundancyCheck.html.

[8] University of Tours-France. Channel coding and error detection, 2001. http://www.rfai.li.univ-tours.fr/ramel/fr/codage_canal.pdf.

[9] OpenSSL Project. Speed standard command, 2002. http://www.openssl.org/docs/apps/speed.html.

[10] Rivest Ron. RFC-1320 The MD4 Message-Digest Algorithm, 1992. http://www.ietf.org/rfc/rfc1320.txt.

[11] Rivest Ron. RFC-1321 The MD5 Message-Digest Algorithm, 1992. http://www.ietf.org/rfc/rfc1321.txt.

[12] SSH Communications Security. Cryptography a-z - cryptographic hash functions, 2003. http://www.ssh.com/support/cryptography/introduction/hash.html.

[13] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *IEEE Symposium on Foundations of Computer Science*, pages 124–134, 1994.

[14] Stallings William. *Network and internetwork security: principles and practice*. Prentice-Hall, Inc., 1995.