Comparison of Digital Signature Algorithm and Authentication Schemes for H.264 Compressed Video

Ramzi Haddaji
Electrical department
National Engineering School of Monastir, University of Monastir, Tunisia
Laboratory of Electronic and Microelectronic, University of Monastir, Tunisia
Monastir, Tunisia

Raouf Ouni
Mathematical department
National Engineering School of Monastir, University of Monastir, Tunisia
Faculty of sciences, Tunis El-Manar University
Tunis, Tunisia

Samia Bouaziz
Electrical department
National Engineering School of Monastir, University of Monastir, Tunisia
Laboratory of Electronic and Microelectronic, University of Monastir, Tunisia

Abdellatif Mtibaa
Electrical department
National Engineering School of Monastir, University of Monastir, Tunisia
Laboratory of Electronic and Microelectronic, University of Monastir, Tunisia
Monastir, Tunisia

***Abstract*—In this paper we present the advantages of the elliptic curve cryptography for the implementations of the electronic signature algorithms "elliptic curve digital signature algorithm, ECDSA", compared with "the digital signature algorithm, DSA", for the signing and authentication of H.264 compressed videos. Also, we compared the strength and add-time of these algorithms on a database containing several videos sequences.**

***Keywords*—Elliptic curve cryptography; H.264; DSA (Digital signature algorithm); ECDSA (Elliptic Curve Digital Signature Algorithm); Implementation**

I. Introduction

The media industry has witnessed a phenomenal and unprecedented explosion in the recent decade. Communication, technology and media have transcended all boundaries, and the entire global community seems to have been brought together into one unified whole. Therefore in this era of evolving communication, different types of business related to media such as IPTV, Voice IP and videoconferencing, have also found solid grounds, these must be secured to protect privacy and to prevent from hackers [1].

Certain implementation security aspects of video are authentication, data integrity and confidentiality.

Authentication is the act of verifying a claim of identity.

Data integrity in information security means maintaining and assuring the accuracy and completeness of data over its entire life-cycle. This means that data cannot be modified in an unauthorized or undetected manner.

Confidentiality is the property, that information is not made available or disclosed to unauthorized individuals, entities, or processes [2].

The multimedia information including video data has some special characteristics like high capacity, redundancy and high correlation among pixels which leads us to choose the type of video encoding on which we will work. This brings us to use H.264 given the advantage that provides this type as size standpoint and video quality [3].

In this paper we focus our work in the authentication aspect which is verified using the signature algorithms. We compare the implementation of the most known two signature algorithms DSA, digital signature algorithm, and ECDSA, elliptic curve digital signature algorithm [4]-[5].

As this type of data requires memory space, the process of electronic signature is not used directly on the video but rather on what we call the hash of this one. A hash function known also a one-way function is a cryptographic tool which produce a fixed size fingerprint regardless of the size of the input [4].

The remaining of this paper is organized as follows. In Sect. 2, we recall properties and give some example of hash functions. In section 3 and 4, we describe the signature algorithms DSA and ECDSA. H.264 encoding is briefly described in section 5. Performance evaluation and comparative results of our implementation are given in detail in Sect. 6. Finally, some conclusions are made.

II. Hash Function

Cryptographic hash function plays an important role in the world of cryptography. They are employed in many applications for digital signatures, data integrity, message authentication, and key derivation. Secure Hash Algorithm (SHA-1) specifies which generates condensed of message called message digest. Hash functions takes a message of variable length as input and produce a fixed length string as output referred to as hash code or simply hash of the input message. The basic idea of cryptographic hash function is use

of hash code as compact and non ambiguous image of message from which latter cannot be deduced. The term non ambiguous refers to the fact that the hash code can be as it was uniquely identifiable with the source message. For this reason it is also called as digital finger print of the message. The hash functions [4]- [6] are classified into keyed and unkeyed hash function; the keyed hash functions are used in the Message Authentication Code (MAC) whose specification are dictates two distinct inputs a message and a secret key. The unkeyed hash function have there categories hash function based on block ciphers, modular arithmetic and customized hash function. The hash functions have one-way property; given *n* and an input *M*, computing *H (M) =n*, must be easy and given *n*, it is hard to compute *M* such that *H(M)=n*. The type of attacks are the collision attack (find two message *M=M'* with *H(M)=H(M')*, the preimage attacks (given a random value *y* , find a message *M* with *H(M)=y*) and the second preimage attack (given a message *M* , find a message *M=M'* with *H(M)=H(M')* [7].

The most common used family of hash functions are SHA and MD families. The SHA-1 is required for use with the digital signature algorithm as specified in Digital Signature Standard (DSS) and whenever a secure hash algorithm is required. Both the transmitter and intended receiver of a message in computing and verifying a digital signature uses the SHA-1 [7]-[8]. It is necessary to ensure the security of digital signature algorithm, when a message of any length is input, the SHA produces m bits output called Message Digest (MD). The MD is then used in the digital signature algorithm. Signing the MD using the private key rather than the message often improved efficiency of the process because the MD is usually much smaller than the message. The same MD should be obtained by the verifier using the user public key when the received version of the message is used as input to SHA.

In the recent years much progress has been made in the design of practical one-way hashing algorithms which is efficient for implementation by both hardware and software. Noteworthy work includes the MD family which consist of three algorithms MD2, MD4, MD5 [9]-[10]-[11]-[12]. In our work we are interested of MD5 [11]-[12], which is the most adapted hash function in the authentication and signature of video data. Let begin by a brief description of MD5 which is developed by Ron Rivest, a much more detailed description can be found in RFC 1321 [11]. MD5 works by first padding the message until it is a multiple of 512 bits long. Padding is done as follows:

1) Append a '1' bit to the message.
2) Append '0' bits until the message is 64 bits shorter than a multiple of 512 bits.
3) Append a 64-bit representation of the message's original length.

The state of MD5 is kept in four 32-bit words, A, B, C, and D, all of which are initialized to magic constant values. MD5 processes the message in 512-bit blocks. As we process the ith block of message, we update Ai-1, Bi-1, Ci-1, and Di-1 to Ai, Bi, Ci, and Di. The output of MD5, a 128 bit value, is the final state of A, B, C, and D concatenated. For each block of message, we have four rounds of updates. Each round updates one of the four 32-bit words A, B, C, or D four times. (For a total of sixteen updates per block of message.) Initially on each round, Ai←Ai1, Bi← Bi1, etc. Each of the updates is something similar to Ai ←Bi+(Ai+F (Bi; Ci; Di)+Mi+Ti <<< s), where F is a function, Mi is the ith block of the message, and Ti and s are magic constants. (The symbol <<< means \rotate left".) At the end of each round, we finish by updating all of the values one last time, namely: Ai ←Ai + Ai-1, Bi← Bi + Bi-1, etc.

The maximum security depends on the length of message digest generated by the hash functions which is limited by the size of input to the algorithm. It also shows how the modification is done with satisfying the properties like compression, preimage resistance, and collision resistance. The simulation results show that proposed scheme provides better security than the existing one, in figure 1 we illustrate the diagram of a general hash function.

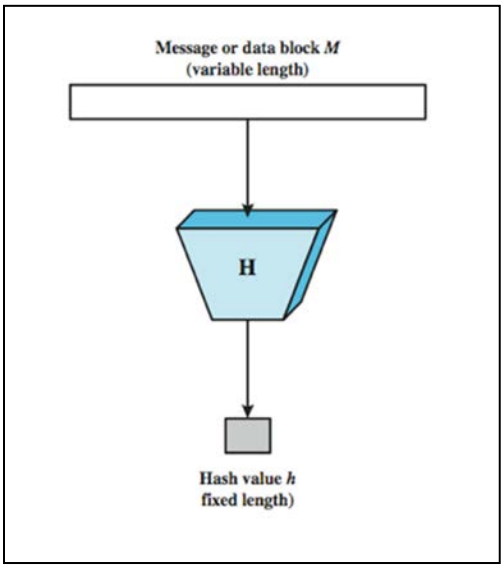

Fig. 1. Diagram of the hash function

III. Digital Signature Algorithm

Digital signature is a mechanism by which a message is authenticated which means proving that a message is effectively coming from a given sender, much like a physical signature on a paper document. For instance, let suppose that Alice wants to digitally sign a message to Bob. To do so, she uses her private-key to encrypt the message; she then sends the message along with her public-key (typically, the public key is attached to the signed message). Since Alice's public-key is the only key that can decrypt that message, a successful decryption constitutes a Digital Signature Verification, and meaning that there is no doubt that it is Alice's private key that encrypted the message [13].

The DSA was proposed in August 1991 by the U.S. National Institute of Standards and Technology (NIST) and became a U.S. Federal Information Processing Standard (FIPS 186) in 1993. The FIPS 186 standard is also referred to as the Digital Signature Standard (DSS). The DSA was the first digital signature scheme accepted as legally binding by a government. The algorithm is a variant of the ElGamal

signature scheme. It exploits small subgroups in $\mathbb{Z}_p^*$ in order to decrease the size of signatures. The key generation, signature generation, and signature verification procedures for DSA are given next.

DSA key generation. Each entity A does the following:

1. Select a prime q such that 2159 *<q< 2160*.
2. a 1024-*bit* prime number p with the property that $q \mid p-1$. (The DSS mandates that p be a prime such that *2^{511+64t} < p < 2^{512+64t}* where $0 \leq t \leq 8$ then I is a I prime.)
3. Select an element $h \in \mathbb{Z}_p^*$ and compute $g = h^{p-1} \mid q \bmod p$ repeat until $g \geq 1$. (g is a generator of the unique cyclic group of order q $\in \mathbb{Z}_p^*$)
4. Select a random integer x in the interval [1; q-1].
5. Compute $y = g^x \bmod p$
6. The public key is *(p; q; g; y)*; And the private key is x.

DSA signature generation. To sign a message

m, *A* does the following:

1. Select a random integer k in the interval [1; q-1].
2. Compute $r = (g^k \bmod p) \bmod q$
3. Compute $k^{-1} \bmod q$
4. Compute $s = k^{-1}\{h(m) + xr\} \bmod q$ where h is the Hashed message.
5. If s = *0* then go to step 1. (If s = *0*, then $s^{-1} \bmod q$ does not exist; s^{-1} is required in step 3 of signature verification.)
6. The signature for the message m is the pair of integers *(r; s)*.

DSA signature verification. To verify A's signature

(r; s) on *m* , B should:

1. Obtain an authentic copy of A's public key *(p; q; g; y)*.
2. Verify that *r* and *s* are integers in the interval *[1; q-1]*.
3. Compute $s^{-1} \bmod q$ and *h (m)*.
4. Compute $u_1 = h(m)w \bmod q$ and $u_2 = rw \bmod q$
5. Compute $v = (g^{\{u_1\}} g^{\{u_2\}} \bmod p) \bmod q$.
6. Accept the signature if and only if *v = r*.

Since *r* and *s* are each integers less than *q*, DSA signatures are *320 bits* in size. The security of the DSA relies on two distinct but related discrete logarithm problems. One is the discrete logarithm problem in $\mathbb{Z}_p^*$ where the number field sieve algorithm [4] applies; this algorithm has a sub exponential running time. More precisely, the running time of the algorithm is $O(\exp(c + o(1))(\ln p)^{1/3}(\ln(\ln p))^{2/3})$, where $c \cong 1{,}923$, and *ln (n)* denotes the natural logarithm function. If p is a *1024-bit* prime, then the precedent expression represents an infeasible amount of computation; thus the DSA is currently not vulnerable to this attack. The second discrete logarithm problem works to the base *g* given *p, q, g,* and *y*, find *x* such that $y \equiv gx \ (mod\ p)$. For large *p* (e.g., *1024-bits*), the best algorithm known for this problem is the Pollard rho-method [4]-[6], and takes about $\sqrt{\pi\, q/2}$ (2) steps. If $q \approx 2^{160}$, then the expression (2) represents an infeasible amount of computation; thus the DSA is not vulnerable to this attack. How- ever, note that there are two primary security parameters for DSA, the size of p and the size of *q*. Increasing one without a corresponding increase in the other will not result in an effective increase in security. In figure 2, we illustrate the digital signature process.

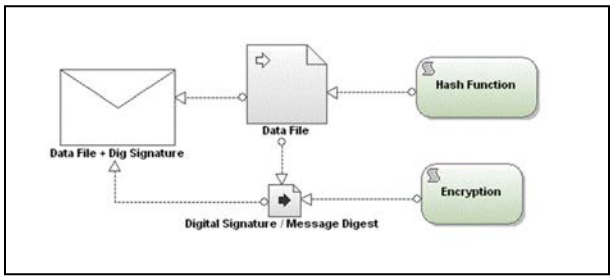

Fig. 2. Digital signature process

IV. Elliptic Curve Digital Signature Algorithm

A. Elliptic Curve Cryptography

The theory of elliptic curves is deep and an enormous amount of research has been done on elliptic curve cryptography during the past twenty years or so. Therefore, it is impossible to present an extensive review of the field here and only subjects which are the most relevant are discussed in the following. Interested readers are referred to [14], for example, for further information.

All elliptic curve cryptosystems are based on an operation called elliptic curve point multiplication which is defined as *Q = kP* where *k* is an integer and *Q* and *P* are points on an elliptic curve. A point is represented with two coordinates as *(x, y)*. The reason why elliptic curve point multiplication is used in cryptosystem is that it is relatively easy to compute but its inverse operation called elliptic curve discrete logarithm problem, that is finding *k* if *P* and *Q* are known, is considered impossible to solve with present computational resources if parameters are chosen correctly. Thus, elliptic curve discrete logarithm problem can be compared, for example, to integer factorization problem which is used in the popular RSA cryptosystems [4]. There is, however, a notable difference because sub-exponential algorithms for solving elliptic curve discrete logarithm problem are not known and, therefore, key lengths can be shorter than in RSA. Elliptic curve point multiplication is computed by using two principal operations; namely, point addition and point doubling. Point addition is the operation $P_3 = P_1 + P_2$ where P_i are points on an elliptic curve. Point doubling is the operation $P_3 = 2P_1$. In this design, point multiplication is computed with the so-called Montgomery's ladder. Elliptic curves used in cryptosystems are defined over finite fields denoted by

GF(q) where *q* is the number of elements in the field. It is commonly preferred especially in hardware implementations to use binary field $GF(2^m)$s where an element of the field is presented with *m bits*. In this design, the field $GF(2^{163})$ is used and it is constructed by using normal basis. Arithmetic operations are computed as follows:

- Addition *a + b* is computed with a bitwise exclusive-or (*XOR*).

- Multiplication $a \times b$ is computed as presented by Wang et al. in [15]. This multiplier structure is referred to as Massey-Omura multiplier in the paper [16].
- Squaring a^2 is simply a cyclical rotation of the bit vector representing *a*.
- Finding an inverse element a^{-1} such that $a^{-1}a = 1$ is performed as suggested by Itoh and Tsujii in [17] and it is called henceforth Itoh-Tsujii inversion. One Itoh-Tsujii inversion requires 9 multiplications and 162 squarings if *m = 163*.

Point representation with two coordinates as *(x, y)* is referred to as the affine coordinate representation. When points are represented in affine coordinates, both point addition and point doubling require inversion in $GF(2^m)$. Inversion is by far the most expensive operation and, thus, it is advantageous to trade inversions for multiplications. This can be done by representing points with projective coordinates as *(X, Y, Z);* that is, with three coordinates. Mappings between these two representations are performed as *(x, y, 1)* and *(X/Z, Y /Z)*. As can be seen, the mapping from affine to projective coordinates does not require any operations but the mapping from projective to affine coordinates requires two multiplications and one inversion. Using projective coordinates is very advantageous because point additions and point doublings can be performed without inversions and the total number of inversions in elliptic curve point multiplication is therefore one. A very efficient algorithm for computing (1) on elliptic curves over $GF(2^m)$ was presented in [18] by Julio Lopez and Ricardo Dahab. The authors of [18] shows that it suffices to consider only the x-coordinate and the y-coordinate can be recovered in the end [18]. This leads to a very efficient algorithm with projective coordinates. Point addition $(X_3, Z_3) = (X_1, Z_1) + (X_2, Z_2)$ can be computed as follows:

$$Z_3 = (X_1Z_2 + X_2Z_1)^2, X_3 = xZ_3 + X_1Z_2X_2Z_1 \quad (2)$$

where *x* is the *x*-coordinate of the base point P. The cost of point addition is four multiplications, two additions and one squaring. Point doubling $(X_3, Z_3) = 2(X_1, Z_1)$ is even simpler

$$X_3 = X_1^4 + a_6Z_1^4,\ Z_3 = X_1^2Z_1^2 \quad (3)$$

where a_6 is a fixed curve parameter. Thus, point doubling costs two multiplications, four squarings and one addition. The *y*-coordinate is recovered in the end by

computing $x_1 = X_1/Z_1$ and $x_2 = X_2/Z_2$ and then by using the formula:

$$y_1 = \frac{(x_1 + x)((x_1 + x)(x_2 + x) + x^2 + y)}{x} + y \quad (4)$$

where *(x, y)* is the base point *P*. This can be computed with one inversion, ten multiplications, six additions and one squaring.

B. ECDSA

ECDSA is a standard of ANSI, IEEE, and NIST, among others. The following description is based on Johnson and others' presentation in [19]. The algorithm operates so that first the user, who is commonly called Alice or A for short, generates two keys, private and public, by performing a key pair generation procedure. Then, she publishes her public key. Alice signs a message by performing a signature generation procedure after which she sends both the message and the attached signature to the receiver who is called Bob, or B for short. Bob can verify the signature on the message by first getting Alice's public key and then by performing the signature verification procedure. Key pair generation, signature generation and signature verification are consider in the following sections.

Key Pair Generation.

Private and public key for an identity A is generated as follows:

$$d \in_R [1, n - 1]\ Q = dG \quad (5)$$

Where $d \in_R [1, n - 1]$ means that *d* is an integer selected at random from the interval $[1, n - 1]$. The integer *d* is A's private key and Q is A's public key. The computation of (5) requires generation of one random integer and computation of one elliptic curve point multiplication.

Signature Generation.

In order to generate a signature for a message *M* the identity A computes

$$k \in_R [1, n - 1]\ r = [kG]_x \pmod n$$
$$e = H(M)$$
$$s = k^{-1}(e + dr) \pmod n \quad (6)$$

A's signature on *M* is *(r, s)*. The notation $[kG]_x$ denotes the *x*-coordinate of the result point of kG. Notice that A uses his/her private key *d* in the signature generation. Thus, other identities cannot produce the same signature without knowing *d*. Signing a message requires generation of one random integer, computation of one elliptic curve point multiplication and one hashing. In addition, modular inversion, addition and multiplication are required.

Signature Verification.

Identity B verifies A's signature *(r, s)* on the message *M* by computing

$$e = H(M)$$
$$w = s^{-1} \pmod n$$
$$u_1 = ew \pmod n$$
$$u_2 = rw \pmod n$$
$$v = [u_1G + u_2Q]_x \pmod n \quad (7)$$

where Q is A's public key and thus known by B. If $v = r$, B accepts the signature, otherwise (s)he rejects it. Verification requires one hashing and two elliptic curve point multiplications which are combined with a single elliptic curve point addition. Modular inversion and two multiplications are needed, as well.

V. H.264/AVC Compressed Video

An H.264 video encoder is mainly comprised of motion estimation, motion compensation, intra frame prediction, discrete cosine transformation, quantization and entropy encoding [20]. Figure 3 shown block diagram of H.264 Encoder. The brief overview of H.264 block is as follows. Encoder has intra prediction mode, which removes spatial

redundancy from the frame. The feedback path of the decoder module is an access point, which is used to decode intra predicted frame correctly. It works on different intra mode to remove spatial redundant data from the reference frame.

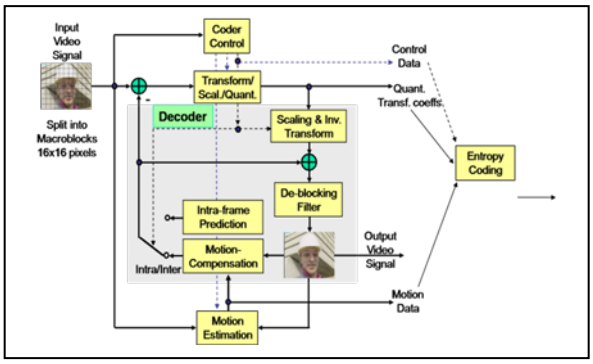

Fig. 3. Diagram block of H.264 encoder

Depending on the H.264 profile, different types of frames such as I-frames, P-frames and B-frames, may be used by an encoder. An I-frame, or intra frame, is a self-contained frame that can be independently decoded without any reference to other images. The first image in a video sequence is always an I-frame. I-frames are needed as starting points for new viewers or resynchronization points if the transmitted bit stream is damaged. I-frames can be used to implement fast-forward, rewind and other random access functions. An encoder will automatically insert I-frames at regular intervals or on demand if new clients are expected to join in viewing a stream. The drawback of I-frames is that they consume much more bits, but on the other hand, they do not generate many artifacts. A P-frame, which stands for predictive inter frame, makes references to parts of earlier I and/or P frame(s) to code the frame. P-frames usually require fewer bits than I-frames, but a drawback is that they are very sensitive to transmission errors because of the complex dependency on earlier P and I reference frames. A B-frame, or bi-predictive inter frame, is a frame that makes references to both an earlier reference frame and a future frame.

In the figure.4, we give a sequence example of I, B and P frames.

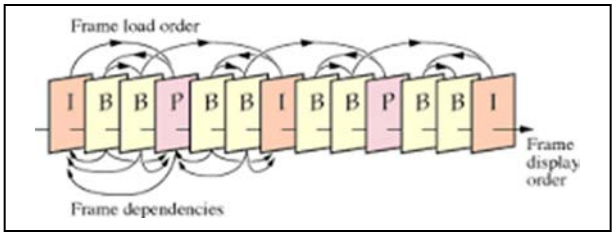

Fig. 4. Sequence of I, B and P frames

An H.264 encoder generated up to 50% fewer bits per second for a sample video sequence than an MPEG-4 encoder with motion compensation. In figure 5 the H.264 encoder was at least three times more efficient than an MPEG-4 encoder with no motion compensation and at least six times more efficient than Motion JPEG.

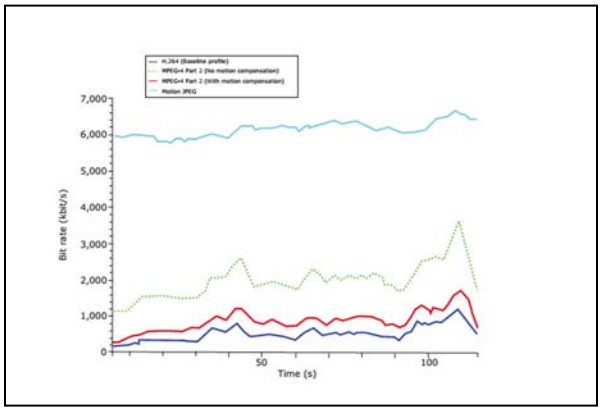

Fig. 5. Comparison of Bit rate of different encoders

VI. Experimentals Results

In this section we give the results of the comparison we do between DSA and ECDSA used for the signing of large number of H.264 video. We use MATLAB on a 64-bit Intel Core I7-4500U CPU 2.4 GHz, 6 G RAM machine to implement DSA and ECDSA signatures generation scheme and to test their performances. Our results are given below. Experimental results are given in this section to demonstrate the benefit of using the ECDSA based on the elliptic curve cryptography. These benefits can being seen in the gain of the time and the smallest size of the key in the implementation. We used DSA and ECDSA to sign the hashing output of some H.264 videos. Here below we give some results of our experimental results.

A. Comparison of the speed of hash function

We start by selecting the appropriate hash function to use for the videos signing. For this purpose we have compared the speed of the implementation of the most commonly used hash functions. There are several techniques in which are based the construction of hash functions. For example include the SHA-1 function. The choice of the hash function for the signature depends on the nature of the document to be signed. In the figure.6 we compare the speed of the main existant hash functions. For the rest and for signing the videos with real time constraint we used the MD5 function view the advantage that provides this function with respect to speed.

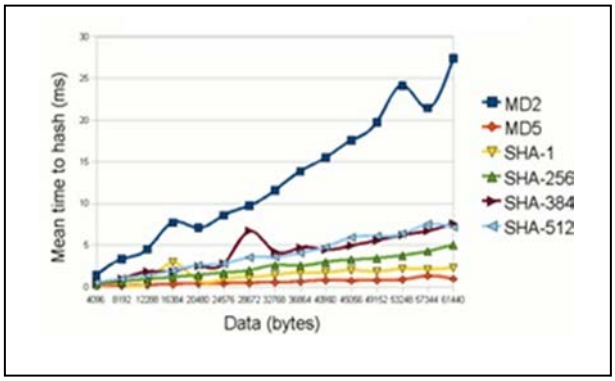

Fig. 6. Speed of secure hash functions

B. DSA vs ECDSA

In the table I below given by NIST, we give a comparison of the key size between DSA and ECDSA for a given level of security. We can see that the key size is very small in the case of ECDSA over DSA which can be an advantage in applications where we have real-time and memory constraints.

TABLE I. COMPARISON OF THE KEY SIZE

Security (bit)	DSA –Size of the key	ECDSA-Size of the key
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	512

In the figure 7 we illustrate the time to break DSA and ECDSA depending on the size of the key.

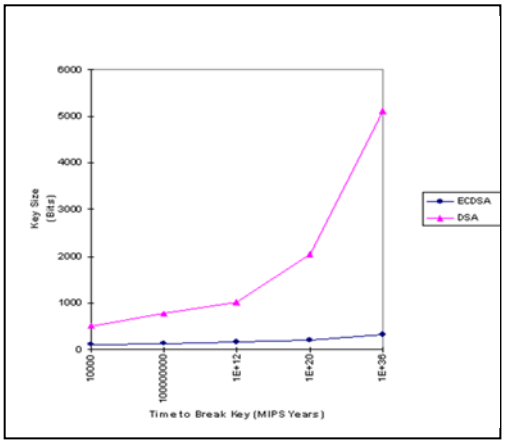

Fig. 7. Comparison of time to break DSA and ECDSA

We also compare the speed and space that requires the hardware implementation of both electronic signing protocols in figure 8 below we can see the advantage of using ECDSA. If we implement our algorithms using VLSI cores, whether in relation to the space used in number of gates or speed, ECDSA differs greatly from its rival DSA.

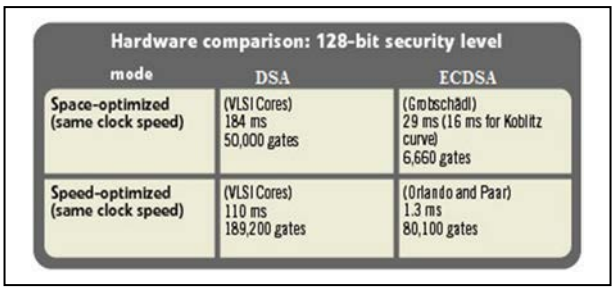

Fig. 8. Hardware comparison of space and time of DSA and ECDSA

Also in the figures 9, 10, 11 and 12, below we show the difference in the shape of histograms in the case of two H.264 videos using in the first two figure fig.9 and fig.10 the DSA protocol and the other two figures fig.11 and fig.12 the protocol ECDSA. We can notice the difference in scope between the two cases of the presented histograms which is due to reduced key size.

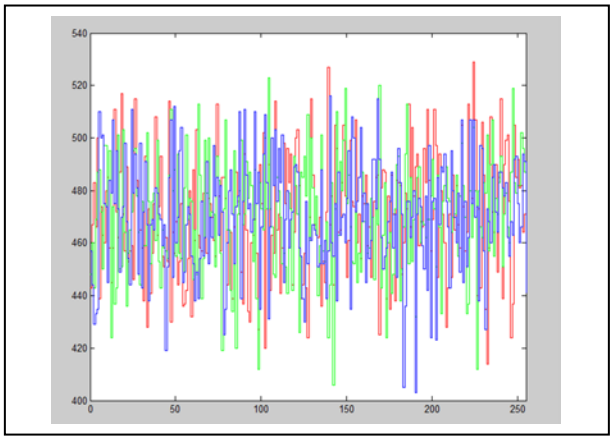

Fig. 9. Histogram of hashing and signed video 1 with DSA

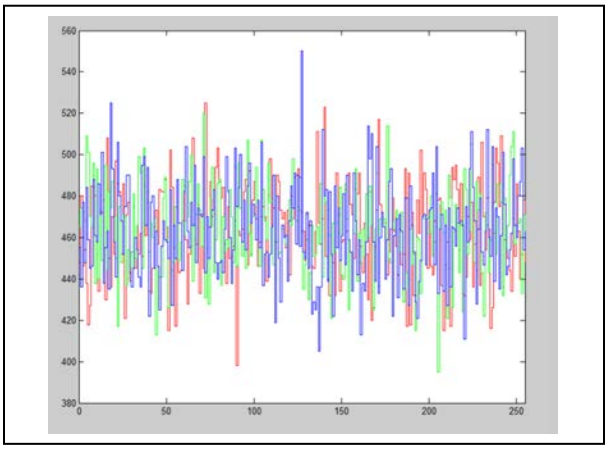

Fig. 10. Histogram of hashing and signed video 2 with DSA

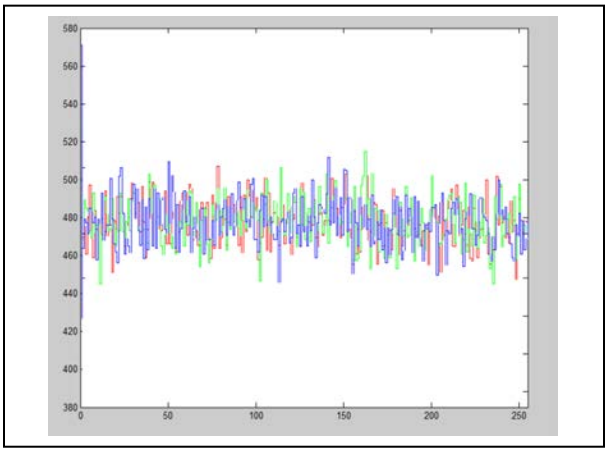

Fig. 11. Histogram of hashing and signed video 1 with ECDSA

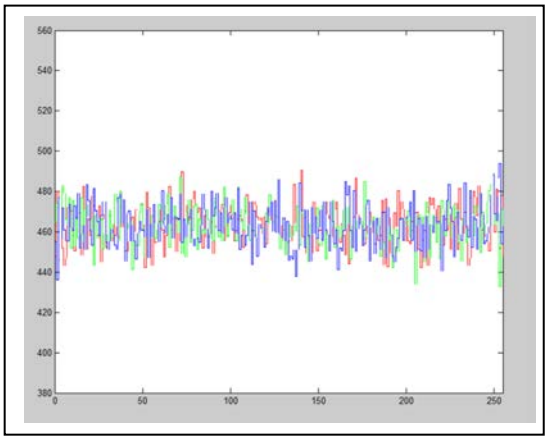

Fig. 12. Histogram of hashing and signed video 2 with ECDSA

We also compared the time of the signature process in second of these two algorithms depending on the size for a library containing a large number of H.264 videos. The speed of ECDSA over DSA is clearly denoted in figure 13 despite the growth in the size of the videos.

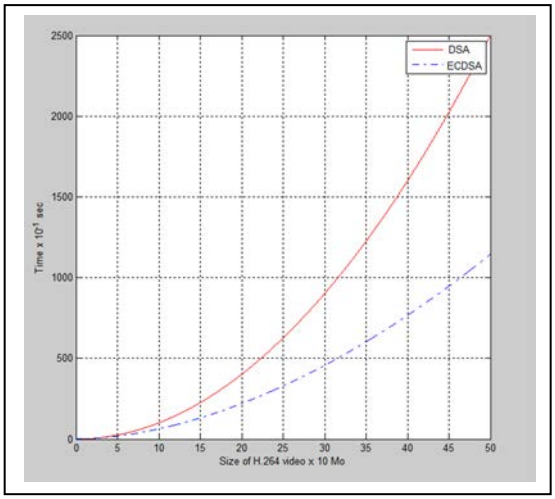

Fig. 13. Comparison of timing signing scheme- DSA vs ECDSA

VII. Conclusion

In this paper we compare the performance of two famous methods for electronic signing DSA and ECDSA in order to sign H.264 videos. We studied their speed, the number of gates used in the hardware implementation and the histograms' distribution of the some signed and hashed videos by MD5 function in the cases of these two algorithms.

References

[1] T. Plevyak, V. Sahin, Next Generation Telecommunications Networks, Services, And Management (Wiley-Ieee, 2010).

[2] M. Krause, H. F Tipton, Information Security Management Handbook. (6th ed. Auerbach Publications, CRC Press LLC, 2010).

[3] I. E. Richardson, The H.264 Advanced Video Compression Standard 2nd Edition (Wiley, 2010).

[4] A. Menezes, P. Van Oorschot, S. Vanstone, Handbook of applied cryptography, (CRC Press, 1996).

[5] Q. Zhang , Z. Li And C. Song, The Improvement Of Digital Signature Algorithm Based On Elliptic Curve Cryptography, 2011 Ieee Artificial Intelligence, Management Science And Electronic Commerce (Aimsec), Pp-1689 – 1691.

[6] P. Williams, Applied Cryptography (John Wiley & Sons, 1996).

[7] M. Stevens, Attacks on Hash Functions and Applications, Ph.D. Thesis, Dept. Computer Engineering, University of of Leiden, Netherland, 2012.

[8] FIPS 180-4, Secure Hash Standard (SHS)– Current version of the Secure Hash Standard (SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512), 2012.

[9] R. L. Rivest, The MD4 Message Digest Algorithm, 1990 CRYPTO Lecture Notes in Computer Science, vol. 537, Springer, pp. 303–311.

[10] R. L. Rivest, The MD4 Message-Digest Algorithm, Internet Request for Comments, 1990, RFC 1186; obsoleted by RFC 1320.

[11] R. L. Rivest, The MD5 Message-Digest Algorithm, Internet Request for Comments, 1992, RFC 1321.)

[12] S. Yu and K. Aoki , Finding Preimages in Full MD5 Faster than Exhaustive Search, 2009 Advances in Cryptology - EUROCRYPT 2009, Volume 5479 of the series Lecture Notes in Computer Science pp 134-15.

[13] M. O. Rabin, Digitalized Signatures, Foundations of Secure Computation (Richard A. Demillo, David P. Dobkin, Anita K. Jones, and Richard J. Lipton, eds.), Academic Press, 1978, pp. 155–168.

[14] H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen and F. Vercauteren, Handbook of Elliptic and Hyperelliptic Curve Cryptography, Dis. Math.Its App. 1st Edition, (2005).

[15] Y. Wang, Task Parallel Implementation of Matrix Multiplication on Multi-socket Multi-core Architectures, Algorithms and Architectures for Parallel Processing, 2015 15th International Conference, ICA3PP 2015, Zhangjiajie, China, Proceedings, Part III.

[16] A. Reyhani-Masoleh , M. A. Hasan, A new construction of Massey-Omura parallel multiplier over GF(2m), 2002 IEEE Transactions on Computers (Volume:51 , Issue: 5) PP-511-520.

[17] R. K. Kodali , C. N. Amanchi ; S. Kumar ; L. Boppana, FPGA implementation of Itoh-Tsujii inversion algorithm, 2014 Recent Advances and Innovations in , Engineering (ICRAIE), 2014, pp 1 – 5.

[18] J. López, R. Dahab, Fast Multiplication on Elliptic Curves Over *GF*(2m) without precomputation, 2002 Cryptographic Hardware and Embedded Systems, Volume 1717 of the series Lecture Notes in Computer Science pp 316-327.

[19] D. Johnson, A. Menezes, S. Vanstone, The Elliptic Curve Digital Signature Algorithm (ECDSA), 2001 International Journal of Information Security, Volume 1, Issue 1, pp 36-63.

[20] Itu, H.264: Advanced video coding for generic audiovisual services, International Telecommunication Union, 2003.