Computer Science and Engineering 2017, 7(1): 1-11
DOI: 10.5923/j.computer.20170701.01

Access Control of Medical Images using Elliptic Curve Cryptography through Effective Multi-Key Management in a Mobile Multicasting Environment

Tmkk Jinasena[1,*], Rgn Meegama[1], Rb Marasinghe[2]

[1]Department of Computer Science, University of Sri Jayewardenepura, Sri Lanka
[2]Department of Medical Education and Health Sciences, University of Sri Jayewardenepura, Sri Lanka

Abstract Access control is so vital in eHealth, because compromised sensitive medical data often leads to severe consequences for both patients and health workers, resulting in financial losses or even patient death. In this paper, we propose a technique to implement a dynamic and flexible access control mechanism to ensure present access rights of sensitive medical data in a collaborative medical discussion in a mobile environment. Initially, a symmetric and public key encryption using elliptic curve cryptography is used to encrypt a user session. At this stage, the elliptic curve is defined in a prime finite field with the characteristic of p where p is a prime and $p > 3$. Curve parameters a and b are carefully chosen to avoid vulnerable curves. Subsequently, unique public and private key pairs are generated for all the users in the session. Results show the importance of having optimal elliptic curve implementations for mobile usage.

Keywords Access Control, Elliptic Curves, Cryptography, Privacy, eHealth

1. Introduction

Digital content access control is one of the key concerns in computer security today, and it is more important than ever before. A huge amount of digital data is constantly flowing through the internet, mobile networks, and cable and satellite televisions, and electronic medical data exchanged among large groups through public networks, especially in mobile environments, is especially vulnerable. In most cases, it is necessary to protect data from unauthorized and inappropriate access and changes by defining what information users can view and modify. Access criteria are generally associated with roles, groups, locations, or times. However, there are three main types of access control methods: Discretionary Access Control (DAC), Mandatory Access Control (MAC), and Non-Discretionary (Role-Based) Access Control (RBAC) [1]. Furthermore, access control methods are classified as military or commercial based on their usage. MAC is based on Bell-LaPadula's 1973 multilevel security model, which is more concerned with confidentiality than integrity. In MAC, security policies are defined regardless of user operations. Thus, it is more suitable for military applications. On the other hand, DAC is the most used access control method. It is used in many operating systems, including UNIX, Windows 2000, and FreeBSD. The main disadvantage of DAC is the fact that its three-dimensional access control matrix has O(n2) growth. Finally, RBAC proposed by Ferraiolo and Kuhn in 1992 [1] blends MAC and DAC. It can be customized for individual applications regardless of policies [2-6].

In our case, we need to facilitate collaborative medical discussion over mobile devices where sensitive medical data, possibly large content, is shared through public networks while guaranteeing its C-I-A (Confidentiality, Integrity, and Availability) properties [3]. Moreover, we need to have a dynamic and flexible access control method to ensure the right access by the right user at the right time. However, there are no fixed access levels or roles for users. The one who initiates the communication becomes the coordinator of that session. Thus, s/he defines the access levels of the subordinates of the session. In another session, s/he can be a subordinate with low access privileges. Therefore, access control needs to be provided through the content. Moreover, we need to multicast the same content for multiple users with different access levels.

2. Background

Symmetric key cryptography is faster than asymmetric key cryptography mainly due to its small key sizes. However, it can only guarantee the confidentiality of the data. In a distributive environment, there are many security

* Corresponding author:
kasun@dscs.sjp.ac.lk (Tmkk Jinasena)
Published online at http://journal.sapub.org/computer

concerns, such as key distribution, authentication, authorization, integrity, and non-repudiation. To address such concerns, public key cryptography should be used. Therefore, in such scenarios, a combination of both symmetric and asymmetric key cryptography is being used to reap the benefits of both.

2.1. AES Cryptography

The Advanced Encryption Standard (AES), originally known as Rijndael, was introduced by Joan Daemen and Vincent Rijmen in 1998, and it was recommended by the National Institute of Standards and Technology (NIST) in late 2001 [7]. The AES is a symmetric-key block cipher with a fixed block size of 128 bits, and it uses keys of any multiples of 32 bits between 128 and 256. It is based on both substitution and permutation, and it is fast in both software and hardware implementations. The AES operates on a 4x4 column-major order matrix and has four main operations: AddRoundKey, SubBytes, ShiftRows, and MixColumns [8]. It has 10 to 14 cycles for each block based on its key size [9]. The SubBytes phase is a non-linear phase where it replaces each byte with a value in an 8-bit substitution box (s-box). In order to ensure good linear properties and avoid simple algebraic attacks, the s-box is constructed using the multiplicative inverse over Galois Field GF (2^8). The time complexity of the Biclique attack, which is faster than the full brute force attack, is $O(2^n)$. Thus, given the present computational power, it is virtually unbreakable [10-13].

2.2. RSA Cryptography

Cryptosystems usually increase their key sizes to obtain higher security, and they can be as big as 300 digits or even bigger. Thus, it has become problematic to store them in small devices safely in public-key cryptography. In 1977, Ron Rivest, Adi Shamir, and Leonard Adleman introduced a public-key encryption method named RSA based on a mathematical problem called factorizing large integers where its time complexity is polynomial [14]. Currently, it is the public-key encryption method recommended by the NIST. However, it is being threatened by the rapid growth in computing power.

In 1999, a polynomial time algorithm for integer factorization and computation of discrete logarithms on quantum computers was discovered by Peter Shor [15]. This meant that once the quantum computers reached the 1000 Q-bit range, RSA could easily be broken. Recently, a few alternative approaches have emerged. Among them, Elliptic Curve Cryptography (ECC) has been promising due to its unique capabilities [6, 16-19].

2.3. ECC

The new paradigm of public-key encryption using ECC was suggested independently by Neal Koblitz and Victor S. Miller in 1985 [14]. The main advantage of the ECC over RSA is its ability to provide equal security for significantly smaller key sizes. Thus, it provides the most security per bit of any known public-key scheme. Table 1 shows the equivalent security key sizes of ECC, RSA, and symmetric key cryptography. Although the computational overhead of both RSA and ECC grows as O (N^3), where N is the key length in bits, ECC is more efficient and stronger than RSA. In fact, it is faster than all the other methods. Furthermore, it needs less computational power and less space. Hence, it is ideal for limited bandwidth communications, wireless devices, and PC cards [20-23].

Table 1. Comparable Key Sizes for Equivalent Security

Symmetric scheme (key size in bits)	ECC-based scheme (size of n in bits)	RSA/DSA (modulus size in bits)
56	112	512
80	160	1024
112	224	2048
128	256	3072
192	384	7680
256	512	15360

ECC introduced a whole new algebra; thus, it is not vulnerable to common attacks. Instead of prime factorization or polynomial arithmetic, it uses Elliptic Curve (EC) arithmetic to perform computations. The security of ECC depends on the 1000-year-old unsolved mathematical problem called the Elliptic Curve Discrete Logarithm Problem (ECDLP), which refers to how difficult it is to determine k for a given kP and P. The ECDLP problem appears to be much more difficult than the integer factorization problem and the discrete logarithm problem (DLP) of $\mathbb{Z}_p$. The order of $E(\mathbb{Z}_p)$ is defined as $\#E(\mathbb{Z}_p)$. According to Hasse's theorem, $p + 1 - 2\sqrt{p} \leq \#E(\mathbb{Z}_p) \leq p + 1 + 2\sqrt{p}$ [24]. The best known attack for ECC so far is Pollard's rho attack, where it runs in exponential time, which is $\frac{\sqrt{\pi n}}{2}$ [25].

The time complexity of RSA is sub-exponential, whereas it is exponential for ECC. **Figure 1** shows the variation of the time complexities against the key sizes.

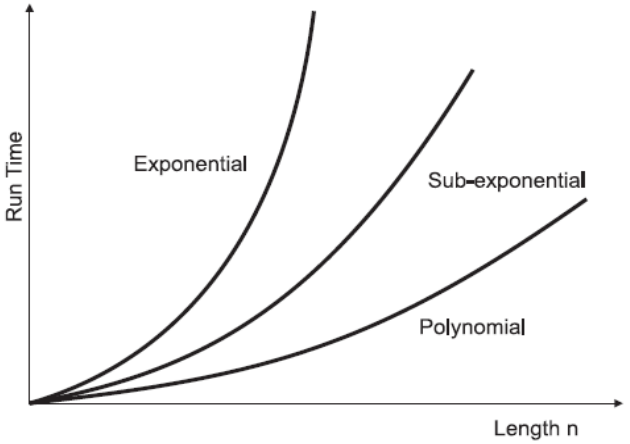

Figure 1. Time complexity vs key size

The general equation for an EC is as follows:

$$y^2 + b_1xy + b_2y = x^3 + a_1x^2 + a_2x + a_3$$

ECs over real numbers use a special class of ECs of the form $y^2 = x^3 + ax + b$, also known as the Weierstrass equation of characteristic [24]. The discriminant of the polynomial $16(4a^3 + 27b^2)$ should not be zero in order to possess three distinct roots for a given EC.

Furthermore, singular ECs are not suitable for cryptography, since they can be easily cracked due to the fact that they are isomorphic to either addition or multiplication. Super-singular ECs should also be avoided, since they are vulnerable to MOV attack. However, we cannot use real numbers for encryption, since they are prone to round-off error. Thus, we restrict the values of x and y to a prime finite field $\mathbb{Z}_p$. Such curves are denoted by the form of $y^2 = (x^3 + ax + b) \pmod p$. Here, p is a prime, and $p > 3$. P is also known as the characteristic of the field. Further, the curve parameters $a, b \in \mathbb{Z}_p$ should be carefully chosen to satisfy the condition $4a^3 + 27b^2 \neq 0 \pmod p$ [26]. However, when a finite field is chosen, pretty curves and lines are not applicable, yet the underlying algebra works as usual. When the curve is defined on a finite field $\mathbb{Z}_q$ with q elements, the number of points on the curve N, including the additive identity element O, is bounded by $|N - (q + 1)| \leq 2\sqrt{q}$.

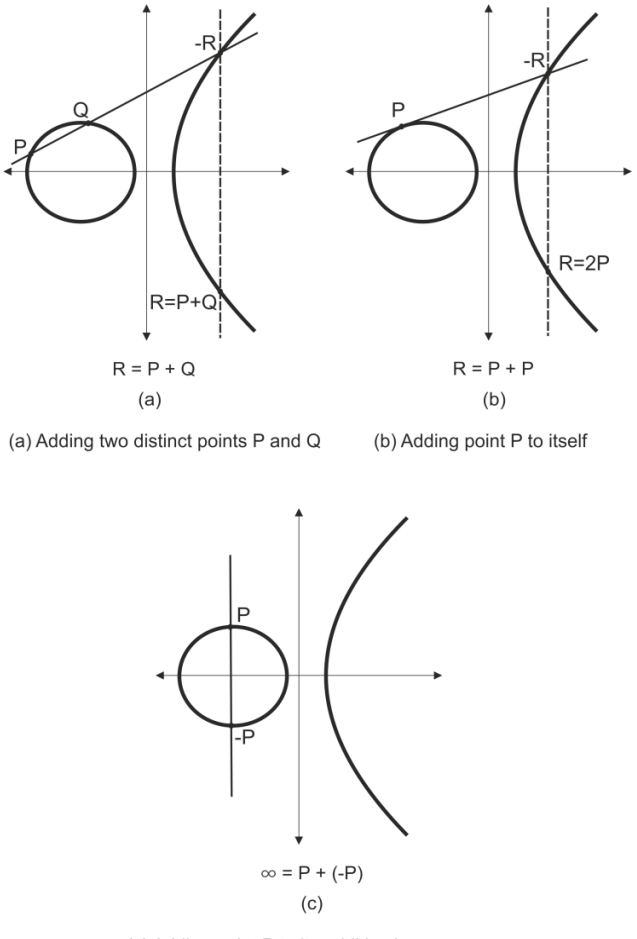

Figure 2. Three adding cases in an EC

2.3.1. Point Addition

The point addition in ECs has nothing to do with the conventional arithmetical addition. It is the group operator for the points on an EC. Since it is commutative, EC forms an Abelian group. Let E be an EC over a field K, given by an equation $y^2 = x^3 + ax + b$. We begin by defining a binary operation + on E(K). For a given P and Q ∈ E(K), the following equations can be used to compute a third point

$$R = P + Q \in E(K):$$
$$x_R = \alpha^2 - x_P - x_Q$$
$$y_R = \alpha\,(x_R - x_P) + y_P$$

where $\alpha = \frac{y_Q - y_P}{x_Q - x_P}$

The point addition involves five subtraction operations, one addition, three modular multiplications, and one inversion operation. As shown in **Figure 2**, the EC has three adding cases. Case (a) shows the addition of two distinct points P and Q on the curve, whereas case (b) shows the adding of the point to itself. Finally, case (c) shows the adding of a point's mirror or complement to itself in order to obtain the resulting point O at infinity.

2.3.2. Point Doubling

Point doubling or scalar multiplication in ECs also does not represent conventional arithmetical multiplication. As we have seen above, if we extend the point addition to double a point P ∈ E(K), we can obtain a point R ∈ E(K) such that R = P + P = 2P by using the following equations:

$$x_R = (\lambda^2 - 2x_P)$$
$$y_R = (\lambda(x_P - x_R) - y_P)$$

where $\lambda = \left(\frac{3x_P^2 + a}{2y_P}\right)$

In ECC, scalar multiplication and inversion are very time-consuming operations. However, ECs have properties that enable the efficient implementation of scalar multiplication. For example, rather than computing $126P$ with 126 additions as follows:

$$126P = P + P \ldots + P.$$

It can be calculated by six doublings and five additions as follows:

$$126P = 2(2(2(2(2P + P) + P) + P) + P) + P).$$

Further, the inverse addition or the subtraction of ECs is easy to compute. This can be further reduced to seven doublings and one subtraction as follows:

$$126P = 2(2(2(2(2(2P))))) - P).$$

Figure 3 shows the case of such consecutive point doubling in an EC. However, this scalar multiplication is the key to ECC. If we add point P, n times, a general case can be written as $R = n.P \in E(K)$. This operation is elegant, as it is not easy to find the number n even though we know the points P, R, and the curve. Thus, n can be used as a secret or private key, whereas R together with P and the curve becomes its unique public key.

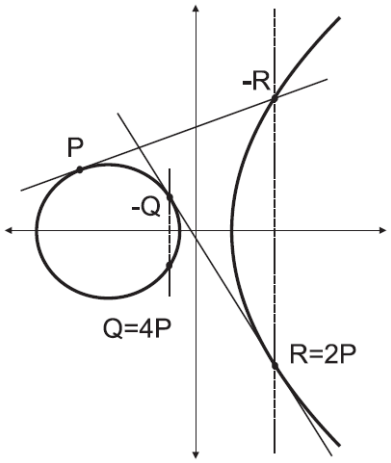

Figure 3. Doubling point P repeatedly to find 4P

Let p = 13 be a prime number and consider the EC $E_{13}(5;10): y^2 = x^3 + 5x + 10$. We obtain quadratic residues $\mathbb{Q}_{13}$ = {1, 3, 4, 9, 10, 12} from the reduced set of residues $\mathbb{Z}_{13}$ = {1, 2, . . 11, 12}. This means that the valid points of the EC are as follows:

$E_{13}(5; 10)$ = [(0 : 1 : 0); (0 : 6 : 1); (0 : 7 : 1); (1 : 4 : 1); (1 : 9 : 1); (3 : 0 : 1); (4 : 4 : 1); (4 : 9 : 1); (5 : 2 : 1); (5 : 11 : 1); (6 : 3 : 1); (6 : 10 : 1); (8 : 4 : 1); (8 : 9 : 1); (9 : 2 : 1); (9 : 11 : 1); (12 : 2 : 1); (12 : 11 : 1)]

The following commands can be used in SAGE to generate the valid points of $\mathbb{Z}_{13}$, as shown in **Figure 4**:

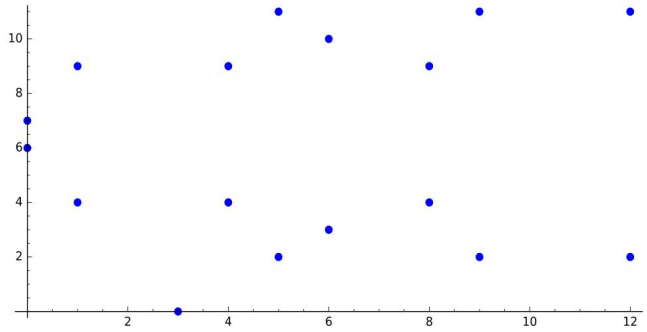

Figure 4. Valid points of Z_{13}

SAGE: E=EllipticCurve(GF(13),[5,-3])
show (E.plot())

Let P = (5; 2) ∈ E_{13}(5; 10); Multiplicative values of P are as follows:

1*P = (5 : 2 : 1), 2*P = (0 : 7 : 1), 3*P = (9 : 2 : 1), 4*P = (12 : 11 : 1) , 5*P =(8 : 9 : 1), 6*P = (4 : 9 : 1), 7*P = (1 : 9 : 1), 8*P = (6 : 3 : 1), 9*P = (3 : 0 : 1), 10*P = (6 :10 : 1), 11*P = (1 : 4 : 1), 12*P = (4 : 4 : 1), 13*P = (8 : 4 : 1), 14*P = (12 : 2 : 1), 15*P = (9 : 11 : 1), 16*P = (0 : 6 : 1), 17*P = (5 : 11 : 1), 18*P =P*0= (0 : 1 : 0),19*P =1*P= (5 : 2 : 1), 20*P =2*P= (0 : 7 : 1)

2.3.3. Elliptic Curve Diffie-Hellman (ECDH) Key Exchange

The DH key-agreement scheme was proposed by Whitfield Diffie and Martin Hellman in 1976 [27]. The DH scheme is used to exchange secret keys between two or more parties. However, it is an interactive way of exchanging keys where the sender and the receiver need to be in the process at the same time. ECDH is the EC version of the DH scheme. If (P_A, S_A) and (P_B, S_B) are public and private keys of users U_A and U_B, respectively, where P_A and P_B are points on an EC, U_A and U_B can compute a secret key K as $K_A = S_A \times P_B$ by U_A and $K_B = S_B \times P_A$ by U_B [28, 29, 30]. According to the properties of ECDH,

$$K_A = S_A \times P_B = S_A \times (G \times S_B)$$
$$= (S_A \times G) \times S_B = P_A \times S_B = K_B = K.$$

2.3.4. ElGamal Encryption using ECC

ElGamal encryption was proposed by Taher ElGamal in 1984 [31]. He was the first person to propose the idea of using a DLP for public-key cryptography. However, ElGamal encryption is a passive way of exchanging keys. In other words, the sender can encrypt the data off-line, and the receiver can decrypt it later. Assume that user U_B wants to send a secret key to user U_A; U_A chooses a random point $\alpha \in EC(a, b)$ and multiplies it by a random number S_A where $1 \leq S_A \leq n-1$ to find β such that $\beta = \alpha \times S_A$. S/he lets α and β be publicly known to everybody and secretly keeps S_A as his/her private key. U_B chooses a random point $\gamma \in EC(a, b)$ and calculates C_1 such that $C_1 = \gamma \times \alpha$ and C_2 such that $C_2 = K + \gamma \times \beta$ where K is the secret key s/he wants to share with U_A. U_B sends the message (C_1, C_2) to U_A, and U_A decrypts the message and gets the key K as follows [32, 33, 34].

$$K = C_2 - S_A \times C_1.$$

2.3.5. Elliptic Curve Digital Signature Algorithm (ECDSA)

ECDSA is a variation of the digital signature algorithm that consists of two parts of the signature: $sign_1$ and $sign_2$. Let H be the hash value of the message, which is an integer, and k is a random number, $sign_1 = (k \times G)_x \bmod p$. That means $sign_1$ only keeps the x-coordinate modulo p of $k \times G$. If $sign_1 = 0$, then try another k. Now, $sign_2 = k^{-1}(H + S_A . \text{sign}_1) \bmod p$ where k-1 is the multiplicative inverse of k modulo p, which is obtained by the extended Euclidean algorithm, and S_A is the private key of user A. When user B receives ($sign_1$, $sign_2$) along with the original document, s/he can compute the $w = sign_2^{-1} \bmod p$, $u_1 = H.w \bmod p$, $u_2 = sign_1.w \bmod p$, and a point on the curve $(x, y) = u_1 \times G + u_2 \times P_A$ where P_A is the public-key of user A. Now, s/he can verify the integrity of the message as well as the sender by comparing $sign_1 \equiv x \bmod p$. This method has been used by Bitcoin to verify whether the digital coins are spent by the right users. However, it is necessary to use a different k for each signature. For example, if the same k is used for two signs ($sign_1$, $sign_2$) and ($sign_1$', $sign_2$'), k can be computed as $sign_2 - sign_2' = k^{-1}(H–H')$. Finally, the secret key S_A can be computed by using the equation $sign_2 = k^{-1}(H - S_A.sign_1)$. This technique was used to crack the Sony PlayStation 3 (PS3) in 2010 [35]. Moreover, it was used in an incident where the users of Android Bitcoin Wallet lost money due to faulty random number generators in August 2013 [36, 37, 38, 39].

2.3.6. Elliptic Curve Integrated Encryption Scheme (ECIES)

ECs cannot be directly used for encryption. Thus, in practice, they are combined with symmetric key algorithms and used as a hybrid scheme. In such cases, the EC is only used to encrypt the symmetric key that was used to encrypt the original data. ECIES is a combination of the Key Encapsulation Mechanism (KEM) and the Data Encapsulation Mechanism (DEM). The KEM is responsible for mapping a symmetric key to a point in the EC. The task of the DEM is to encrypt the plain text using the above symmetric key and authenticate the ciphertext [8, 23, 40].

2.3.7. Windows Media Digital Rights Management (WM-DRM)

Digital Rights Management (DRM) is associated with technologies and algorithms that help a content provider to enforce limitations, such as who can use the media content provided by them and in what way. ECC has been used for DRM for a while.

The Windows Media framework is a DRM that allows third-party vendors to create and distribute revenue-generating content. In WM-DRM, they use an EC-based encryption method to protect the access rights of their content, such as audio and video. It was a proprietary algorithm, and from 1999 to 2003, they released three versions of it. However, all three versions were compromised long ago. Other providers, such as Apple and Sony, have their own methods as well. According to Kerckhoffs's principle, the secrecy of a cryptosystem should only depend on the key [41]. However, all these systems have been implementing security by hiding their algorithms [42, 43, 44, 45, 46].

2.3.8. Sony PlayStation

Sony's PS3 console does not allow users to run codes that are not signed by one of its trusted parties. In other words, the developer first has to give his source code to some trusted party and get his signature on the binary. The Sony bootloader checks the signature before it executes the code. Sony has used ECC to implement its security mechanism where a secret key pair within the system is used to communicate with the license server. As we have explained above, the weakness of not avoiding the same random number twice enabled hackers to break it easily and pass their keys to others [35, 47].

3. Methodology

3.1. Symmetric Key Encryption

Although symmetric key encryption is faster, it requires a large number of keys in a multi-user environment. In particular, when the same content needs to be transferred to multiple users with different accessibility levels, payloads have to be encrypted for each user separately, which is very inefficient and time consuming. Therefore, in our proposed method, we first encrypt the content using a symmetric key and then encrypt the header separately for each user. The symmetric key encryption method is defined as $\Pi(E, D)$ where $c \leftarrow E_{k_i}(p)$ and $p \leftarrow D_{k_i}(c)$. Here, p represents the plain text, c represents the ciphertext, and k_i represents the symmetric session key of the i^{th} session. Further, E_{k_i} and D_{k_i} are the encryption and decryption algorithms, respectively, under the symmetric key k_i such that $D_{k_i}(E_{k_i}(M)) = M$. The AES is a well-tested, robust, and relatively efficient algorithm. As a result, we have selected it as our symmetric key algorithm. However, under the present hardware capabilities, its efficiency is insufficient for encrypting real-time video, even for videos of Phase Alternating Line (PAL) quality [13, 48].

3.2. Public-key Encryption

In this research, apart from ensuring confidentiality, we need to provide different access levels for each user separately. That is, for the same content, one user can edit it, whereas another can only read it. On the other hand, regarding another content, rights can be the opposite. Because of this dynamic nature, traditional role-based access control would not be an effective method.

In our approach, we define a public-key encryption method as follows:

$$\Theta(\text{KeyGen}, \varepsilon, \delta) \text{ as } c \leftarrow \varepsilon_{P_j}(p) \text{ and } p \leftarrow \delta_{S_j}(c)$$

Where ε and δ are the encryption and decryption algorithms, respectively, such that $\delta_{S_i}(\varepsilon_{P_i}(M)) = M$ and $\delta_{P_i}(\varepsilon_{S_i}(M)) = M$ to encrypt the secret symmetric key of the content.

ECC over a prime finite field has been chosen as the public-key encryption method. First, we define a prime finite field $\mathbb{Z}_{\boldsymbol{P}}$ to generate unique public and private key pairs (P_i, S_i) for all the users $U_s = U_1, U_2, \ldots U_i \ldots U_n$ in the system. Next, we define a base point or the generator G of the field, such that $n \times G = O$ for a large n. The EC, prime finite fields $\mathbb{Z}_{\boldsymbol{P}}$, and the base point G are publicly known to everybody in the system. Thus, each user U_i can choose a random number $1 < k < (n-1)$ and multiply G, k times to get another point R in the curve such that $R = k \times G$. This k becomes his/her private key S_i, whereas R, together with G, and EC become his public key P_i.

There are some vulnerable curves. Thus, the curve parameters p, a, and b should be chosen to define a suitable EC with higher robustness. If so, according to the properties of ECC, it would be difficult for an attacker to compute a user's private key even s/he knows all the other information. However, in our first approach, we define different public and private key pairs $\{(P_{i_1}, S_{i_1}), \ldots (P_{i_j}, S_{i_j})\}$ for the same user U_i for his/her different accessibility levels $L_i = \{ L_{i1}, L_{i2}, \ldots, L_{ij} \}$. In other words, if user U_A wants to send read-only content to user U_B, U_A uses U_B's read-only public key $P_{\text{Bread-only}}$ to encrypt the header of the document or the symmetric key. Hence, user U_B has to use his read-only private key $S_{\text{Bread-only}}$ to get the symmetric key in order to open the document. Because we use our own application to

view the content, decryption happens within the application. Therefore, user U_B will not be able to see the decrypted symmetric key.

However, managing a large number of key pairs for a single user is the biggest problem in this approach. Thus, we move to our second approach, which is defining the access-level key pairs associated to the master key pair. That means the application itself can generate a relevant access-level key pair once the user gives his/her master key. For example, in a public network, U_B can compute U_A's read-only private key using U_A's master key as $P_{Aread\text{-}only} = P_{Amaster} \times L_{read\text{-}only}$ where $L_{read\text{-}only}$ is a constant for all users. On the other hand, U_A can compute his/her read-only private key as $S_{Aread\text{-}only} = S_{Amaster} + L_{read\text{-}only}$.

As a result, it is sufficient for a user to worry about managing a single key pair. In both approaches above, another issue is overlapping a user's accessibility key with another's master or access-level keys. To avoid such scenarios, two methods have been applied. One is to use a large key space with a high-order n to reduce the possibility of overlapping keys. The other is to use two key spaces ψ and $\mathbb{Z}_P$ for master keys and access-level keys separately. Because we need to use the same master key to compute access-level keys, the master key space ψ is defined in a subfield of the original field $\mathbb{Z}_P$. However, in our second approach, there is a fixed relationship between a user's master keys and his/her access-level keys. This gives more opportunities for an attacker to find one of his/her keys, which would thus compromise all of his/her keys.

In our third approach, we compute access-level keys in a random manner rather than a fixed association. This results in a different access-level key even for the same access level of the same user. For this, we used an extended version of the ElGamal encryption algorithm over the EC. U_A uses U_B's public key to send a secret message X to U_B where X is another point on the curve.

For example, if U_A wants to send a read-only key to U_B, U_A selects a random number k and computes $C_1 = G \times k$. Then, U_A computes $C_2 = X_{read\text{-}only} + (k + L_{read\text{-}only}) \times P_{BMaster}$. Subsequently, s/he sends (C_1, C_2) to U_B where U_B finds the secret point $X_{read\text{-}only}$ as in equation 1.

$$X_{\text{read-only}} = (C_2 - C_1 \times (S_{B_{Master}} + L_{read-only})) + (C_1 - P_{B_{Master}}) \times L_{read-only}. \quad (1)$$

Proof:

$$\begin{aligned}
&(C_2 - C_1 \times (S_{B_{Master}} + L_{read-only})) + (C_1 - P_{B_{Master}}) \times L_{read-only} \\
&= ((X_{\text{read-only}} + (k + L_{read-only}) \times P_{B_{Master}}) - ((G \times k) \times (S_{B_{Master}} + L_{read-only}))) + (((G \times k) - P_{B_{Master}}) \times L_{read-only}) \\
&= ((X_{\text{read-only}} + (k + L_{read-only}) \times P_{B_{Master}}) - ((G \times k \times S_{B_{Master}}) + (G \times k \times L_{read-only}))) + (((G \times k) \times L_{read-only}) - (P_{B_{Master}} \times L_{read-only})) \\
&= ((X_{\text{read-only}} + (k \times P_{B_{Master}}) + (L_{read-only} \times P_{B_{Master}})) - (G \times S_{B_{Master}} \times k)) - (P_{B_{Master}} \times L_{read-only}) \\
&= ((X_{\text{read-only}} + (k \times P_{B_{Master}}) + (L_{read-only} \times P_{B_{Master}})) - (P_{B_{Master}} \times k)) - (P_{B_{Master}} \times L_{read-only}) \\
&= X_{\text{read-only}}.
\end{aligned}$$

Though we still use that fixed-number $L_{read\text{-}only}$, $X_{read\text{-}only}$ becomes a one-time key due to the random number k.

Assume that U_A wants to send the symmetric key X to U_B under read-only permission. Consider the above $E_{13}(5, 10)$ as our EC and take $G = (5, 2) \in E_{13}(5, 10)$. Further, take the private keys of U_A and U_B as $S_A = 5$ and $S_B = 10$, respectively, by making their public keys $P_{AMaster} = (8, 9) \in E_{13}(5, 10)$ and $P_{BMaster} = (6,10) \in E_{13}(5, 10)$. Moreover, let $L_{read\text{-}only} = 3$ and the symmetric key X maps to the point $X_{read\text{-}only} = (4, 4) \in E_{13}(5, 10)$.

Now, take 7 as the random number k between 1 and 17 and compute $C_1 = G \times k = (5, 2) \times 7 = (1, 9) \in E_{13}(5, 10)$.

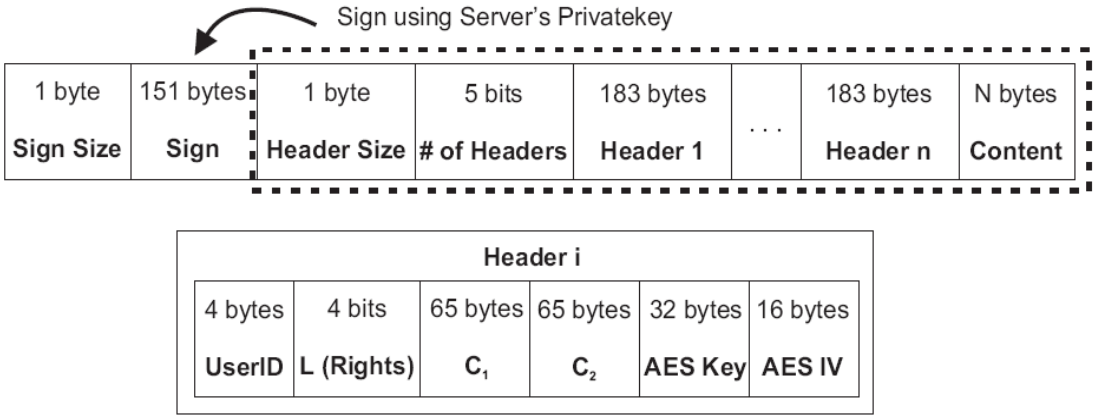

Figure 5. File structure

Then compute $C_2 = (4, 4) + (7 + 3) \times (6, 10) = (12, 11) \in E_{13}(5, 10)$. Once U_B receives (C_1, C_2) and $L_{read\text{-}only}$, s/he finds $X_{read\text{-}only} = ((12, 11) – (1, 9) + (10 + 3)) + ((1, 9) – (6, 10)) \times 3 = (4, 4) \in E_{13}(5, 10)$. However, this is only for a single user pair. Our fourth attempt is to extend this to a multicasting environment to avoid separate transmission for its users. Therefore, if the same content is shared with multiple users under different access rights, a single multi-casting can be used. However, this is simply done by concatenating multiple secret key encryptions to the header. Each encrypted field in the header can have a specific access right L_i regarding a specific user U_i. Finally, the application is responsible for implementing the required access control corresponding to the L_i where i represents the access rights, such as read-only, save, edit, etc. In other words, if the U_A wants to send an image for U_B with read-only permissions, U_C with save permissions, and U_D with edit permissions, U_A first needs to encrypt the image using AES symmetric key encryption. Then U_A needs to encrypt they AES key and IV using each receivers public key P_i and his/her access level L_i. For example, P_B with $L_{read\text{-}only}$, P_C with L_{save}, and P_D with L_{edit}. Structure of one such encrypted header is shown in the bottom diagram of the Figure 5. Next, these individual headers will concatenate as shown in the above diagram of the Figure 5. Finally find the digest of the whole message and sign using U_A's private key. Once U_A multicast this to all the users, each user can verify the sender and the integrity of the message using message signature and the digest. Moreover, each can get their header and decrypt it. Because, the header has encrypted using the respective user's public key, only that user will be able to decrypt it. Finally, the mobile application's responsibility is to execute the received access rights for that user. For example, U_A's app does not allow U_A to edit or save this image, whereas U_B can save the same image. Neither users nor apps have fixed access roles in the system. Access permission comes with the content and app dynamically adjusts to it.

3.3. File Structure

As shown in **Figure 5**, the file starts with the 1-byte field, which shows the sign size. Next, it contains the signature of the remaining part that is signed using the server's private key. By verifying the signature, the application can confirm the integrity of the content as well as the sender. Because it is multicasting to several users with different access rights, there is a separate header for each user. The client application can find its header by checking the User ID. Then, it uses its private key together with access rights field L_i, C_1, and C_2 to decrypt the AES key.

Once the application obtains the symmetric key, it can decrypt the content and allow the user to perform the rights defined in the L_i.

3.4. Algorithms

Algorithm 1 shows the creation of the final encrypted file with access rights for multicasting.

Algorithm 1: File Encryption

Input: data file, sender's public and private keys, receivers' public keys, rights list L_i

Output: Encrypted file

1. Read file and compress the data
2. Randomly generate AES key and IV
3. Encrypt the data using AES key and IV
4. for i=1 to no of users do
5. Encrypt AES key and IV using receiver publickey[i] and rights L[i] and append to the encrypted file
6. Sign header and data using server's private key and append to the encrypted file
7. end for
8. return: encrypted_file

Algorithm 2 shows the process of decryption for a given recipient.

Algorithm 2: File Decryption

Input: Encrypted file, receiver's ID, receiver's private key

Output: Decrypted file

1. Read the sign and verify the integrity.
2. Get the respective user's header.
3. Decrypt the header using receiver's private key and get the AES key and IV
4. Decrypt the data using AES key and IV
5. Decrypt the data file and create the decrypt_file.
6. return: decrypt_file

4. Results and Discussion

In order to identify efficient algorithms as well as bottlenecks, several algorithms were tested thoroughly. When the encryption of large medical images happens at the server side, it can be easily empowered by a high-performance server to increase its efficiency. However, the real bottleneck happens at the mobile device due to its limited resources. The mobile application was tested using Android 5.1 and a Samsung Galaxy Core Prime phone with 1.2 GHz Quad-core Cortex-A53 CPU, Adreno 306 GPU, and 1GB RAM.

As shown in Table 2, AES symmetric key encryption and decryption times are measured against the different image sizes and different key sizes. Because most of the processes running in the background are Worker Threads or AsyncTask, run times are significantly different from one run to another. Thus, each process is repeated 50 times, and the minimum time is taken as its best run time. The results show that it is almost the same in the long run for any key size. However, the IV generation time is significantly longer than the key generation time.

Then, the AES encryption and decryption sizes and times are measured against the file sizes. PKCS5Padding is used as the padding method. According to the results shown in Table 2, the encrypted and decrypted sizes are slightly larger due to padding. These sizes are not dependent on the AES key size due to AES's fixed block size of 128 bits for all its key sizes.

As an enhancement, compression could be applied to remove coding redundancies prior to the encryption to reduce the file size and the encryption time. However, compression after encryption could compromise the strength of the algorithm, because it destroys the uniform distribution of the changes across the whole ciphertext [48].

Table 2. AES Encryption and Decryption Times (ms)

Image	Key Size (bits)	Encryption Time	Decryption Time
Image 1	128	80.5	98.6
Image 1	192	80.5	109.3
Image 1	256	80.5	119.2
Image 2	128	287.4	341.2
Image 2	192	287.4	380.4
Image 2	256	287.4	418.7
Image 3	128	695.0	810.7
Image 3	192	695.0	903.6
Image 3	256	695.0	996.3
Image 4	128	1187.9	1382.8
Image 4	192	1187.9	1543.0
Image 4	256	1183.9	1416.0
Image 5	128	1710.3	1602.7
Image 5	192	1696.5	2605.7
Image 5	256	1699.1	2602.6
Image 6	128	1702.7	2579.7
Image 6	192	1707.9	2602.4
Image 6	256	1710.3	2624.0
Image 7	128	3094.2	4391.0
Image 7	192	3098.0	4500.0
Image 7	256	2383.7	3651.7

Table 3 shows that the encryption and decryption times increase with the file size but do not depend on the key size. Decryption times are slightly longer than encryption times. Although the local and mobile network conditions are improving daily, network latencies significantly affect the run time. Thus, images are kept in the local storage.

Next, we measured the run times of asymmetric encryption and decryption algorithms. While we chose ECC for the final system, RSA was also implemented to compare the results. Table 4 shows that the RSA key generation time increases exponentially as the key size increases.

One of the objectives of the research is to find the most optimal curve in terms of its key size, robustness, and run time to implement in a mobile device. Therefore, different key sizes are tested to compare its security against the run time. However, finding curve parameters for a secure curve is not as easy as one would think. To avoid the pitfalls of choosing vulnerable curves, the NIST-approved ECs together with ECC algorithms implemented in the Spongy Castle library were used. Other necessary parameters were taken from the Certicom documentation [49, 50].

According to Table 5, EC key generation time also increases with field size. However, sect571r1 is significantly longer.

Table 6 and Table 7 show the sign and verification times and lengths for RSA and EC against different hash algorithms. It clearly shows that RSA sign lengths are significantly longer than EC sign lengths.

Table 3. AES Encryption/Decryption Length vs File Size (bytes)

Image	Original Size	Encrypted Size	Decrypted Size
Image 1	1440000	1440016	1441792
Image 2	5184000	5184016	5185536
Image 3	12582912	12582928	12584960
Image 4	21496000	21496016	21497856
Image 5	39923712	39923728	39925760
Image 6	40144896	40144912	40146944
Image 7	55987200	55987216	55988224

Table 4. RSA Key Size vs Key Generation Time

Key Size (bits)	Time (ms)
128	6
256	11
512	33
1024	74
2048	540
3072	620
4096	3527

Table 5. ECC Curve vs Key Generation Time

Name	Size	Time (ms)
brainpoolp160r1	160	98
c2pnb163v3	163	132
brainpoolp192t1	192	122
prime192v3	192	120
secp192k1	192	117
brainpoolp224r1	224	177
brainpoolp256r1	256	198
P-256	256	226
brainpoolp320t1	320	290
brainpoolp384r1	384	385
brainpoolp512r1	512	583
sect571r1	571	1817

Table 6. RSA Sign and Verification Times (ms) and Sign Size (byte) Vs Hash Algorithm

Key Size & Hash	Sign Time	Verify Time	Signature Size
1024 with SHA1	5.2	0.4	128
1024 with SHA256	5.2	0.4	128
2048 with SHA1	30.8	1.1	256
2048 with SHA256	30.9	1.1	256
3072 with SHA1	93.9	2.2	384
3072 with SHA256	93.9	2.2	384
4096 with SHA1	211.1	3.7	512
4096 with SHA256	211.2	3.6	512

Table 7. ECC Sign and Verification Times (ms) and Sign Size (byte) vs Hash Algorithm

Name	Sign Time	Verify Time	Signature Size
brainpoolp160r1 with SHA1	5.0	167.0	47
brainpoolp160r1 with SHA256	5.0	166.5	47
c2pnb163v3 with SHA1	9.3	225.9	48
c2pnb163v3 with SHA256	9.3	224.0	47
brainpoolp192t1 with SHA1	5.8	206.1	54
brainpoolp192t1 with SHA256	5.9	208.3	54
prime192v3 with SHA1	5.8	206.2	56
prime192v3 with SHA256	5.9	208.8	55
secp192k1 with SHA1	6.4	208.7	55
secp192k1 with SHA256	6.6	211.8	55
brainpoolp224r1 with SHA1	7.6	264.4	64
brainpoolp224r1 with SHA256	7.4	262.5	63
brainpoolp256r1 with SHA1	8.5	313.7	70
brainpoolp256r1 with SHA256	8.3	318.0	71
P-256 with SHA1	8.0	309.9	70
P-256 with SHA256	7.8	306.0	71
brainpoolp320t1 with SHA1	12.9	438.0	87
brainpoolp320t1 with SHA256	12.6	439.4	88
brainpoolp384r1 with SHA1	18.7	584.4	102
brainpoolp384r1 with SHA256	18.9	571.9	102
brainpoolp512r1 with SHA1	31.4	985.8	135
brainpoolp512r1 with SHA256	31.2	999.6	135
sect571r1 with SHA1	69.7	3399.3	150
sect571r1 with SHA256	69.6	3409.5	151

Moreover, EC times are much better compared to respective RSA sign times. However, RSA takes significantly less time for verification. As before, sect571r1 is much slower compared to the other curves. Further, it shows that the sign and verification times do not depend on the hash algorithm.

Table 8 shows the DH key exchange times over EC and ElGamal encryption and decryption times over EC.

Table 8. Curve vs Run Time

Name	Size	ECDH
brainpoolp160r1	160	121.0 ms
c2pnb163v3	163	29.0 ms
brainpoolp192t1	192	145.9 ms
prime192v3	192	395.0 ms
secp192k1	192	6.9 ms
brainpoolp224r1	224	183.1 ms
brainpoolp256r1	256	213.4 ms
P-256	256	14.7 ms
brainpoolp320t1	320	272.0 ms
brainpoolp384r1	384	316.7 ms
brainpoolp512r1	512	476.4 ms
sect571r1	571	241.5 ms

As shown in Figure 6, some curves are significantly faster than the others, perhaps due to code optimizations, such as using the best methods to compute EC operations.

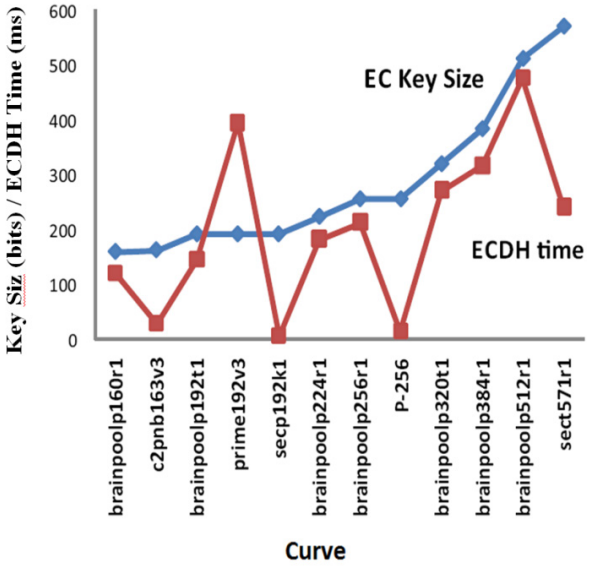

Figure 6. ECDH time and key size

Theoretically, run time should increase as the key size increases. However, the results show that certain curves with high key sizes are faster than some of the curves with lower key sizes. This is mainly due to the code optimization of certain algorithms against the others.

Since the NIST's smallest recommended key size is 224, developers have not put much effort into the optimization of smaller curves [51].

Furthermore, the expected run time of ECC is shorter than its equivalent secured RSA version. However, it is significantly longer due to the non-optimized implementation of ECC algorithms in the Spongy Castle library.

Unlike in symmetric key encryption, asymmetric key encryption alone cannot be used to encrypt data. Thus, it needs to be coupled with a symmetric key algorithm to be used for encryption. In the present work, we used ECIES for public-key encryption. Moreover, mismatches of implementations in java. security and Spongy Castle create some inconsistencies in the EC calculations, such as computing the wrong public key for a given private key and EC. As a result, key pairs may not be able to decrypt the content. According to Martinez et al., it is impossible to implement a software version of ECIES that is compatible with all standards [40]. The biggest barrier here is the Certicom patent for ownership of ECC. Thus, usage and development of ECC have become limited. Although Certicom provides advice on using their curves and parameters to obtain safe curves as well as to facilitate interoperability, Daniel and Tanja [52] have shown that most of those recommended curves are not safe. E382, M383, and Curve383187 are reported to be some of the latest safe curves.

Further, real-world attackers have used the gap between ECDLP difficulty and ECC security to compromise the security. Most of the attackers have broken systems without solving the ECDLP. They have used incorrect data for some rare curve points, branching time, and cache time to break systems. ECC has not yet been tested as thoroughly as RSA has for its robustness. Researchers have already found sub-exponential algorithms for certain parameters of ECC. Thus, crypt-analysts should explore it thoroughly to find its vulnerabilities.

Besides, it has already been proved that both RSA and ECC can be cracked with the use of Quantum computers in the future when they reach the 1000 Q-bit level.

One of the limitations of our approach is that again the application itself is responsible for implementing defined access levels. Decryption happens within the application and the user's private key cannot be directly used to decrypt the symmetric key. Otherwise, once the symmetric key is obtained, the user can decrypt and use the content as s/he wishes. This can be avoided by implementing custom file structures for the content instead of using generic file structures. Besides, a suitable compression algorithm can be applied prior to the encryption to reduce the size of the content by eliminating the data redundancies. Another limitation is the inability to protect real-time videos due to the inefficiency of the AES.

5. Conclusions

In the present work, a combination of two robust cryptography methods has been used to implement a dynamic and flexible access control method that can ensure the access rights of sensitive medical data without compromising the strength of any. The method can be used to enforce different access rights for different users over the same content in a multicasting environment. It uses multiple keys effectively to enforce different access rights without making any extra key storage problem. The method is well suited to mobile devices, since ECC requires fewer computations and less space compared to other public-key methods.

REFERENCES

[1] Ryan Ausanka-Crues, "Methods for access control: Advances and limitations," *Harvey Mudd College*, vol. 301, 2001.

[2] Jason Crampton, "Cryptographic enforcement of role-based access control," *in Formal*.

[3] Anthony Harrington and Christian Jensen, "Cryptographic access control in a distributed file system," in *Proceedings of the Eighth*.

[4] Urs Hengartner and Peter Steenkiste, "Exploiting hierarchical identity-based encryption for access control to pervasive computing information," DTIC Document, Tech. rep. 2004.

[5] Anne V. D. M. Kayem and et al, "Enhancing identity trust in cryptographic key management systems for dynamic environments," *Security and Communication Networks,* vol. 4, no. 1, pp. 79-94, 2011.

[6] Anne VDM Kayem and et al., "A presentation of access control methods," *in Adaptive*.

[7] L. Xiaoqin and et al, "Application of the Advanced Encryption Standard and DM642 in the image transmission system," in *Computer Science Education (ICCSE), 2012 7th International Conference on,* July 2012, pp. 444-447.

[8] Bing Ji and et al, "New Version of AES-ECC Encryption System Based on FPGA in WSNs," *Journal of Software Engineering,* vol. 9, no. 1, pp. 87-95, 2015.

[9] Jignesh R. Patel and Rajesh S. Bansode, "Hybrid Security Algorithms for Data Transmission using AES-DES," *International Journal of Applied Information Systems (IJAIS),* vol. 2, no. 2, pp. 15-21, 2012.

[10] Xiyao Liu and et al, "A Secure Medical Information Management System for Wireless Body Area Networks," *KSII Transactions on Internet \& Information Systems,* vol. 10, no. 1, 2016.

[11] Challa Vamshi Krishna and et al, "Design Implementation of Composite Field S-Box using AES 256 Algorithm," *International Journal of Emerging Engineering Research and Technology (IJEERT),* vol. 3, no. 12, pp. 43-51, 2016.

[12] Vishal R. Pancholi and Bhadresh P. Patel, "Enhancement of Cloud Computing Security with Secure Data Storage using AES," *International Journal for Innovative Research in Science and Technology,* vol. 2, no. 9, pp. 18-21, 2016.

[13] Kundankumar Rameshwar Saraf and et al, "Text and image encryption decryption using advanced encryption standard," *IJETTCS,* vol. 3, no. 3, pp. 118-26, 2014.

[14] Nils Gura and et al, Marc Joye and Jean-Jacques Quisquater, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, ch. Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs, pp. 119-132.

[15] Rajesh P. Singh and et al, "Public Key Cryptography Using Permutation P-Polynomials over Finite Fields.," *IACR Cryptology ePrint Archive,* vol. 2009, p. 208, 2009.

[16] Haodong Wang and et al, "Comparing symmetric-key and public-key based security schemes in sensor networks: A case study of user access control," in *Distributed*.

[17] Davide Schipani and et al, "Efficient evaluation of polynomials over finite fields," *arXiv preprint arXiv:* 1102.4771, 2011.

[18] Payman Mohassel, "Fast computation on encrypted polynomials and applications," in *Cryptology and*

[19] Edgar R. Weippl and et al, "Content-based Management of Document Access Control," in

[20] Yu Fang Chung and et al, "Access control in user hierarchy based on elliptic curve cryptosystem," *Information Sciences*, vol. 178, no. 1, pp. 230-243, 2008.

[21] Nils Gura and et al, "An end-to-end systems approach to elliptic curve cryptography," in *Cryptographic*.

[22] Xuan Hung Le and et al, "An energy-efficient access control

scheme for wireless sensor networks based on elliptic curve cryptography," *Communications and Networks, Journal of*, vol. 11, no. 6, pp. 599-606, 2009.

[23] Santanu Chatterjee and Ashok Kumar Das, "An effective ECC based user access control scheme with attribute based encryption for wireless sensor networks," *Security and Communication Networks,* vol. 8, no. 9, pp. 1752-1771, 2015.

[24] Nigel P. Smart, "Elliptic Curves," in *Cryptography Made* Simple.: Springer, 2016, pp. 67-78.

[25] Steven D. Galbraith and Pierrick Gaudry, "Recent progress on the elliptic curve discrete logarithm problem," *Designs, Codes and Cryptography,* vol. 78, no. 1, pp. 51-72, 2016.

[26] Ali Soleymani and et al, "A Novel Public Key Image Encryption Based on Elliptic Curves over Prime Group Field," *Journal of Image and Graphics,* vol. 1, no. 1, 2013.

[27] Charles Edge and Daniel O' Donnell, "Introduction to Cryptography," in *Enterprise Mac Security.:* Springer, 2016, pp. 497-499.

[28] Daniel J. Bernstein and et al, "High-speed high-security signatures," *Journal of Cryptographic Engineering,* vol. 2, no. 2, pp. 77-89, 2012.

[29] Tibor Jager and et al, "Practical invalid curve attacks on TLS-ECDH," *Computer Security--ESORICS* 2015, pp. 407-425, 2015.

[30] C. A. O. Yang and et al, "One ECDH key agreement scheme with authentication based on ECDLP," *Journal of Chongqing University of Posts and Telecommunications (Natural Science Edition),* vol. 1, no. 24, 2012.

[31] Marc Joye, "Secure ElGamal-Type Cryptosystems Without Message Encoding," in *The New Codebreakers.:* Springer, 2016, pp. 470-478.

[32] Ziad E. Dawahdeh and et al, "Modified ElGamal Elliptic Curve Cryptosystem using Hexadecimal Representation," *Indian Journal of Science and Technology,* vol. 8, no. 15, July 2015.

[33] Eun-Hee Goo and Seung-Dae Lee, "Reconfigurable real number field elliptic curve cryptography to improve the security," *Journal of Computer Virology and Hacking Techniques,* vol. 11, no. 3, pp. 123-128, August 2015.

[34] D. Boruah and M. Saikia, "Implementation of ElGamal Elliptic Curve Cryptography over prime field using C," *in Information Communication and Embedded Systems (ICICES), 2014 International Conference on,* Feb 2014, pp. 1-7.

[35] Hermann Seuschek and et al, "A Cautionary Note: Side-Channel Leakage Implications of Deterministic Signature Schemes," *in Proceedings of the Third Workshop on Cryptography and Security in Computing Systems.* ACM, pp. 7-12, January 2016.

[36] Jeremy Dubeuf and et al, "ECDSA Passive Attacks, Leakage Sources, and Common Design Mistakes," *ACM Transactions on Design Automation of Electronic Systems (TODAES),* vol. 21, no. 2, p. 31, 2016.

[37] Don B. Johnson and Alfred J. Menezes, "Elliptic curve DSA (ECDSA): An enhanced DSA," *Proceedings of the 7th conference on USENIX Security Symposium,* vol. 7, 1998.

[38] Don Johnson and et al, "The elliptic curve digital signature algorithm (ECDSA)," *International Journal of Information Security,* vol. 1, no. 1, pp. 36-63, 2001.

[39] Akansha Singh and et al, "A Key Agreement Algorithm Based on ECDSA for Wireless Sensor Network," *in Proceedings of 3rd International Conference on Advanced Computing, Networking and Informatics,* pp. 143-149, 2016.

[40] Martinez GligorGayoso and et al, "A survey of the elliptic curve integrated encryption scheme," *Journal of Computer Science and Engineering,* vol. 2, no. 2, pp. 7-13, August 2010.

[41] Dominic Bucerzan and Crina Ra, "Image Processing with Android Steganography," in *Soft Computing Applications.*: Springer, 2016, pp. 27-36.

[42] Mike Godwin, "What Every Citizen Should Know About DRM," *Public Knowledge New America Foundation, Washington,* 2004.

[43] Renato Iannella, "Open digital rights management," in *World.*

[44] H. L. Jonker and et al, "Security aspects of DRM systems," in *25th.*.

[45] Radia Perlman and et al, "Privacy-preserving DRM," in *Proceedings of the 9th*.

[46] Shruti Sarma Hazarika, "Digital Rights Management: A Restrictive Rather than a Defensive Mechanism and the Survival of the 'Fair Use' Doctrine," *Available at SSRN 2180305,* 2012.

[47] Chris McIntosh, "Cyber-security: Who will provide protection?," *Computer Fraud & Security,* vol. 2015, no. 12, pp. 19-20, 2015.

[48] Manuel Egele and et al, "An Empirical Study of Cryptographic Misuse in Android Applications," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security,* New York, NY, USA, 2013, pp. 73-84. [Online]. http://doi.acm.org/10.1145/2508859.2516693.

[49] L. W. Paczkowski and et al, Trusted display and transmission of digital ticket documentation, #aug#~25 2015,

[50] M. Struik, Cryptographic method and apparatus, #may#~26 2015,

[51] Emilia K, "Fast elliptic curve cryptography in OpenSSL," in *Financial Cryptography and Data Security.:* Springer, 2011, pp. 27-39.

[52] Daniel J. Bernstein and Tanja Lange, "SafeCurves: Choosing safe curves for elliptic-curve cryptography," in *ShmooCon 2014,* 2014. [Online]. http://safecurves.cr.yp.to/