http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 5 (1): 204-229, 2006
ISSN 1812-5638
© 2006 Asian Network for Scientific Information

Elliptic Curve Cryptography over Binary Finite Field GF(2^m)

Kefa Rabah
Department of Physics, Eastern Mediterranean University,
Gazimagusa, North Cyprus, via Mersin 10, Turkey

Abstract: The rapid growth of information technology that has resulted in significant advances in cryptography to protect the integrity and confidentiality of data is astounding. Elliptic curve cryptography is gaining wide acceptance as an alternative to the conventional cryptosystems (DES, RSA, AES, etc.). Elliptic curve cryptosystems require less computational power, memory, communication bandwidth and network connectivity. Elliptic curve ciphers today are based in smartcards, Personal Digital Assistants (PDAs), pagers and mobile phones and can be easily implemented with processors clocked in single digits of MHz. This study describes the basic design principle of Elliptic Curve Crypto (ECC) protocols. The ECC processor is normally used to perform elliptic curve operations for: EC Diffie-Hellman, EC ElGamal and ECDSA protocols. As an example we will implement ECC defined over binary finite field GF(2^m).

Key words: Cryptography, data security, secure communication, wireless, ECC, ECDSA, DH, ElGamal, RSA, IPsec, TLS, openSSL

INTRODUCTION

In the modern information-oriented society, various devices are connected to the Internet as terminals, which necessitate technology for information security. Today, the world continues to witness an explosion of technology designed to help people communicate faster and more easily. We carry powerful digital computers in our pockets, exchange digital information in addition to voice data with our mobile phones and surf the Web with high-end PDAs. In the near future, especially the coming of age of 3G wireless devices, every type of electronic data channel will be used to exchange every type of electronic information. One of the great challenges of the ability to communicate digitally is securing the increased amount of electronic information now exchanged over the network.

Over the last three decades the traditional cryptosystems like DES, DLP, RSA, DSA etc. have thus far been the answer to the wide range of issues that impact modern communication, including the assurance of privacy, the certainty of the transmitter or receiver's identity and the integrity of the communication[1,2]. Today, these traditional crypto-algorithms which were once considered effective have become in impractical in light of recent technological development of constrained environment devices. These conventional crypto-tech approaches cannot support the new generation of digital communication and information access devices with their low power, small form factor and high performance requirements. The emerging breed of laptops, handhelds, mobile phones, PDAs and wireless devices require a next-generation crypto-security technology.

However, it is important to note that traditional cryptographic algorithms like DLP[3], RSA[4] etc. are not particularly efficient in small form factor, low-power, resource constrained devices, as they require memory intensive power hungry big integer computation co-processor to complete the calculations in a timely manner[1]. Adding such a co-processor significantly raises the cost of manufacture, rendering many devices impractical. The cost of producing a smartcard, for example, is increased by as much as 400% when an additional processor is required[5]. For embedded systems or telecommunications applications characterized by extremely high volumes and a wide variety of devices, many of which have limited computing resources, the trend has been towards alternate crypto-algorithms.

One technology in particular, called Elliptic Curve Cryptography (ECC), has become the cryptography of choice for mobile computing and communications devices due to its size and efficiency benefits. Elliptic curve cipher uses very small keys and is computationally very efficient, which makes it ideal for the smaller, less powerful devices being used today by majority of individuals to access network services. Its efficiency enables constrained, wireless devices to establish secure end-to-end connections. Hence, the server-side crypto-security networks have to be enabled to perform ECC operations for the next generation of wireless devices that use variable parameters in an efficient way to reduce systems cost. In Koblitz[6] and Miller[7], independently proposed the Elliptic Curve Cryptosystem (ECC)-a crypto-algorithm method of utilizing a Discrete Logarithm Problem (DLP) over the points on an elliptic curve. By their proposal, ECC could be used to provide both digital signatures and an encryption scheme. Over the past decade, ECC and later ECDLP (Elliptic Curve Discrete Logarithm Problem)[8,9] has received considerable attention from mathematicians around the world and no significant breakthroughs have been made in determining weaknesses in the algorithm and to-date has weathered umpteen mathematical attacks. Elliptic curve cryptosystems have thereby come to be accepted today as the most viable public-key technology for high-security applications. They are also most suitable for constrained environment devices such as those in which smartcards and personal wireless devices are typically deployed. Over the coming years, there will continue to be a great need for designing and implementing ECC enabled cryptosystems ranging from software, protocols and hardware solutions to secure cutting-edge technologies such as: smartcards, mobile phones, browsers, servers, Radio Frequency Identification (RFID) tags and environmental sensors.

The ECC provides higher strength-per-bit than any other current public-key cryptosystems[10]. If you compare elliptic curves to RSA and DLP you find many advantages: to obtain the same security one can use smaller fields for elliptic curves[11,12]. Therefore, elliptic curves can be implemented easier and faster. Furthermore, because of its higher strength-per-bit, ECCs are being increasingly used in practical applications in embedded systems (e.g., IC card and mobile devices) instead of RSA, which today is the most used public-key cryptosystems. The Elliptic Curve Crypto-schemes are also used for implementing protocols such as EC-digital signature scheme (ECDSA)[13], Diffie-Hellman key exchange scheme[14], EC ElGamal crypto-schemes[15] and so on. Following in the footsteps of DES[16]; ECC in conjunction with advance symmetric algorithm, AES[17], has already been incorporated into a number of key international standards, including ANSI X9.63, IEEE 1363-2000, IETF RFC 3278, ISO 15946-3 and NIST SP 800-56[18]. Further evidence of widespread acceptance are the inclusion of ECC in IPsec, TLS and OpenSSL[19,20]. Adoption into global standards will assist in pushing ECC into wider commercial usage. This integration in to the standard crypto-security systems will greatly enhance the overall usage of the next generation of wireless devices like 3G, which are extremely powerful but must still conserve their power consumption to achieve a longer working battery cycle. This is another advantage for using ECC which uses very small keys and is computationally very efficient which makes it ideal for the smaller, less powerful devices being used today by majority of individuals to access network services. With the commercial-grade implementations of ECC, developers should also expect to see an overall speed increase introduced by computational optimizations.

Today RSA is the powerhouse crypto-security of choice for e-commerce transactions, but its key size must double by the end of this decade to provide the same level security. At this point RSA crypto-technology become an extreme heavyweight for constrained environment devices (Fig. 1). Another factor going against the future of RSA is they are too slow compared to ECC, due to their dependence on computational involving two large prime integers, while the latter does not. The future of Internet security will also be greatly enhanced when they finally switch to elliptic curve based crypto-server security. With ECC smaller key-size requirements and enhanced computational efficiency, IT connectivity providers will be able to utilize fewer crypto-server security for providing secure network connections. Table 1 compares the equivalent security level for some commonly considered cryptographic key sizes.

As ECC becomes more important and more widely used, we foresee the following scenario. Commercial entities such as financial services or online stores running e-business will carry out transaction with users over the wireless conveniently and securely. The servers are required to perform crypto-operations such as signature generation and verification and key exchange. Most of these operations will employ standardized settings such as the NIST recommended elliptic curves[20]. However, there may be users that generate curves themselves, or Certificate Authorities (CA) that issue certificates over non-standardized curves that promise a performance gain. For example, they might want to use a curve defined over GF(2^{155}) as described by Schroeppel *et al.*[21], a curve over GF(2^{178}) as presented by Schroeppel *et al.*[22], or curves suited to special field element representations by De Win *et al.*[23]. There may also be standards which are rarely used and thus do not justify the implementation of a special case and future standards may introduce new curves. Still, a server must be able to handle all requests efficiently to reduce computing time and hence efficiency and cost.

It should be stressed, however, that there are also potential drawbacks associated with ECC from a practical perspective: RSA operations involving public keys (i.e., signature verification or encryption) can be performed with very short keys (Table 2). This leads to very fast operations which are usually faster than the corresponding ECC operations. However, RSA operations with private keys are usually much slower. In addition, the performance advantage for public key will decrease in future as longer bit length will become more common place for both RSA and ECC, (Fig. 1 and Table 1). In this study we analyze

Table 1: Comparison of the equivalent security level for some commonly used cryptographic key sizes

Time to break In MIPS years	RSA/DSA key size	ECC key size	RSA/ECC key size ratio
10^4	512	106	5:1
10^8	768	132	6:1
10^{11}	1.024	160	7:1
10^{20}	2.048	210	10:1
10^{78}	21.000	600	35:1

Table 2: Comparisons of ECC vs. RSA vs. DH*

	ECC (256 R1)	RSA (3072)	DH (3072)
Key generation	166 ms	N/A**	38s
Encrypt/Verify	150 ms	52 ms	74s
Decrypt/Sign	168 ms	8s	74s

*Timings were taken on BlackBerry 7230 at 128-bit security level. Timings at a 256-bit security level would show even greater differences between ECC, RSA and DH. **Time was too long to measure[22]

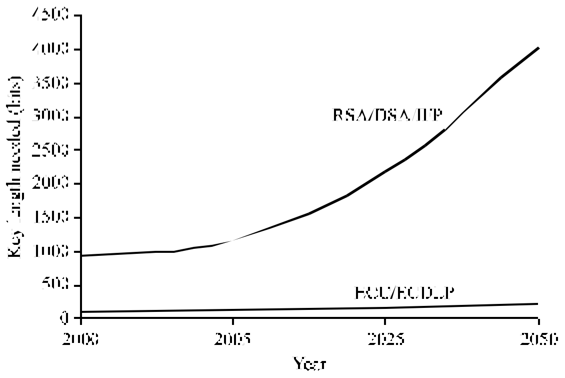

Fig. 1: Proposed the minimum key sizes (in bits) to be regarded as safe for RSA and ECC

the elliptic curve operations of these ECC protocols and design procedure of the Elliptic Curve Cryptographic defined over binary finite field GF(2^m).

GENERAL ELLIPTIC CURVES

Elliptic curve cryptology uses mathematical ideas such as: groups, Abelian groups, operations with groups, fields, operations with fields, elliptic curves and elliptic curve arithmetic using fields and groups. Let k be a field, $\overline{K}$ its algebraic closure and K* its multiplicative group. An elliptic curve over K will be defined as the set of solution in projective plane $\mathbb{P}^2(\overline{K})$ of a homogeneous Weierstrass equation of the form:

$$E: \; Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3 \qquad (1)$$

With $a_1, a_2, a_3, a_4, a_6 \in K$. This equation is referred to as long Weierstrass form. Such a curve should be non-supersingular in the sense that, if the equation is written in the form F(X, Y, Z) = 0, then the partial derivatives of the curve equation $\partial F/\partial X$, $\partial F/\partial Y$ and $\partial F/\partial Z$ should not vanish simultaneously at any point on the curve.

Let $\hat{K}$ be a field satisfying $K \subseteq \hat{K} \subseteq \overline{K}$. A point(X, Y, Z) on the curve is $\hat{K}$ -rational if (X, Y, Z) $= \alpha(\hat{X}, \hat{Y}, \hat{Z})$ for some $\alpha \in \overline{K}$, $(\hat{X}, \hat{Y}, \hat{Z}) \in \hat{K}^3 \setminus \{0,0,0\}$, i.e. up to projective equivalence, the coordinates of the point are in $\hat{K}$. The set of $\hat{K}$ -rational points on E is denoted by $(\hat{K})$. When the field of definition of the curve, K, is clear from the context, we will refer to $\hat{K}$ -rational points simply as rational points. The curve has exactly one rational point with coordinate Z equal to zero, namely (0,1,0). This is the point at infinity, which will be denoted by O.

For convenience, we will most often use the affine version of Weierstrass equation, given by:

$$E: \; Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6 \qquad (2)$$

Where, $a_i \in K$. The $\hat{K}$ -rational points in the affine case are the solutions to E in $\hat{K}^2$ and the point at infinity O. For curves over the reals, this point can be thought of as lying infinitely far up the y-axis[8]. We will switch freely between the projective and affine presentations of the curve, denoting the equation in both cases by E. For $Z \neq 0$, a projective point (X, Y, Z) satisfying Eq. 1 corresponds to the affine point (X/Z,Y/Z) satisfying Eq. 2.

Finite field selections: Key in implementation of ECC is selection of elliptic curve groups over the finite fields of the form GF(p) or $\mathbb{F}_p$ and GF(p^m) or $\mathbb{F}_{p^m}$ where, p is prime and m is a positive integer. By definition elliptic curve groups are additive groups. Various finite field can be generated using values of p, m and f(x), where $f(x) = x^m + \sum_{i=0}^{m-1} f_i x^i$, $f_i \in GF(p^m)$. Also using generic algorithm for different kind of field faster efficiency rate can be achieved.

Field over GF(p) is also known as integer modulo p, even, or odd prime[8]. Using GF(p) will require a coprocessor to perform the modular operation. Hence, obviously the performance with a GF(p) will be lower compared to that over binary field GF(2^m). Furthermore, addition of a coprocessor in small form factor devices like smartcard increases the cost by 20 to 40%. Hence, for practical applications finite field of the form $K = GF(2^m) = \mathbb{F}_{2^m}$ are very important. For such elliptic curves the theory as developed by Rabah[8] has to be modified.

ELLIPTIC CURVE OVER CHARACTERISTIC 2 (BINARY FINITE FIELD), GF(2^m)

We now specialize on finite field $\mathbb{F}_q$ where, $q = 2^m$, $m \geq 1$. Elements of the finite field $\mathbb{F}_{2^m}$ are binary of fixed length m. There are several ways to define arithmetic in this field, but the most common are polynomial representation. Up till now, no experiments have suggested a link between the field representation and the complexity of the resulting elliptic curves. So the choice of representation for the finite field does not appear to affect the security level of the elliptic curves defined over it.

Under these assumptions, a representative for each isomorphism class elliptic curves over $\mathbb{F}_q$ is given by:

$$E_{a_1, a_2 a_6} : Y^2 + a_1 XY = X^3 + a_2 X^2 + a_6 \qquad (3)$$

Where, $a_1 = 1, a_6 \in \mathbb{F}_q^*$ and $a_2 \in \{0, \gamma\}$ with γ a fixed element in $\mathbb{F}_q$ of trace $Tr_{q|2}(\gamma) = 1$, where, $Tr_{q|2}$ is the linear trace from $\mathbb{F}_q$ to $\mathbb{F}_2$. This function is not directly related to trace of Frobenius and no confusion should arise since they are used in quite different context. In this case, the expression for the j-invariant reduces to $j(E) = a_1^{12}/\Delta$. In characteristic two, the condition $j(E) = 0$, i.e., $a_1 = 0$, is equivalent to the curve being supersingular, which is a very special type of curve and is avoided in cryptography. We assume, therefore, that $j(E) \neq 0$.

The finite field $\mathbb{F}(2^m)$, called a binary field, can be viewed as a vector space of dimension m over $\mathbb{F}_2$. That is, there exists a set of m elements $\{e_0, e_1, \ldots e_{m-1}\}$ in $\mathbb{F}_{2^m}$ such that each $a \in \mathbb{F}_{2^m}$ can be written uniquely in the form $a = \sum_{i=0}^{m-1} a_i e_i$, where, $a_i \in \mathbb{F}_2$, i.e., $a_i = 0$ or 1. The set $\{e_0, e_1, \ldots e_{m-1}\}$ is called a basis of $\mathbb{F}(2^m)$ over $\mathbb{F}_2$. We can then represent a as a binary vector $a = (a_0, a_1, \ldots a_{m-1})$. There are many different basis of $\mathbb{F}_{2^m}$. The most natural basis are, of course, polynomial basis, normal basis and optimal normal basis. In polynomial basis, we just have $e_i = t^i$ where, t is he fixed complex (or whatever) root of our reduction trinomial : $t^m + t^k + 1$. (Note that it is possible to chose another basis, called a normal basis, of the form: $\{\beta, \beta^2, \beta^{22}, \ldots, \beta^{2\ m-1}\}$ where, $\beta \in \mathbb{F}_{2^m}$. It is well known that such a basis always exists). The most natural basis are, of course, polynomial basis, normal basis and optimal normal basis (Table 3).

Optimal extension finite field: Field of $\mathbb{F}(p^m)$ uses p = 2 such that it gives implementer f(x) of form trinomial or polynomial. An alternative representation of $\mathbb{F}(p^m)$ is the Optimal Extension Fields (OEFs). OEFs is an alternative construction to the $\mathbb{F}(p^m)$ where, $p = 2^n \pm c$, where c is arbitrary positive rational number $\log_2 c = n/2$. Also p is prime less than but close to the word size of the processor. In addition, m is chosen such that, $f(x) = x^m - w$, is an irreducible. The goal is to select parameters which provide adequate security without incurring excessive computation time.

Lenstra and Verheul[24] showed that under certain assumptions, 925-bit RSA and DSA systems may be considered equivalent in security to a 132-bit ECC system. The field GF($(2^8-17)^{17}$), for example, a security level of 134-bits which is far more secure than 512-bit RSA system which has been popular for smartcard applications. Optimal Extension Fields (OEF) are computationally more efficient than other field types offering roughly the same level of security[25]. Table 4 displays three fields of similar field order, which implies a similar security strength of the cryptosystem based on these fields[26]. The third column denotes the number of cycles for one field multiplication which is the crucial operation determing the efficiency of the system. One can see that the OEF displayed in the third row performs more efficient than the binary field displayed in the first row.

Composite extension finite fields and subfields: When m is a composite number, m = rs, then the composite extension finite field GF(2^m) can be considered an extension field of degree s over finite field GF(2^r). The finite field GF(2^r) is called a subfield of GF(2^m). The elliptic curve over a subfield is also used in computing the order of an elliptic curve over a composite extension finite field using Hasse-Weil's theorem[8]. We can also represent elements in a composite extension finite field over its subfield using either one of the two basis discussed earlier.

We should point out that the finite field GF($(2^r)^s$) is isomorphic to the finite field GF(2^m), but their field operations (additions and multiplications) are different depending on the irreducible field polynomials, f(x) of GF($(2^r)^s$) over GF(2^r) and GF(2^r) over GF(2). It also depends on the possible factorizations of m (other than factors r and s). The choice of those field polynomials are essential to determine the algorithmic complexity of arithmetic operation of GF($(2^r)^s$). However, a recent attack on ECCs over composite fields makes them inappropriate for use in practice[25].

Table 3: Comparisons of different finite fields for implementing ECC

Finite fields					
Prime finite fields $\mathbb{F}_p, p > 2$			Extension finite fields $\mathbb{F}(p^m)$		
General primes (and mersenne)	Special form of primes		Char(p)=2		
	Pseudo mersenne	Generalized mersenne	Binary	Composite	char(p)>2 OEF
$\mathbb{F}_p$	$\mathbb{F}(2^n - c)$	$\mathbb{F}(2^n - 2^s - 1)$	$\mathbb{F}(2^n)$	$\mathbb{F}((2^n)^m)$	$\mathbb{F}((2^n \pm c)^m)$

Table 4: ECC field performance

Field	Field order	No. cycles for multiply
$GF(2^{135})$	2^{135}	19.600
$GF(2^8)^{17}$	2^{136}	7.479
$GF(2^8\text{-}17)^{17}$	2^{134}	5.084

ARITHMETIC OVER FINITE FIELD GF(2^m) USING POLYNOMIAL BASIS EPRESENTATION

A good understanding of the complexity of the mathematics in the field $\mathbb{F}_{2^m}$ is necessary to appreciate the difficulty of solving equations of elliptic curves defined over this field[8,9]. The elements of the finite field $\mathbb{F}_{2^m}$ are binary strings of fixed length m. As indicated above there are many ways to construct a finite field with a prime power number elements, but the most common are polynomial representation, normal and optimal normal representation.

The polynomial basis representation: Polynomial basis representation can be used to implement addition, multiplication, division and remainder or modulus operation. This being the case, implementation presented in this section, uses field polynomial representation because of its relatively easy arithmetic operations.

Let $f(x) = x^m + f_{m-1}x^{m-1} + f_{m-2}x^{m-2} + \ldots + f_2x^2 + f_1x + f_0$, where, each $f_i \in \mathbb{F}_2$. We call f(x) the reduction polynomial (or sometimes, field polynomial). The polynomial f(x) must be irreducible; that is, it cannot be factored into two polynomials over $\mathbb{F}_2$, each of degree less than m. In polynomial basis form, the elements of $\mathbb{F}_{2^m}$ are treated as polynomials of degree m, with the individual bits representing coefficients in $\mathbb{F}_2$ of each term. So elements are of the form:

$$\mathbb{F}_{2^m} \equiv g = \left\{ \begin{matrix} a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \ldots + \\ a_2x^2 + a_1x + a_0 \mid a_i = \{0,1\} \end{matrix} \right\} \qquad (4)$$

These elements $(a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \ldots + a_2x^2 + a_1x + a_0)$ can be written in binary strings or vector form as $(a_{m-1} \ldots a_1, a_0)$ of length m. $\mathbb{F}_{2^m}$ has 2^m elements, i.e., $\mathbb{F}_{2^m} \equiv g = \{(a_{m-1} \cdots a_1 a_0) : a_i = \{0,1\}\}$. Particularly, we can write the zero element 0 = (0, 0,… 0) and the multiplicative identity 1 = (0, 0, … 0, 1).

Let $\mathbb{F}^*_{2^m}$ denote the set of all non-zero elements in $\mathbb{F}_{2^m}$. The field $\mathbb{F}^*_{2^m} \equiv G$ is cyclic, which means: $G = \{g^0, g^1, \ldots, g^n\}$ where, $g^n = I$, with $n = 2^m - 1$ and I is the identity element. Here $g \in G$, where, $g \neq 0$, is called a generator (or primitive) element. By definition, this means that any group element can be expressed as a power of g. The multiplicative inverse of an element: $a = g^i$ is $a^{-1} = g^{(-i)\bmod(2^m-1)}$. This method of representing $\mathbb{F}_{2^m}$ is called a polynomial basis representation.

Since $\mathbb{F}_{2^m}$ is a mathematical 'field', it has the binary operations of addition and multiplication. Table 5 shows bit-to-binary-to-polynomial representation and their decimal equivalence. Note that the presence of the coefficient is marked by a digit 1 and the absence is marked by a digit zero, 0, e.g $g = \{a_3x^3 + a_2x^2 + 1\}$, will be written as: $(a_3a_2a_1a_0) = (1101)$.

Irreducible polynomials: A polynomial f(x) is said to be irreducible if we cannot write, f(x) = h(x).u(x), for any polynomials h(x), u(x) of degree strictly less than the degree of f(x). An irreducible polynomial $f(x) = x^m + f_{m-1}x^{m-1} + \ldots + f_1x + f_0$ of degree m over $\mathbb{F}_2$ should satisfy these necessary conditions:

- The constant term $f_0 = 1$; otherwise, we can factor x out. Hence, from now on we always write the general form as: $f(x) = x^m + f_{m-1}x^{m-1} + \ldots + f_1x + 1$.
- There is an odd number (≥3) of nonzero terms; otherwise, f(x) whose number of nonzero terms is even has a factor (x+1).
- There must be at least one term of odd degree; otherwise, f(x) of all even powers is a square of a polynomial of degree (m/2).

Table 5: Implement of binary-to-polynomial basis representation

Digit No.	8	7	6	5	4	3	2	1	0
Digit	1	1	1	1	1	1	1	1	1
Binary	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Polynomial	a_8x^8	a_7x^7	a_6x^6	a_5x^5	a_4x^4	a_3x^3	a_2x^2	a_1x^1	a_0x^0
Decimal	256	128	64	32	16	8	4	2	1

- It is easy to verify this property: If $f(x) = x^m + f_{m-1}x^{m-1} + \ldots + f_1x + 1$, is an irreducible polynomial of degree m, then so are polynomials h(x) = f(x+1) and $u(x) = x^m f(1/x) = x^m + f_1x^{m-1} + \ldots + f_{m-1}x + 1$.
- The compositions of h(x) and u(x) will give us a few more irreducible polynomials.

Primitive polynomials: If the period of f(x) is (2^m-1), that is the order of the multiplicative subgroup $\mathbb{F}^*_{2^m} = \mathbb{F}_{2^m} \setminus \{0\}$, then f(x) is called a primitive polynomial. For example, if $f(x) = x^m + f_{m-1}x^{m-1} + \ldots + f_1x + f_0$ is an irreducible polynomial of degree m over $\mathbb{F}_2$ and r is a root of f(x) in an extension field of $\mathbb{F}_2$ (that is a finite field $\mathbb{F}_{2^m}$), then $r, r^2, r^{2^2}, \ldots, r^{2^{m-1}}$ are all roots of f(x). Indeed, if f(r) = 0, then for any d, we have $f_i^{2^d} = f_i$ since f_i equals to 0 or 1. Hence:

$$\begin{aligned} f(r^{2^d}) &= f_{m-1}r^{2^d \cdot (m-1)} + \ldots + f_1r^{2^d} + f_0 \\ &= (f_{m-1}r^{m-1} + \ldots + f_1r + f_0)^{2^d} = 0 \end{aligned}$$

Note that all the roots of the polynomial f(x) have the same multiplicative order, that is called the period (or order) of the function f(x).

All roots of a primitive polynomial f(x) are primitive elements of $\mathbb{F}_{2^m}$. For example, the polynomial u(x) = $x^m + \ldots + x + 1$ divides $(x^{m+1}+1)$. Hence, its period is equal to (m+1) or less and u(x) is not a primitive polynomial of $\mathbb{F}_{2^m}$, unless m = 2.

A primitive polynomial can be built from a primitive element a of a finite field $\mathbb{F}_{2^m}$ by the formula:

$$f(x) = (x + a)(x + a^2)(x + a^{2^2}) \ldots (x + a^{2^{m-1}})$$

If another primitive polynomial

$$h(x) = (x + b)(x + b^2)(x + b^{2^2}) \ldots (x + b^{2^{m-1}})$$

is built from a primitive element b, such that $b \notin \{a^{2^n} \mid 0 \le n \le m-1\}$, then $h(x) \ne f(x)$.

In other words, each primitive element is a root of only one primitive polynomial of $\mathbb{F}_{2^m}$. In fact, the set of roots of all primitive polynomials for a finite field are exactly all primitive elements in $\mathbb{F}^*_{2^m} = \mathbb{F}_{2^m} \setminus \{0\}$. Hence, there can be many primitive polynomials for a finite field $\mathbb{F}_{2^m}$, when m≥3. And in fact, a primitive polynomial cannot be reducible over $\mathbb{F}_2$.

Rules for selecting a reduction polynomial

- A trinomial over $\mathbb{F}_2$ is a polynomial of the form: $T_{m,k}(x) = x^m+x^k+1$, where $1 \le k \le m-1$. In fact, a trinomial: $T_{m,k}(x) = x^m+x^k+1$ is irreducible if and only if its reciprocal trinomial $T_{m,m-k}(x) = x^m+x^{m-k}+1$ is irreducible. Hence, we should be interested in trinomials of the following form only: $T_{m,k}(x) = x^m+x^k+1$, where, $1 \le k \le m/2$.
- A pentanomial over $\mathbb{F}_2$ is a polynomial of the form: $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$, where $1 \le k_1 < k_2 < k_3 \le m-1$. Such polynomials always exists for $m \ge 4$. In practice, it is recommended to use pentanomials whose coefficient triples (k_1, k_2, k_3) or (k_3, k_2, k_1) will have the first coefficient as small as possible and next coefficients are kept as small as possible after fixing the previous one or ones in the triple order. These polynomials would have more efficient computations of finite field operations.

ANSI X9.62 specifies the following rules for selecting the reduction polynomial for representing the elements of $\mathbb{F}_{2^m}$ [13]:

- If there exists an irreducible trinomial of degree m over $\mathbb{F}_2$, then the reduction polynomial f(x) must be an irreducible trinomial of degree m over $\mathbb{F}_2$. To maximize the chances for interoperability, ANSI X9.62 recommends that the trinomial used should be, x^m+x^k+1, for the smallest possible k.
- If there does not exist an irreducible trinomial of degree m over $\mathbb{F}_2$, then the reduction polynomial f(x) must be an irreducible pentanomial of degree m over $\mathbb{F}_2$. To maximize the chances for interoperability, ANSI X9.62 recommends that the pentanomial used should be: $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$, chosen according to the following criteria: (i) k_3 is as small as possible; (ii) for this particular value of k_3 and k_2 is a small as possible and (iii) for these particular values of k_3, k_2 and k_1 is as small as possible.

Field addition and multiplication over binary finite field GF(2^m): Field elements are added and multiplied as follows:

Field addition (+): Is performed component-wise by XORing (XOR symbol: $\oplus$, but here for simplicity we will use + instead). Addition in $\mathbb{F}_{2^m}$ is $(a_{m-1}\ldots a_1\, a_0)+(b_{m-1}\ldots b_1\, b_0)=(c_{m-1}\ldots c_1\, c_0)$ where each $c=a_i+b_i$ over $\mathbb{F}_2$. That is, addition is just the componentwise XOR. In the field $\mathbb{F}_{2^m}$, each element $(a_{m-1}\ldots a_1 a_0)$ is its own additive inverse, since $(a_{m-1}\ldots a_1 a_0)+\ (a_{m-1}\ldots a_1 a_0)=(0\ldots 00)$, the additive identity. Thus, addition and subtraction are equivalent operations in $\mathbb{F}_{2^m}$. Since $g\oplus g=0$ for all binary elements, then g+g = 0 = g-g. So we note that addition and subtraction are equivalent in this field, Fig. 2a on how to perform XOR operation. In hardware, a bit-parallel adder requires m-tuples as in $a=(a_{m-1}, a_{m-2},\ldots a_0)$. In hardware, a bit-parallel adder requires m XOR gates and addition can be generally computed in one clock cycles.

Field multiplication (·): Multiplication of field elements in $\mathbb{F}_{2^m}$ uses the same shift-and-add algorithm as is used for multiplication of integers, except that the "add" is replaced with "XOR". This has the virtue that the operation can no longer generate carriers, simplifying the implementation. That is, $(a_{m-1}\ldots a_1\, a_0)\cdot(b_{m-1}\ldots b_1\, b_0)=(r_{m-1}\ldots r_1\, r_0)$ where, the polynomial $(r_{m-1}x^{m-1}+\ldots+r_1x+r_0)$ is the remainder when the product polynomial $(a_{m-1}\,x^{m-1}+\ldots+a_1x+a_0)\cdot(b_{m-1}\,x^{m-1}+\ldots+b_1x+b_0)$ is divided by the polynomial f(x) over $\mathbb{F}_2$, where, the operation (·) is carried out Fig. 2b. (Note that all polynomial coefficients are reduced modulo 2.)

Field exponentiation: The exponentiation $(a_{m-1}\ldots a_1 a_0)^e$ is performed by multiplying together e copies of $(a_{m-1}\ldots a_1 a_0)$.

Multiplicative inversion: The rules for doubling an elliptic curve point and for adding two elliptic curve points, involve computing reciprocal, either 1/x or $1/(x_1+x_2)$. Multiplicative inversion of elements in a field is usually so slow that people have gone to great lengths to avoid it. Menezes *et al.*[5] and Beth and Schaefer[27] discuss projective schemes, which use about nine multiplications per elliptic curve step , but use very few reciprocals. An efficient algorithm for computing an inverse of an element in $\mathbb{F}_{2^m}$ was proposed by Itoh *et al.*[28].

$\oplus$	0	1
0	0	1
1	1	0

(a)

(.)	0	1
0	0	0
1	0	1

(b)

Fig. 2: (a) Addition rule and (b) Multiplication rule under finite field, $\mathbb{F}_2$

Rule to perform multiplicative inverse: We need to discuss also some methods of computing the inverse of a non-zero element. This operation obviously has an important role in field arithmetic, these are:

- The general method is using this identity: $a^{-1}=a^{2^m-2}=(a^{2^{m-1}-1})^2, \forall a\neq 0$. Recall that: $a^{2^{m-1}-1}=1$. In implementations, we can even analyze further the power exponent (2^{m-1}-1) of a to reduce our computation to a few multiplications. For m even or odd, we have:

  Odd: $2^{m-1}-1=(2^{(m-1)/2}\text{-}1)\cdot(2^{(m-1)/2}+1)$ and Even: $2^{m-1}-1=2\cdot(2^{(m-2)/2}-1)\cdot(2^{(m-2)/2}+1)$

  These formulae yield an algorithm for computing inverse by factorization of the exponent. Consider the field, GF(2^{155}):

$$2^{155}-1=2\cdot(2^{77}-1)\cdot(2^{77}+1)$$
$$2^{77}-1=2\cdot(2^{38}-1)\cdot(2^{28}+1)+1$$
$$2^{38}-1=2\cdot(2^{19}-1)\cdot(2^{19}+1)$$
$$2^{19}-1=2\cdot(2^{9}-1)\cdot(2^{9}+1)+1$$
$$2^{9}-1=2\cdot(2^{4}-1)\cdot(2^{4}+1)+1$$
$$2^{4}-1=2\cdot(2^{2}-1)\cdot(2^{2}+1)$$
$$2^{2}-1=2\cdot(2^{1}-1)\cdot(2^{1}+1)$$

  It requires 10 multiplications to compute an inverse in GF(2^{155}). In general, the method requires: $[\log_2^{(m-1)}+H^{(m-1)}-1]$ field multiplications.

- Alternatively, using the extended Euclidean algorithm: finding polynomials u(x) and v(x) such that ged(f(x), a(x)) = f(x)·u(x)+a(x)·v(x). When ged(f(x), a(x)) = 1, we can write 1 $\equiv$ a(x)·v(x)(mod f(x)). In other words, the polynomial v(x) is the inverse of a(x), modulo f(x).

Squaring: In particular, the squaring operation of a polynomial $(a_{m-1}x^{m-1}+\ldots+a_1x+a_0)$ that is performed in modulo 2 is in fact a linear operation, i.e.,:

$$(a_{m-1}x^{m-1}+\ldots+a_1x+a_0)^2 = a_{m-1}x^{2(m-1)}+a_m x^{2(m-2)}+\ldots+a_1x^2+a_0 \qquad (5)$$

In terms of bit strings, we write: $(a_{m-1}\ldots a_1a_0) = (a_{2(m-1)}, 0, a_{2(m-2)}, 0, \ldots 0, a_1, 0, a_0)$. Then we reduce the resulting polynomial by modulo f(x). In hardware, a bit-parallel realization of this squarer requires at most (r-1)(m-1) gates[29,30], where, r represents the number of non-zero coefficients of the polynomial when implemented using Elliptic Curve Processor (ECP) architecture (Fig. 3 and 4) field arithmetic module. The components are the Main Controller (MC), the Arithmetic Unit Controller (AUC) and Arithmetic Unit (AU). The MC is the ECP's main controller which performs the computation of the scalar multiplication, kP.

The AUC controls the AU and performs the computation of point additions, point doublings and coordinate conversions. The AU also performs $GF(2^m)$ field additions, squares, multiplications and inverses under AUC control[31].

Orlando and Paar[31] proposed a design for the standard basis field representation and it is based on the transformation from squaring operation into an addition and a multiplication by a constant that depends only on the field polynomial. For example, squaring a polynomial in a modulo 2 field is a linear operations. In the formula for squaring a binomial, $(a+b)^2 = a^2+2ab+b^2$, the cross-term vanishes modulo 2 and the square reduces to a^2+b^2. Consequently, we can square a sum by squaring the individual terms. For example, $(u^3+u+1)^2 = u^6+u^2+1$.

In terms of bitstrings, to square a polynomial, we spread it out by interleaving a 0 bit between each polynomial bit. For example, x^3+x+1 is represented as **1011** and the square $(x^3)^2+x^2+1^2 = x^6+x^2+1$ is represented by **1000101** (Table 5). Sadly, computer manufacturers have largely ignored the need for an instruction to carry out this operation; nonetheless, it can be done quickly using table lookup to convert each byte to its 15-bit square. The squared polynomial is then reduced modulo f(x). Squaring is so much faster than regular multiplication that it can be ignored in rough comparisons of the timings.

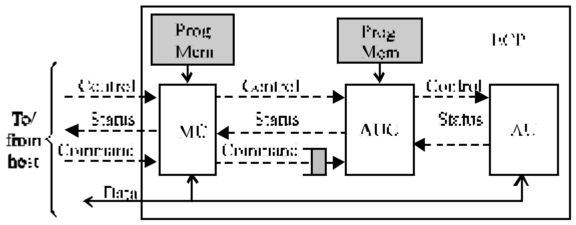

Fig. 3: Elliptic Curve Processor (ECP) architecture

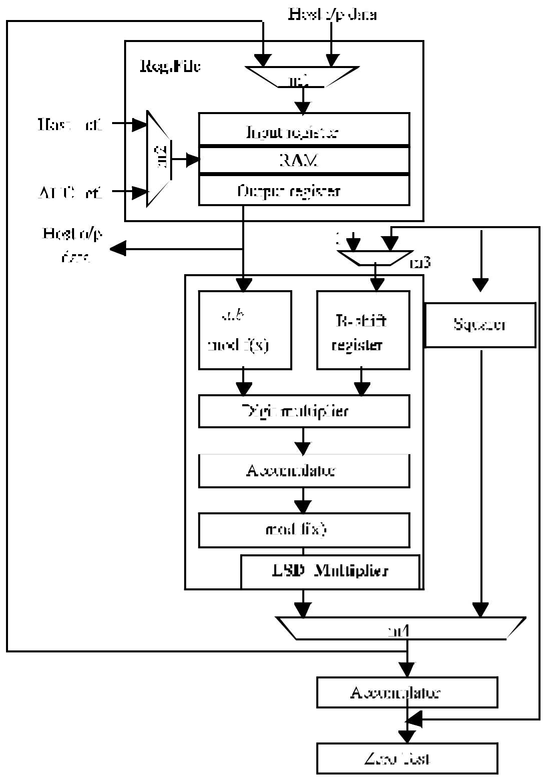

Fig. 4: Elliptic curve Arithmetic Unit (AU)

THE ELLIPTIC CURVE OVER FINITE FIELD GF(2^m)

Elements of the field $\mathbb{F}_{2^m}$ are m-bit string. The rules in $\mathbb{F}_{2^m}$ can be defined by either polynomial representation and by normal or optimal normal basis representation as started earlier. Let $\mathbb{F}_{2^m}$ be a characteristic 2 finite field and let $a, b \in \mathbb{F}_{2^m}$ satisfy $b \neq 0$ in $\mathbb{F}_{2^m}$. Then an elliptic curve $E(\mathbb{F}_{2^m})$ over $\mathbb{F}_{2^m}$ defined by the parameters $a, b \in \mathbb{F}_{2^m}$ consists of the set of solutions or points P = (x, y) for $x, y \in \mathbb{F}_{2^m}$ to the equation:

$$E: y^2 + xy = x^3 + ax^2 + b \text{ in } \mathbb{F}_{2^m} \quad (6)$$

together with an extra point O called the point at infinity. (Here the only elliptic curves over $\mathbb{F}_{2^m}$ of interest are non-supersingular elliptic curves[27].) Since $\mathbb{F}_{2^m}$ operates on the bit strings, computers can perform arithmetic in this field very efficiently. The number of points on $E(\mathbb{F}_{2^m})$ is denoted by $\#E(\mathbb{F}_{2^m})$.

Counting points on elliptic curves: A user of ECC will be interested to know the structure of the group, especially he would like to know the number of the points, $\#E(\mathbb{F}_q)$. If he knows $\#E(\mathbb{F}_q)$ he knows whether the Pohlig -Hellman-attack can be applied. It is impossible to write down all the points for large q. But it is not necessary at all. The formula of Hesse-Weil can be applied, which reduces the amount of work enormously, which states that if E is an elliptic curve over $\mathbb{F}_q$ [8,32], then, then:

$$q+1-2\sqrt{q} \leq \#(E(\mathbb{F}_q)) \leq q+1+2\sqrt{q} \qquad (7)$$

$\#E(\mathbb{F}_q) = q+1-t$ such that $|t| = 2\sqrt{q}$ is called the trace of the curve.

Note that $E(\mathbb{F}_q)$ (where, $q = 2^m$) is called the group of rational points i.e., points that are rational over the ground field $\mathbb{F}_q$. Not to be confused with E(Q)! the knowledge of the rational points on the curve is essential to cryptosystems because we often desire curves E with a large prime dividing $\#E(\mathbb{F}_q)$. So we can either construct curves with the right properties (like $\#E(\mathbb{F}_q)$) containing a large prime) or generate curves randomly and check if they have the desired properties. Hence, efficient computation of $\#E(\mathbb{F}_q)$ is an important question. Both approaches are useful depending on the application. For example EC factoring algorithms search for curves where the number to be factored, decomposes smoothly.

The order n of a point $P \neq O$ on an elliptic curve is a positive integer such that $nP = O$ and $mP \neq O$ for any integer m such that $1 \leq m < n$. The order n of a point must divide the order N of the elliptic curve. In fact, it is true for any group. If the elliptic curve order $N = \#E(\mathbb{F}_q)$ is a prime number, then the group is cyclic and obviously all points except the point at infinity O are of order N.

Elliptic curve arithmetic over finite field $\mathbb{F}_{2^m}$: It is again as is the case with elliptic curve over $\mathbb{F}_p$ [8], there is a chord-and-tangent rule for adding point on an elliptic curve $E(\mathbb{F}_{2^m})$ to give a third elliptic point. Together with this addition operation, the set of points on $E(\mathbb{F}_{2^m})$ forms a group with O serving as its identity.

The algebraic formula for the addition of points and the double of a point are specified as follows:

- Rule to add the point at infinity to itself: O+O=O
- Rule to add the point at infinity to any other point:

$$(x, y)+O = O+(x, y) = (x, y) \text{ for all } (x, y) \in E(\mathbb{F}_{2^m})$$

- Rule to add two points with the same x-coordinates when the points are either distinct or have x-coordinate 0: (x, y)+(x, x+y)=O for all $(x, y) \in E(\mathbb{F}_{2^m})$. That is, the negative of the point (x, y) is -(x, y) = (x, x+y)
- Rule to add a point to itself (double a point): Let $(P_1 = (x_1, y_1) \in E(\mathbb{F}_{2^m})$ and $P_2 = (x_2, y_2) \in E(\mathbb{F}_{2^m})$ be two points such that $x_1 \neq 0$. Then $P_1+P_2 = (x_1, y_1)+(x_2, y_2) = (x_3, y_3) = P_3 \neq O$, where:

$$x_3 = \lambda^2 + \lambda + a \text{ in } \mathbb{F}_{2^m}, y_3 = x_1^2 + (\lambda+1)x_3 \text{ in } \mathbb{F}_{2^m}$$

$$\text{and } \lambda = \frac{x_1^2 + y_1}{x_1} \text{ in } \mathbb{F}_{2^m} \qquad (8)$$

- Rule to add two points with different x-coordinates: Again let $P_1 \in E(\mathbb{F}_{2^m})$ and $P_2 \in E(\mathbb{F}_{2^m})$ be two points such that $x_1 \neq x_2$. Then: $P_1+P_2 = (x_3, y_3) = P_3 \neq O$, where:

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a \text{ in } \mathbb{F}_{2^m},$$
$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1 \text{ in } \mathbb{F}_{2^m}$$
$$\text{and } \lambda = \frac{y_1 + y_2}{x_1 + x_2} \text{ in } \mathbb{F}_{2^m} \qquad (9)$$

In either case, when P_1+P_2 (doubling) and $P_1 \neq P_2$ (point addition), major operations are field multiplication and field inversion. (Squaring and field addition are enough ignorable because of its less computation time.) From these formulas of point addition or doubling, we can determine the number of field operations required for each kind of elliptic curve operation. We see that an addition step usually requires eight additions, two multiplications, one squaring, three reductions mod f(x) and one inversion. A Doubling step usually requires four additions, two multiplications, two squaring, four reductions mod f(x) and one inversion. A Negation step requires one addition. The important contributors to the run time are the multiplications and inversions[28,33]. Just as modular exponentiation determines the efficiency of RSA cryptographic systems [1,4], scalar multiplication dominates the execution time of ECC systems. In all the protocols that are fundamental implementation of ECC, say ECDH, ECDSA, ECAES etc. the most time consuming part of the computations are scalar multiplications[33]. Elliptic curves have some properties that allow optimization of scalar multiplications. Thus, far the efficiency of the finite field arithmetic, especially multiplication, determines the overall efficiency of the elliptic curve cryptosystem.

THE FINITE FIELD GF(2^4) USING POLYNOMIAL BASIS REPRESENTATION

Let us consider the elliptic curve defined over $\mathbb{F}_{2^4}$ which from Eq. 6, we have $a = g^4$ and $b = g^0 = 1$ is given by:

Table 6: 16 elements of $\mathbb{F}_{2^4}$ obtained using the primitive irreducible Polynomial: $f(x) = x^4+x+1$

0 (0000)	1 (0001)	x (0010)	x+1 (0011)
x^2 (0100)	x^2+1 (0101)	x^2+x (0110)	x^2+x+1 (0111)
x^3 (1000)	x^3+1 (1001)	x^3+x (1010)	x^3+x+1 (1011)
x^3+x^2 (1100)	x^3+x^2+1 (1101)	x^3+x^2+1 (1110)	x^3+x^2+x+1 (1111)

Table 7: Powers of g using the element g = (0010) as the generator, including the elements of $\mathbb{F}_{2^4}$ using trinomial basis: $\{1, g, g^2, g^3\}$

$g^0 = (0001)$	$g^1 = (0010)$	$g^2 = (0100)$	$g^3 = (1000)$	$g^4 = (0011) = g+1$	$g^5 = (0110) = g^2+g$
$g^6 = (1100) = g^3+g^2$	$g^7 = (1011) = g^3+g+1$	$g^8 = (0101) = g^2+1$	$g^9 = (1010) = g^3+g$	$g^{10} = (0111) = g^2+g+1$	$g^{11} = (1110) = g^3+g^2+g$
$g^{12} = (1111) = g^3+g^2+g+1$	$g^{13} = (1101) = g^3+g^2+1$	$g^{14} = (1001) = g^3+1$	$g^{15} = (0001) = g^0$		

$$E: y^2 + xy = x^3 + g^4x^2 + 1 \qquad (10)$$

Note that $b \neq 0$, so E is indeed an elliptic curve. $\mathbb{F}_{2^4}$ is constructed using the primitive irreducible polynomial $f(x) = x^4+x+1$ and a root g. The set of points on $E(\mathbb{F}_{2^m})$ forms an abelian group under this addition rule. Notice that the addition rule can always be computed efficiently using simple field arithmetic. The element g = x = (0010) (i.e., g = x(mod f(x)), is a root of f(x) in $\mathbb{F}_{2^4}$) is a generator of $\mathbb{F}^*_{2^4}$ since the number of elements $E(\mathbb{F}_{2^4})^* = 2^4 - 1 = 15$ and it is cyclic, the powers of g obtained are (Table 6).

Note that all the 15 non-trivial powers of g are primitive elements of the multiplicative subgroup $\mathbb{F}^*_{2^4}$. The following are sample calculations to demonstrate the property of $\mathbb{F}_{2^4}$:

Addition: (0110)+(0101) = (0011).

Multiplication:

$$\begin{aligned}(1101)\cdot(1001) &= (x^3+x^2+1)\cdot(x^3+1) \bmod f(x)\\ &= x^6+x^5+2x^3+x^2+1 \bmod f(x)\\ &= x^6+x^5+x^2+1 \bmod f(x)\\ &\quad \text{(coefficients are reduced modulo 2)}\\ &= (x^4+x+1)\cdot(x^2+x)+(x^3+x^2+x+1) \bmod f(x)\\ &= x^3+x^2+x+1 = (1111)\end{aligned}$$

Exponentiation: To compute $(0010)^5$, first find $(0010)^2$: $\Rightarrow$ (0010)(0010)$\Rightarrow$ xx(mod f(x)) $\Rightarrow$ $(x^4+x+1)(0)+(x^2)$ modf(x) $\Rightarrow x^2 \Rightarrow$ (0100)

Then: $(0010)^4 \Rightarrow (0010)^2(0010)^2 \Rightarrow$ (0100)(0100) $\Rightarrow x^2x^2$ mod f(x) $\Rightarrow (x^4+x+1)(1)+(x+1)$ mod f(x) $\Rightarrow$ (x+1) $\Rightarrow$ (0011) $\Rightarrow$ (x+1) $\Rightarrow$ (0011)

Finally: $(0010)^5 \Rightarrow (0010)^4\cdot(0010) \Rightarrow (0011)\cdot(0010) \Rightarrow$ (x+1), (x^2+x) mod f(x) $\Rightarrow (x^4+x+1)(0) + (x^2+x)$ mod f(x) $\Rightarrow x^2+x \Rightarrow$ (0110)

Multiplicative inversion: Rule to perform multiplicative inverse: There exists at least one element g in $\mathbb{F}_{2^m}$ such that all non-zero elements in $\mathbb{F}_{2^m}$ can be expressed as a power of g. Such an element g is called a generator of $\mathbb{F}_{2^m}$. The multiplicative inverse of an element $a = g^i$ is $a^{-1} = g^{(-i)\bmod(2^m-1)}$. The element g = (0010) is a generator for the field. The powers of g are listed in Table 7. The multiplicative identity for the field is g^0 = (0001). The multiplicative inverse of g^7 = (1011) is $g^{-7\bmod 15} = g^{8\bmod 15}$ = (0101). To verify this, we see that:

$$\begin{aligned}(1011)\cdot(0101) &= (x^3+x+1)(x^2+1) \bmod f(x)\\ &= x^5+x^2+x+1 \bmod f(x)\\ &= (x^4+x+1)(x)+(1) \bmod f(x) = 1 = (0001)\end{aligned}$$

Which is the multiplicative identity.

Table 7 shows the use of generator notation (g^e) rather than bit notation, as used in the examples above. Also, using generator notation allows multiplication without reference to the irreducible polynomial $f(x) = x^4+x+1$. In a true cryptographic application, the parameter m must be large enough to preclude the efficient generation of such a table otherwise the cryptosystem can be broken. In today's practice, m = 163 is a minimum suitable choice.

TRACE OF A FINITE FIELD ELEMENT

Trace of an element a is a linear mapping $Tr: \mathbb{F}_{p^m} \rightarrow \mathbb{F}_p$ defined by:

$$Tr(a) = \sum_{i=0}^{m-1} a^{p^i} = a + a^{p^1} + a^{p^2} + \cdots + a^{p^{m-1}}$$

Basic properties: $Tr(a^p) = Tr(a)$ and $Tr(a+b) = Tr(a)+Tr(b)$ and more generally: $Tr(v\cdot a) = v\cdot Tr(a)$ for $a, b \in \mathbb{F}_{p^m}$, $v \in \mathbb{F}_p$.

When p = 2, Tr(0) = 0 and Tr(1) = 1 if m is odd and Tr(1) = 0 if m is even. These properties can be checked easily from definition and the equality formula for finite fields of characteristic p: $(a+b)^p = a^p+b^p$ for all $a, b \in \mathbb{F}_p$, with following properties:

- For an element $a \in \mathbb{F}_{2^m}$, Tr(a) equals 0 for one half of the elements in $\mathbb{F}_{2^m}$ and equals 1 for the other half. When p > 2, there are p^{m-1} elements of trace 0 in $\mathbb{F}_{2^m}$. In fact, there is also equal distribution of values of trace function $\mathrm{Tr}(\cdot)$ in the finite field $\mathbb{F}_{2^m}$.
- For an element $a \in \mathbb{F}_{2^m}$, its trace Tr(a) = 0 if the polynomial (x^2+x+a) is reducible over $\mathbb{F}_{2^m}$ or, in other words, has two roots over $\mathbb{F}_{2^m}$. Conversely, Tr(a) = 1 if the polynomial (x^2+x+a) is irreducible over $\mathbb{F}_{2^m}$ or it has no root over $\mathbb{F}_{2^m}$.

Properties on order of an elliptic curve: The following are the Properties on Order of an Elliptic Curve:

- Let E be a non-supersingular elliptic curve, E: $y^2+xy = x^3+ax^2+b$, over the finite field $\mathbb{F}_{2^m}$. Then the order of the elliptic curve E is #E = 2Tr(a) mod 4[34].
- Let E be a non-supersingular elliptic curve: E: $y^2+xy = x^3+ax^2+b$, over the finite field $\mathbb{F}_{2^m}$. Then for any point P of order other than 2 on E, the point Q = 2p = (x_Q, y_Q) will satisfy the condition: $\mathrm{Tr}(x_Q) = \mathrm{Tr}(a)$. This property later gives a way to represent a point of an elliptic curve over a finite field $\mathbb{F}_{2^m}$ using only m bits[35].
- The elliptic curve: E: $y^2 = x^3+x^2$, over a prime finite field $\mathbb{F}_p$ has its order satisfying the modular condition: $\#E(\mathbb{F}_p) \equiv 0 \bmod 4$.

Example:

$$\mathrm{Tr}(g^3) = g^3 + g^{3\times 2} + g^{3\times 2^2} + g^{3\times 2^3}$$
$$= g^3+g^6+g^{12}+g^9 \text{(since: } g^{24 \bmod 15} = g^9)$$
$$= (1000)+(1100)+(1111)+(1010) = (0001) = g^0 = 1$$

Then $\mathrm{Tr}(g^i) = 1$, when i = 3, 6, 7, 9, 11, 12, 13 and 14; and $\mathrm{Tr}(g^i) = 0$, if otherwise; implemented using trinomial basis notation.

We can verify that two self-dual basis for the finite field $\mathbb{F}_{2^4}$ are: $\{g^3, g^7, g^{12}, g^{13}\}$ and $\{g^6, g^9, g^{11}, g^{14}\}$. For example: $\mathrm{Tr}(g^{7\times 2}) = 1$, $\mathrm{Tr}(g^7 g^3) = \mathrm{Tr}(g^{10}) = 0$ They are not normal basis. The above trinomial basis is not a dual basis since we have Tr(1) = 0. Now define $h(z) = \mathrm{Tr}(g^{-1}z), \forall z \in \mathbb{F}_{2^4}$. Then the permutation $\{1, g, g^2, g^3\}$ is its dual basis with respect to the linear function $h(\cdot)$.

In fact, there are seven other elements of $\mathbb{F}_{2^4}$ using trinomial basis: $\{1, g, g^2, g^3\}$ multiplicative subgroup $\mathbb{F}^*_{2^4}$. They are: g^2, g^4, g^7, g^{11}, g^{13} and g^{14}. The polynomial $f(x) = x^4+x+1$ is primitive and its roots are: g, g^2, g^4 and g^8. The only other primitive polynomial for the finite field $\mathbb{F}_{2^4}$ is:

$$h(x) = (x + g^7)[x + (g^7)^2][x + (g^7)^{2^2}][x + (g^7)^{2^3}]$$
$$= (x + g^7)(x + g^{14})(x + g^{13})(x + g^{11}) = x^4 + x^3 + 1$$

Here's an example using $\mathbb{F}_{2^4}/f(x)$ with generator notation to show that the point (g^3, g^8) satisfy Eq. 10 over $\mathbb{F}_{2^4}$, i.e.

$$y^2 + xy = x^3 + g^4x^2 + 1$$
$$(g^8)^2 + g^3g^8 = (g^3)^3 + g^4(g^3)^2 + 1$$
$$g^1 + g^{11} = g^9 + g^{10} + g^0 \text{ (since } g^{16(\bmod 15)} = g^1)$$
$$(0010) + (1110) = (1010) + (0111) + (0001)$$
$$(1100) = (1100)$$

Other than the simple integer multiplication of exponents, XOR is the only other operation required. All the fifteen points which satisfy Eq. 10 are as follows:

$$E(\mathbb{F}_{2^4}) = \left\{\begin{matrix} (0,\ 1) & (1,\ g^6) & (1,\ g^{13}) & (g^3,\ g^8) & (g^3,\ g^{13}) \\ (g^5,\ g^3) & (g^5,\ g^{11}) & (g^6,\ g^8) & (g^6,\ g^{14}) & (g^9,\ g^{10}) \\ (g^9,\ g^{13}) & (g^{10},\ g) & (g^{10},\ g^8) & (g^{12}, 0) & (g^{12},\ g^{12}) \end{matrix}\right\} \quad (11)$$

Which we can write in form of generator bit: P = (1111,1111) as:

$$E(\mathbb{F}_{2^4}) = \left\{\begin{matrix} (0000,\ 0001) & (0001,\ 1100) & (1100,\ 0101) & (1100,\ 1001) & (1010,\ 0111) \\ (0110,\ 1000) & (0110,\ 1110) & (0111,\ 0101) & (1100,\ 1001) & (1010,\ 1111) \\ (1010,\ 1101) & (0000,\ 0111) & (0111,\ 0101) & (1111,\ 0000) & (1111,\ 1111) \end{matrix}\right\} \quad (12)$$

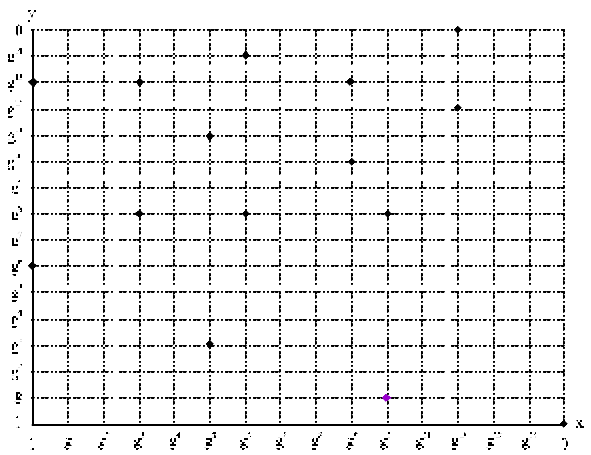

Fig. 5: Elliptic curve E: $y^2+xy = x^3+g^4x^2+1$ defined over $\mathbb{F}_{2^4}$

Figure 5 shows the graphical representation of all the points which satisfy Eq. 11. For a given point $P = (x_p, y_p)$, x_p and y_p are called the x and y coordinates of P, respectively. We now discuss efficient algorithms to expedite implementation procedures in elliptic curve cryptosystems over binary finite field, $\mathbb{F}_{2^4}$.

ALGORITHM FOR ELLIPTIC CURVE SCALAR MULTIPLICATION OVER BINARY FINITE FIELD

Cryptographic schemes based on ECC rely on scalar multiplication of elliptic curve points as demonstrated above. As before given an integer k and a point $P \in E(\mathbb{F}_{2^m})$, scalar multiplication is the process of adding P to itself k times. The result of this scalar multiplication is denoted by Q = kP = P+P+...+P (k-summands). Here P is a fixed point that generates a large subgroup of $E(\mathbb{F}_{2^4})$.

Let $P = (g^{12}, g^{12})$ be the generator of $E(\mathbb{F}_{2^4})$, which we use for scalar multiplication to determine all the points using Eq. 8 and 9 as follows:

Let $P = (x_1, y_1) = (g^{12}, g^{12})$. Then $2P = P+P = (x_3, y_3)$ is computed as follows:

$$\lambda = x_1 + \frac{y_1}{x_1} = g^{12} + \frac{g^{12}}{g^{12}} = g^{12} + g^0$$
$$= (1111) + (0001) = (1110) = g^{11}$$

$$x_3 = \lambda^2 + \lambda + a = (g^{11})^2 + g^{11} + g^4 = g^7 + g^{11} + g^4$$
$$= (1011) + (1110) + (0011) = (0110) = g^5$$

and $y_3 = x_1^2 + \lambda x_3 + x_3 = (g^{12})^2 + g^{11} \cdot g^5 + g^5$

$$= g^9 + g^1 + g^5 = (1010) + (0010) + (0110) = (1110) = g^{11}$$

Hence, $2P = (g^5, g^{11})$.

Let $P = (x_1, y_1) = (g^{12}, g^{12})$ and $Q = (x_2, y_2) = (g^5, g^{11})$. Then $P + Q = (x_3, y_3)$ is computed as follows:

$$\lambda = \frac{y_1 + y_2}{x_1 + x_2} = \frac{g^{12} + g^{11}}{g^{12} + g^5} = \frac{g^0}{g^{14}} = g^{-14 \bmod 15} = g^1$$

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$$
$$= (g^1)^2 + g^1 + g^{12} + g^5 + g^4 = g^6$$

and $y_3 = (x_1 + x_3)\lambda + x_3 + y_1 = (g^{12} + g^6)g^1 + g^6 + g^{12}$
$$= g^{13} + g^7 + g^6 + g^{12} = g^8$$

Hence, $P+Q \equiv P+2P = 3P = (g^6, g^8)$. Using the above two operations, we can double point P to obtain 2P. We can then add P to 2P to obtain 3P and continue in this manner until we eventually reach nP = O, the identity point. Since O+P = P, there are n distinct multiples of P. Below we demonstrate this procedure:

Next by again taking $P = (g^{12}, g^{12})$ and $3P = (g^6, g^8)$ we can calculate for $4P = P+3P = (x_4, y_4)$ as follows:

$$\lambda = \frac{y_1 + y_2}{x_1 + x_2} = \frac{g^{12} + g^8}{g^{12} + g^6} = \frac{g^9}{g^4} = g^5$$

$$x_4 = \lambda^2 + \lambda + x_1 + x_2 + a$$

$$= (g^5)^2 + g^5 + g^{12} + g^6 + g^4 = g^0 = 1$$

and $y_4 = (x_1 + x_4)\lambda + x_4 + y_1 = (g^{12} + g^0)g^5 + g^0 + g^{12}$
$$= g^2 + g^5 + g^0 + g^{12} = g^6$$

Hence, $P+3P = 4P = (1, g^6)$. Continuing with the similar computational procedure, we can obtain all the points in $E(\mathbb{F}_{2^4})$, expressed as multiples of P (Table 8).

We note that since point coordinates are elements of $\mathbb{F}_{2^m}$, all arithmetic in the equations above follows the complex rules of $\mathbb{F}_{2^m}$. So the multiplication is done modulo the irreducible polynomial f(x) and the division operation ('/') in the calculation for λ is actually a field inversion operation. Thus, computing curve points is quite time-consuming for machines to perform and makes it infeasible for a hacker to use a brute-force attack by calculating all multiples of P which is

Table 8: Scalar multiplication of elliptic curve points of $E(\mathbb{F}_{2^4})$ using generator $P = (g^{12}, g^{12})$

$1P = (g^{12}, g^{12})$	$2P = (g^{5}, g^{11})$	$3P = (g^{6}, g^{8})$	$4P = (1, g^{6})$	$5P = (g^{9}, g^{10})$
$6P = (g^{10}, g^{8})$	$7P = (g^{3}, g^{8})$	$8P = (0, 1)$	$9P = (g^{3}, g^{13})$	$10P = (g^{10}, g)$
$11P = (g^{9}, g^{13})$	$12P = (1, g^{13})$	$13P = (g^{6}, g^{14})$	$14P = (g^{5}, g^{3})$	$15P = (g^{12}, 0)$
$16P = O$				

equivalent to solving discrete logarithm in elliptic curve arithmetic, i.e. elliptic curve discrete logarithm problem, ECDLP[8].

NORMAL AND OPTIMAL NORMAL BASIS REPRESENTATION PRIMITIVE NORMAL BASIS

A normal basis $\{\beta, \beta^2, \beta^{2^2}, \dots, \beta^{2^{m-1}}\}$ of the finite field GF(q^m) over a finite field $\mathbb{F}_q$ where q is any prime power, is called a primitive normal basis if β is a primitive root of the multiplicative subgroup GF(q^m)*. According to the theorem by Lenstra and Schoof[36] for every prime power q > 1 and every positive integer m, there exists a primitive normal basis of finite field GF(q^m) over $\mathbb{F}_q$.

Normal basis representation: Normal basis are not special only for finite fields of characteristic 2. More generally, the field $\mathbb{F}_{2^m}$ can be viewed as a vector space of dimension m over $\mathbb{F}_2$. That is, there exists a set of m elements $g_0, g_1, \dots, g_{m-1}$ in $\mathbb{F}_{2^m}$ such that each $g \in \mathbb{F}_{2^m}$ can be written uniquely in the form:

$$g = a_0 g_0 + a_1 g_1 + \cdots + a_{m-1} g_{m-1} \qquad \text{where, } a_i \in \{0,1\}$$

We can then represent g as the 0-1 vector $(a_0 a_1 \dots a_{m-1})$. Addition of the field elements is performed by bitwise XOR-ring the vector representation

In fact, they are defined for any finite field $\mathbb{F}_{q^m}$ where, q is a prime power. A normal basis of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$ is a basis of the form: $\{\beta, \beta^2, \beta^{2^2}, \dots, \beta^{2^{m-1}}\}$, where, $\beta \in \mathbb{F}_{2^m}$. Such a basis always exists. Then $g = (a_0, a_1, \dots, a_{m-1})$ will represent the element: $g = a_0\beta + a_1\beta^2 + a_2\beta^{2^2} + \dots + a_{m-1}\beta^{2^{m-1}}$. By convention, the ordering of bits in normal basis representation is different from that in polynomial basis representation. Particularly, we can write the zero element 0= (0,0,…,0) and multiplicative identity 1 = (1,1,…,1).

Addition of field elements in normal basis is performed by bitwise XOR-ing the vector representations. However, the most important property of a normal basis is that the square of a field element can be computed easily and implemented efficiently on hardware by just a right 1-cyclic shift on the register. Indeed, given an element $g = (a_0, a_1, \dots, a_{m-1})$ represented in a normal basis, we have:

$$g^2 = \left(\sum_{i=0}^{m-1} a_i \beta^{2^i}\right)^2 = \sum_{i=0}^{m-1} a_i \beta^{2^{i+1}}$$
$$= \sum_{i=0}^{m-1} a_{i-1} \beta^{2^i} = (a_{m-1}, a_0, a_1, \dots, a_{m-2})$$

With indices reduced modulo m. Then for any integer s, 1 ≤ s ≤ m-1, the 2^s-th power of element g can be computed quickly by a right s-cyclic shift. That is, $g^{2^s} = (a_{m-s}, a_{m-s+1}, \dots, a_0, a_1, \dots, a_{m-s-1})$. We can verify again the relationship: $g^{2^m} = g$. Similarly, the square root of g can be computed simply by left 1-cyclic shift: $g_{1/2} = (a_1, a_2, \dots, a_{m-1}, a_0)$. This is useful for recovering points when using the point compression technique for embedding messages for encryption. Hence, a normal basis representation over $\mathbb{F}_{2^m}$ is advantageous because squaring a field element can then be accomplished by a simple rotation of the vector representation, an operation that is easily implemented in hardware.

Multiplication in a normal basis: Take, for example, the product C=AB is given by:

$$C = \sum_{i=0}^{m-1} C_i \beta^{2^i} = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} A_i B_j \beta^{2^i + 2^j}$$

Since $\beta^{2^i + 2^j}$ is also an element in $\mathbb{F}_{2^m}$, it can be expressed as:

$$\beta^{2^i + 2^j} = \sum_{r=0}^{m-1} \lambda_{ij}^{(r)} \beta^{2^r}$$

Where, $\lambda_{ij}^{(r)} \in \mathbb{F}_2$. This yields a formulae

$$C_r = \sum_{i=0}^{m-1} A_i B_i \lambda_{ij}^{(r)} \qquad \text{for } 0 \le r \le m-1$$

We also notice that:

$$\beta^{2^{i-s} + 2^{j-s}} = \sum_{r=0}^{m-1} \lambda_{i-s,j-s}^{(r)} \beta^{2^r} = \sum_{r=0}^{m-1} \lambda_{ij}^{(r)} \beta^{2^{r-s}}$$

Which implies that:

$$\lambda_{ij}^{(s)} = \lambda_{i-s,j-s}^{(0)} \quad \text{for } 0 \le i,j,s \le m-1$$

Thus, we have a formula for C_r as:

$$C_r = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} A_{i+r} B_{i+r} \lambda_{ij}$$

This formula has remarkable properties. Consider a circuit built for computing C_0, which receives the inputs as (in this order):

$$A_0, A_1, \ldots, A_{m-2}, A_{m-1}$$
$$B_0, B_1, \ldots, B_{m-2}, B_{m-1}$$

Uses the formulae to compute:

$$C_0 = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} A_i B_i \lambda_{ij}$$

The same circuit can be used to compute C_1:

$$A_1, \ldots, A_{m-2}, A_{m-1}, A_0$$
$$B_1, \ldots, B_{m-2}, B_{m-1}, B_0$$

Unfortunately, as can be seen above multiplication in a normal basis is somehow complicated. Fortunately, multiplication can be easily implemented in hardware when the normal basis used is of a special type called an optimal normal basis[37], which appear to give the most efficient implementation of field arithmetic (with respect to speed and complexity of hardware architecture).

Current implementation of elliptic curves: The algebraic representation of elliptic curves in $\mathbb{F}_{2^m}$ is far more efficient than the geometric representation when computing group operations. However, even the simplified algebraic representation of $\mathbb{F}_{2^m}$ is more than a simple computer can handle in reasonable time. It is possible to represent an elliptic curve in $\mathbb{F}_{2^m}$ so that multiplication can be performed by a few simple linear equations and exponentiation by a left bit rotation. This representation is based on a linear algebraic representation and is called the optimal normal basis representation - a less insightful way to construct the field $\mathbb{F}_{2^m}$, but is computational nicer and easy handling by computers.

Optimal normal basis representation: For many values of m, the finite field $\mathbb{F}_{2^m}$ has an optimal basis representation as well as the polynomial representation described above. An optimal basis gives an alternative way of defining multiplication on the elements of a field. While optimal normal basis multiplication is less insightful than polynomial multiplication, it is in practice much more efficient. Further information on $\mathbb{F}_{2^m}$ using an optimal basis representation also given by Ron *et al.*[38].

Construction of optimal normal basis representation: Optimal normal basis representation (ONB) can be used for a large number of values of m in $\mathbb{F}_{2^m}$. This representation views field elements as linear combinations of the basis matrix for the field. The value of m determines whether a Type I or Type II representation will be used. The determination of the optimal normal basis is as follows:

If $\mathbb{F}_{2^m}$ has only a Type I representation, then define $f(x) = x^m + x^{m-1} + x^{m-2} + \ldots + x^2 + x + 1$. Otherwise, define f(x) by the recursive formula (Type II):

$$f_0(x) = 1,$$
$$f_1(x) = x+1$$
$$f_{i+1}(x) = xf_i(x) + f_{i-1}(x) \text{ for } i = 1, 2, \ldots, m$$

Note that f will be reduced in every iteration and will have coefficients in $\mathbb{F}_2$. The set of polynomials $\{x, x^2, x^{2^2}, \ldots, x^{2^{(m-1)}}\}$ form a basis of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$, called a normal basis. Form A, a m×m matrix, where each row i is the bit string of length m corresponding to, $x^{2^i} \bmod f(x)$. Find A^{-1}.

Next construct T′, a m × m matrix, where each row i is the bit string of length m corresponding to, $x \cdot x^{2^i} \bmod f(x)$. Compute $T = T'.T^{-1}$. Determine the product terms, $I_{ij} = T(j-i,-1)$, for: i, j = 0, 1,…,m-1. These product terms specify the set of equations by which to perform all operations in $\mathbb{F}_{2^m}$.

The generation of the optimal normal basis will be clearer with an example. (Field addition is done as is the case with polynomial basis.) However, for field multiplication let's consider again our field $\mathbb{F}_{2^4}$, which has a Type I ONB so that: $f(x)=x^4+x^3+x^2+x+1$. Thus, $\{x, x^2, x^4, x^8\}$ forms a basis for $\mathbb{F}_{2^4}$ over $\mathbb{F}_2$.

Construct A as:

Row 0: $x \bmod f(x) = x = (0010)$
Row 1: $x^2 \bmod f(x) = x^2 = (0100)$
Row 2: $x^4 \bmod f(x) = x^3+x^2+x+1 = (1111)$
Row 3: $x^8 \bmod f(x) = x^3 = (1000)$

$$\Rightarrow \quad A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad \text{and } A^{-1} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Construct T as:
Row 0: $v = x \cdot x \bmod f(x) = x^2 = (0100)$
Row 1: $v = x \cdot x^2 \bmod f(x) = x^3 = (0100)$
Row 2: $v = x \cdot x^4 \bmod f(x) = x^5 \bmod f(x) = 1 = (0001)$
Row 3: $v = x \cdot x^8 \bmod f(x) = x^9 \bmod f(x) = x^3 + x^2 + x$
$= (1111)$

$$\Rightarrow \quad T' = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad \text{and } A^{-1} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$T = T' \cdot A^{-1}$$

$$= \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Finally, the L_{ij} terms which are 1 are: $L_{0,2}$, $L_{1,2}$, $L_{1,3}$, $L_{2,0}$, $L_{2,1}$, $L_{3,1}$ and $L_{3,3}$,

$$L = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

How do we use L to define operations in $\mathbb{F}_{2^4}$? Recall that $\{x, x^2, x^4, x^8\}$ spans $\mathbb{F}_{2^m}$ and L is a optimal normal basis for $\mathbb{F}_{2^m}$. We will write a general formula for multiplication in $\mathbb{F}_{2^4}$:

Let $A \cdot B = C = (a_0 a_1 a_2 a_3) \cdot (b_0 b_1 b_2 b_3) = (c_0 c_1 c_2 c_3)$, all in $\mathbb{F}_{2^4}$. Write $(a_0 a_1 a_2 a_3)$ and $(b_0 b_1 b_2 b_3)$ in terms of the basis $\{x, x^2, x^4, x^8\}$, i.e.,

$$\begin{bmatrix} a_0 & a_1 & a_2 & a_3 \\ a_1 & a_2 & a_3 & a_0 \\ a_2 & a_3 & a_0 & a_1 \\ a_3 & a_0 & a_1 & a_2 \end{bmatrix}$$

$$\begin{bmatrix} b_0 & b_1 & b_2 & b_3 \\ b_1 & b_2 & b_3 & b_0 \\ b_2 & b_3 & b_0 & b_1 \\ b_3 & b_0 & b_1 & b_2 \end{bmatrix}$$

Where, each squaring results in a cyclic shifts.

$$L \cdot B = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} b_0 & b_1 & b_2 & b_3 \\ b_1 & b_2 & b_3 & b_0 \\ b_2 & b_3 & b_0 & b_1 \\ b_3 & b_0 & b_1 & b_2 \end{bmatrix} = \begin{bmatrix} b_2 & b_3 & b_0 & b_1 \\ b_2 + b_3 & b_3 + b_0 & b_0 + b_1 & b_1 + b_2 \\ b_0 + b_1 & b_1 + b_2 & b_2 + b_3 & b_3 + b_0 \\ b_1 + b_3 & b_1 + b_0 & b_3 + b_1 & b_0 + b_2 \end{bmatrix}$$

The elements of C are the inner products of the corresponding rows and columns of A and B:

$$c_0 = \begin{bmatrix} a_0 & a_1 & a_2 & a_3 \end{bmatrix} \cdot \begin{bmatrix} b_2 \\ b_2 + b_3 \\ b_0 + b_1 \\ b_1 + b_3 \end{bmatrix} = a_0 b_2 + a_1(b_2 + b_3) + a_2(b_0 + b_1) + a_3(b_1 + b_3)$$

$$c_1 = \begin{bmatrix} a_1 & a_2 & a_3 & a_0 \end{bmatrix} \cdot \begin{bmatrix} b_3 \\ b_3 + b_0 \\ b_1 + b_2 \\ b_1 + b_3 \end{bmatrix} = a_1 b_3 + a_2(b_3 + b_0) + a_3(b_1 + b_2) + a_0(b_2 + b_0)$$

$$c_2 = \begin{bmatrix} a_2 & a_3 & a_0 & a_1 \end{bmatrix} \cdot \begin{bmatrix} b_3 \\ b_3 + b_0 \\ b_1 + b_2 \\ b_1 + b_3 \end{bmatrix} = a_2 b_0 + a_3(b_0 + b_1) + a_0(b_2 + b_3) + a_1(b_3 + b_1)$$

$$c_3 = \begin{bmatrix} a_3 & a_0 & a_1 & a_2 \end{bmatrix} \cdot \begin{bmatrix} b_1 \\ b_1 + b_2 \\ b_3 + b_0 \\ b_0 + b_2 \end{bmatrix} = a_3 b_1 + a_0(b_1 + b_2) + a_1(b_3 + b_0) + a_2(b_0 + b_2)$$

With these definitions of addition and multiplication, the 16 binary 4-tuples form a field, although this fact is not obvious.

Note:

- The multiplicative identity is (1111) (cf. polynomial basis which was (0001))
- Notice that: $(a_0a_1a_2a_3)\cdot(a_0a_1a_2a_3) = (a_0a_1a_2a_3)^2 = (a_3a_0a_1a_2)$ and so the square of a field element is simply a right cyclic shift of its vector representation.
- The formula for c_1 in the multiplication can be obtained by adding a 1 to each subscript in the formula for c_0 (where the subscripts are reduced modulo 4). The formula for c_2 can be obtained by adding 2 to each subscript in the formula for c_0 and reducing the formula modulo 4. The formula for c_3 can likewise be obtained.
- From the preceding remark we see that a circuit to compute c_0 from $(a_0a_1a_2a_3)$ and $(b_0b_1b_2b_3)$ can be used to compute c_1 by applying $(a_0a_1a_2a_3)$ and $(b_0b_1b_2b_3)$ to it. Similarly, c_2 and c_3 can be computed by shifting to the left input vectors.

Thus: $(a_0a_1a_2a_3)\cdot(b_0b_1b_2b_3) = (c_0c_1c_2c_3)$, is implemented as follows:

$(c_0c_1c_2c_3) = (0100)\cdot(1101)$
$c_0 = 0(0)+1(0+1)+0(1+1)+0(1+1) = 1$
$c_1 = 1(1)+0(1+1)+0(1+0)+0(0+1) = 1$
$c_2 = 0(1)+0(1+1)+0(0+1)+1(1+1) = 1$
$c_3 = 0(1)+0(1+0)+1(1+1)+0(1+0) = 0$

and $(0100)\cdot(1101) = (1100)$

Also, $(1010)^{10} = (1010)^2\cdot(1010)^8$
$= (0101)\cdot(0101)$
$= (1010)$

and $(1101)^9 = (1101)\cdot(1101)^8$
$= (1101)\cdot(1011)$
$= (0001)$

Once the multiplication algorithms are computed, they need not be recomputed ever again. The optimal normal basis greatly simplifies multiplication in $\mathbb{F}_{2^m}$, but there is an even greater benefit. The square of any field element is computed to the left cyclic shift of that element, a trivial operation!

Additionally, since an element has a cycle of length m, we have the following consequence as by example:

$$(0010)^{161} = (0010)^{161 (\text{mod } 4)} \cdot (0010)^1 = (0010)$$

Exponentiation can be simplified and reduced! Consider the finite field $\mathbb{F}_{2^4}$ using a normal basis. The reduction polynomial is $f(x) = x^4+x+1$, which you may recall is a primitive polynomial. The optimal normal basis of Type II consists of four polynomials: x, x^2, x^4 and x^8(mod f(x)). A generator for non-zero elements of multiplicative subgroup $\mathbb{F}_{2^4}^{\times}$ is chosen to be g = x(mod f(x))

In this representation $\mathbb{F}_{2^4}$ can be generated by the powers of g = (1100). We can write them explicitly in the Table 9, where:

$$(a_0, a_1, a_2, a_3) = a_0x+a_1x^2+a_2x^4+a_3x^8 (\text{mod } f(x))$$

For instance, $1 = (1111) = x + x^2 + x^4 + x^8$. (All polynomials of modulo f(x) and reduced modulo 2.) We can compute the following intermediate terms and the next repeated squares of each term (by simple right shifts). For example:

$x^5 = x^2 + x = x + x^4 = (1010)$ and $x^{10}(0101)$
$x^9 = x^3 + 2\cdot x^2 + x = x^3 + x = x = (1000)$
$x^{12} = x^3 + 3\cdot x^2 + 3.x + 1 = x^3 + x^2 + x + 1 = x^8 = (0001)$

Arithmetic in the finite field $\mathbb{F}_{2^m}$ can be performed efficiently both in hardware and in software when the field elements are represented with respect to an optimal normal basis.

Table 9: Powers of g using the element g=(1100) as the generator. This method of representation is called optimal normal basis representation, of Type II (g, g^2, g^4, g^8)

$0 = (0000)$	$g^1 = (1100) = g + g^2$	$g^2 = (0110) = g^2 + g^4$	$g^3 = (0100) = g^2$	$g^4 = (0011) = g^4 + g^8$	$g^5 = (1010) = g + g^8$
$g^6 = (0010) = g^4$	$g^7 = (0111) = g^2 + g^4 + g^8$	$g^8 = (1001) = g + g^8$	$g^9 = (1000) = g$	$g^{10} = (0101) = g^2 + g^8$	$g^{11} = 1110) = g + g^4 + g^8$
$g^{12} = (0001) = g^8$	$g^{13} = (1101) = g + g^4 + g^8$	$g^{14} = (1011) = g + g^4 + g^8$	$g^{15} = (1111) = g^0 = 1 = g + g^2 + g^4 + g^8$		

Table 10: Scalar multiplication of elliptic curve points of $E(\mathbb{F}_{2^4})$ using generator $P = (g^3, g^5)$

$1P = (g^3, g^5)$	$2P = (g^4, g^3)$	$3P = (g^{13}, g^2)$	$4P = (g, 0)$	$5P = (g^{12}, g^8)$
$6P = (g^8, g^3)$	$7P = (g^{11}, 0)$	$8P = (g^5, g^{11})$	$9P = (g^6, 0)$	$10P = (0, g^9)$
$11P = (g^6, g^6)$	$12P = (g^5, g^3)$	$13P = (g^{11}, g^{11})$	$14P = (g^8, g^{13})$	$15P = (g^{12}, g^9)$
$16P = (g, g)$	$17P = (g^{13}, g^{14})$	$18P = (g^4, g^7)$	$19P = (g^3, g^{11})$	$20P = (O)$

Since we have demonstrated an example of elliptic curve over $\mathbb{F}_{2^4}$ using polynomial basis representation, here we will now give an illustration using optimal normal basis representation. Consider the field $\mathbb{F}_{2^4}$ where the elements are the set of all binary 4tuples with multiplication given by the formulae. Recall that g = (1100) is a generator for the non-zero elements and (1111) is the multiplicative identify.

Consider the non-supersingular curves over $\mathbb{F}_{2^4}$ defined by the equation:

$$E:\ y^2+xy = x^3+g^3$$

(Note that to be absolutely precise with our notation, we should write this equation as:

$$(1111)y^2+(1111)xy = (1111)x^3+(0100)$$

Since (1111) is the multiplicative identity, we simplify our notation as above). The solutions over $\mathbb{F}_{2^4}$ to the elliptic curve equation are:

$$E(\mathbb{F}_{2^4}) = \begin{Bmatrix} (0,g^9) & (g,0) & (g,g) & (g^3, g^5) & (g^3, g^{11}) \\ (g^4, g^3) & (g^4, g^7) & (g^5, g^3) & (g^5, g^{11}) & (g^6, 0) \\ (g^6, g^6) & (g^8, g^3) & (g^8, g^{12}) & (g^{11}, 0) & (g^{11}, g^{11}) \\ (g^{12}, g^8) & (g^{12}, g^9) & (g^{13}, g^2) & (g^{13}, g^{14}) & \end{Bmatrix} \quad (13)$$

Since there are 19 solutions to the equation in $\mathbb{F}_{2^4}$, the group $E(\mathbb{F}_{2^4})$ has 19+1=20 elements, i.e. the group order $\#E(\mathbb{F}_{2^4}) = n = 20$. This group turns out to be a cyclic group of order 20. If we take P=(g^3, g^5) and use the addition formulae, we find group points as listed in (Table 10).

Implementing an elliptic curve in $\mathbb{F}_{2^m}$ with an optimal normal basis greatly speeds computation and lowers storage requirements. In addition to its speed, the elliptic curve resists breaking by current number field sieve methods and the index calculus method used by many such techniques has not been formulated for the elliptic curve. It is very likely that elliptic curve cryptography will dominate cryptosystems in the near future. Recall that in a true cryptographic application, the parameter m must be large enough to preclude the efficient generation of such a table otherwise the cryptosystem can be broken. In today's practice, m = 163 is considered acceptable.

DESIGN AND IMPLEMENTATION OF CRYPTO-SECURE COMMUNICATION PROTOCOL

Design and implementation of the right crypto-algorithm, in general, will provide the fundamental security, however, improper management of the algorithms can lead to insecure applications[1,39]. The prevention of such mishaps often lies in well-defined crypto-protocols. Perhaps the most famous in public key cryptography is the Diffie-Hellman (DH) key exchange protocol. This protocol introduced the public key concept to the world in 1976[40] and has remained a very popular protocol for strong authentication of entities. More recently, driven by needs of the embedded systems world, DH analog haven been introduced for ECC.

The Diffie-Hellman (DH) key agreement protocol is the basic public-key crypto system proposed for secret-key sharing. ECDH is the elliptic curve analog of the traditional Diffie-Hellman key agreement algorithm[40]. The Diffie-Hellman method requires no prior contact between the two parties. Each party generates a dynamic, or ephemeral, public key and private key. They exchange these public keys. Each party then combines its private key with the other party's public key to form the shared secret. This method is also known as carrying out an ECDH key agreement[14].

The public-key crypto-schemes like RSA and ECC, when used for communication protocols for data encryption are extremely computationally demanding and thereby slower than the symmetric ones but provide arbitrary high levels of security and do not require an initial private-key exchange between two communicating parties. However, many symmetric key schemes can encrypt 1000 times faster than public key cryptosystems but are, however, poor at providing non-repudiation and key establishment functionality[39].

In real applications, however, most practical crypto-protocols are hybrid protocols, which incorporate integrated coupled symmetric and public key schemes. The public-key crypto-algorithm is first used for establishing a common symmetric-key over insecure channel[40,41]. Then the symmetric cryptosystem is used for secure communication with high throughput. Due to comparative slowness of the public-key algorithms, dedicated hardware is desirable for efficient implementation and operation of the cryptographic systems. In most applications symmetric key cryptosystems e.g., RC5, IDEA, DES/3DES and AES are today commonly integrated with existing public key cryptosystem like RSA or ECC for efficient crypto-network security.

Crypto-key length considerations: The security of the elliptic curve cryptographic schemes hinges on the apparent difficulty of the discrete logarithm problem in elliptic curve (DLP)[8]. The best algorithm known for the elliptic curve logarithm for non-supersingular curves is the Pollard-ρ algorithm[42] which takes $\sqrt{\pi n}/(2r)$, steps, where, r is the largest prime divisor of the order n of the elliptic curve point P. Thus, to thwart an attack, the underlying field $\mathbb{F}_q$, the curve E and base point P should be selected so that the order n of P is divisible by a prime number r, which is sufficiently large for the purpose at hand.

Keys in ECC are generated by relying on the Elliptic Curve Discrete Logarithm Problem (ECDLP) which is the problem of determining an integer k (provided it exists) such that kP = Q (or equivalently, $K = \log_P Q$), where, P and Q are points on the elliptic curve[8]. ECC does not involve exponentiation, but uses multiplication, hence is not as computationally costly as RSA, which relies on exponentiation computation. Also, the underlying mathematical problem of ECC, ECDLP is fully exponential, whereas sub-exponential algorithms exist for IFP used in RSA[4]. Because it is more 'infeasible' to solve ECDLP the same level of security, can be provided with shorter keys in ECC compared with RSA.

Menezes[11] gives comparisons of the difficulty of these problems. His conclusion is that the ECDLP is computationally more difficult than the DLP or the IFP, hence, cryptosystem based on elliptic curve can be as secure as their more traditional counterparts with significantly smaller fields (Fig. 6). The expected time to solve the ECDLP with a point P as the generator having order of a 160-bit prime is approximately equal to the time required to solve the DLP with a 1024-bit modulus or to factor a 1024-bit n into primes p and q(= 2^m) (Table 1). These estimates are based on the currently best-known algorithms for the problems: for

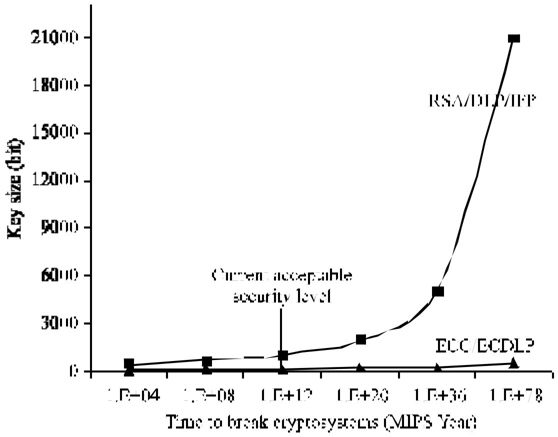

Fig. 6: Comparison of security levels of ECC/ECDLP and RSA/DLP/IFP

the ECDLP this is Pollard's-ρ method; for the DLP and the IFP the best algorithm is the number field sieve. The reason for the large difference in field size for the different systems is because of the running times of the algorithms for the problems. No sub-exponential time algorithm for the ECDLP has been found that applies to general curves; the fastest for the DLP are not applicable in the elliptic curve setting. Due to these factors, ECC is better suited for low bandwidth, computational power and memory situations especially in mobile and wireless environment.

Secure key establishment and generation: Crypto-security communication network protocols usually consist of two phases. The first one is the key establishment phase, which is done initially to exchange the keys. Thereafter, in normal mode application data is transmitted over the insecure network. In key establishment phase one may use ECDH or MQV key agreement scheme. Pure DH or ECDH applications are, however, susceptible to impersonation or "man in the middle" attack, whereby an adversary establishes digital facades between two parties in order to obtain private information. Advanced key exchange protocols such as the Menezes-Qu-Vanstone (MQV) introduce mutual strong authentication which allows both parties to confidently identify each other before exchanging sensitive information[5,8,11]. It is a method of key agreement, which is related to Diffie-Hellman, but offers some significant advantages. MQV offers attributes-such as key-compromise impersonation resilience and unknown key-share resilience (the unknown key-share resilience property has been demonstrated unachievable in this version of protocol)- that are not found with ECDH. However, ECDH offers a very simple and efficient way of creating a shared secret between two entities.

Trusted third party: Certificate Authority (CA): For a added level of transaction security, a Certificate Authority (CA) can be used. A CA is a third part that is trusted to perform the service of validating information about each user and creating signed certificates to that effect. A certificate is a packet of information which includes the users (e.g. Bank's) public key, email address, name, address and other useful information, such as expiration date of the certificate and user privileges. A CA creates, distributes, revokes and generally manages these certificates. For, example, Jane wants to obtain the Bank's public key, she retrieves its certificate from the public key server directory and verifies the CA's signature on the certificate itself. Provided this signature verifies correctly, she has the CA's assurance that the Bank's identity, its public key and all other information in the certificate is correct. Jane can now go ahead and use Bank's public key to encrypt confidential information transaction to send to the Bank or to verify the Bank's signature, protected by the assurance of the certificate.

THE ELLIPTIC CURVE DIFFIE-HELLMAN (ECDH) KEY AGREEMENT SCHEME

In Elliptic Curve Diffie-Hellman (ECDH) key exchange, the two communicating parties, sever S and client C, agree beforehand to use the same curve parameters and base point P, which generates all the points on the elliptic curve E or the order of curve #E. Each party generates their private keys k_S (server) and k_C (client), respectively and the corresponding public keys: $Q_S = k_SP$ and $Q_C = k_CP$.

Both the client and server then exchange their public keys and each multiplies its private key with the other party's public key to derive a common shared secret key: $Q_{SC} = k_SQ_C = k_S(k_CP) = k_{SC}P$, where, $(k_{SC} \equiv k_{CS}) = k_{Msec}$ is the shared master private key. The shared master private key, k_{Msec}, may now be used to encrypt or decrypt a shared message over the insecure network. An attacker cannot determine the shared private key k_{Msec} from the curve parameters, P or the public keys of the parties.

If, however, the attacker can recover k_{Msec} from this data then he is said to solve Diffie-Hellman Problem (DHP). However, it turns out that if the numbers are all sufficiently large , it is very hard to calculate the discrete logarithm in a reasonable time, i.e., finding. $k_{MSec} = \log_P Q_{SC}$. The security of the Diffie-Hellman algorithm depends on this fact. It is believed for most groups in use in cryptography that DHP and the DLP are equivalent[43,44], in complexity-theoretic sense (there is a polynomial time reduction of one problem to the other and vice versa. Maurer *et al.*[45] has shown that breaking the Diffie-Hellman protocol was equivalent to computing discrete logarithms under certain assumptions.

Key establishment phase for wireless transaction: The client initiates a connection by relaying a ClientHello with its precomputed public key Q_C on the insecure wireless channel (Fig. 7). The gate, which is in receive mode, on receiving the client public key passes it to the server through the wired interface and waits for the server's public key. The server daemon, upon receiving the connection request sends a ServerHello with its public key Q_S to the gateway and then starts computing the shared master secret key from the received public key Q_C and the server's secret key k_S:

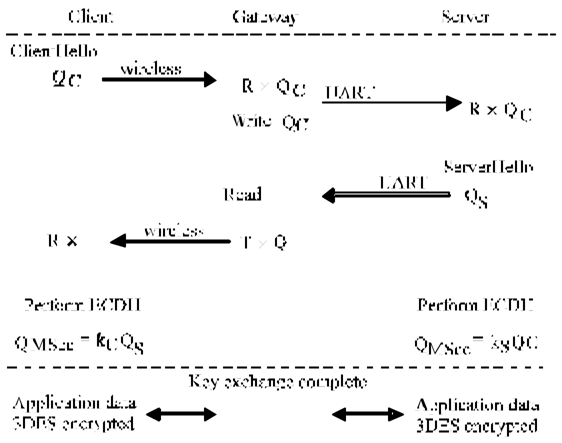

Fig. 7: ECDH key exchange protocol

$$Q_{MSecret} = k_SQ_C \tag{14}$$

Where, $Q_{MSecret}$ is the master secret, to be strictly kept away from the enemy. The gateway transmits the server's public key to the client and waits for the data transmission to begin. The client, on receiving the server's public key Q_S from the gateway computes the master secret:

$$Q_{MSecret} = k_CQ_S \tag{15}$$

The client and server now have the same shared master secret $Q_{MSecret}$ of m-bits using the ECDH key exchange. The key for the symmetric crypto-scheme operation is then derived from this m-bit master secret.

Normal mode of operation: Once the keys are set up, the client and the sever can commence transmission of application data encrypted with symmetric-crypto scheme, e.g. Triple-DES in CBC mode[16]. To close the crypto-network connection securely, we use a close connection control message which deletes the previously generated keys.

Practical application: In a practical application, for example wireless point of sale systems, an exchange begins when a user swipes a smartcard, such as a credit, on the client device-a point of sale terminal. The client first saves the data encoded on the magnetic strip or IC

Table 11: NIST Guidelines for public-key sizes with equivalent security levels[13]

Security (Bits)	Symmetric encryption algorithms	Minimum size (bits) of public keys DSA/DH	RSA	ECC	RSA/ECC
80	Skipjack	1024	1024	160	6:1
112	3DES	2048	2048	224	9:1
128	AES-128	3072	3072	256	12:1
192	AES-192	7680	7680	384	20:1
256	AES-256	15360	15360	512	30:1

Table 12: Hardware implementations of ECC and RSA

Hardware comparison: 128-bit security level		
Mode	RSA-3072	ECC-283
Space-optimized (same clock speed)	(VLSI Cores) 184 ms	(Grobschadl) 29 ms
	50,000 gates	(16 ms for Koblitz curves) 6,6660 gates
Space-optimized (same clock speed)	(VLSI Cores) 110 ms	(Orlando and Paar) 1.3 ms
	189,200 gates	6,6660 gates

device on the smartcard and then initiates the ECDH key exchange protocol. After the shared master secret key is established, a symmetric crypto-algorithm like 3DES engine is fired. The card data is then encrypted with 3DES in CFB mode on the client side and the application data exchange can commence which must be securely terminated on completion of communication protocol.

Relative crypto-key sizes: So what does this mean in practice? NIST has recommended that 128-bit protection is necessary to achieve relatively lasting security (to the year 2036 and beyond) for the symmetric key crypto-schemes. This means moving from 3DES to AES[16]. To avoid compromising the security of the system, NIST's FIPS 140-2 standard[46] indicates that keys for symmetric ciphers such as AES must be matched in strength by public-key algorithms such as RSA and ECC. For example, a 128-bit AES key demands an RSA key size of 3,072-bits for equivalent security but for the same strength, the ECC key size is only 256-bits (Table 11).

ECC key sizes scale linearly, RSA does not. The result is that the gap between systems grows as the key sizes increase. This is especially relevant to implementations of AES where at 256-bit security you need an RSA key size of 15,360-bits compared to 512-bits for ECC (Table 11).

This will have a significant impact on a communication system as the relative computational performance advantage of ECC versus RSA is not indicated by the key sizes but by the cube of the key sizes. The difference becomes even more dramatic as the greater increase in RSA key sizes leads to an even greater increase in computational cost. So going from 1024-bit RSA key to 3072-bit RSA key requires about 27 times (3^3) as much computation while ECC would only increase the computational cost by just over 4 times (1.6^3).

Even in systems that use 1024-bit RSA, this has important design implications. For example, ECC-160 has a 6X smaller key-size than RSA-1024 and can generate a signature 12 times faster on StrongARM7. The performance advantage of ECC stands out even more when we look at higher security levels or when looking at hardware implementations. Table 12 makes clear the relative advantages of ECC on the hardware side of things, in terms of speed and gate count.

IMPLEMENTATION OF ELLIPTIC CURVE ENCRYPTION SCHEME (ECES) OVER FINITE FIELD GF(2^m)

System setup: An underlying finite field $\mathbb{F}_q$ is chosen, where, $q = 2^m$. An elliptic curve E defined over $\mathbb{F}_q$ and a point $P \in E$ are chosen. The order of the point P is denoted by n. The field $\mathbb{F}_q$, curve E, point P and the order n, comprise the system parameters and are public-key information.

Key generation: Each communicating entity shall perform the following operations:

- Entity A selects a random integer k_A from[1, n-1] and compute: A: = k_AP.
- A's public key chain is (E, P, A) placed in public key-server, A's private key is k_A kept secrete.
- Entity B selects a random integer k_B from [1, n-1] and computes: B: = k_BP.
- B's public key chain is (E, P, B) placed in public-key-server, B's private key is k_B kept secrete

Encryption process (B): Entity B sends a message M to entity A and entity B performs the following steps:

- Look up A's public-key: (E, P, A) or A.
- Represent the message M as a pair of field elements (m_1, m_2), where $m_1, m_2 \in \mathbb{F}_q$, not necessarily a point on the curve.
- Accesses its private key k_B.
- Compute his public key point: B: = k_BP: = (x_1, y_1)

- Compute the session key: S_{BA}: = k_BA: = $k_B(k_A)P$: = $k_{BA}P$: = (x_2, y_2), where, k_{BA} is the shared key.
- Combines the field elements m_1, m_2 and x_2, y_2 in a predetermined manner to obtain two ciphertext field elements c_1 and c_2.
- Transmit the data C: = (B, c_1, c_2) to A.

Decryption process (A): Entity A decrypts the ciphertext C: = (B, c_1, c_2) received from B, by performing the following steps:

- Look up B's public-key: (E, P, B) or B.
- Compute the session key: S_{AB}: = k_AB: = $k_A(k_B)P$: = $k_{AB}P$: = (x_2, y_2), using her private-key k_A.
- Recover the message m_1 and m_2 from c_1, c_2 and x_2, y_2.

ELLIPTIC CURVE SIGNATURE SCHEME (ECSS)

In this age of digital piracy it is extremely important to make sure that participants of a given transaction are really who they say they are and that the communicated content is not altered in any way during the transaction. Authentication of both parties, as well as encryption of the content in transit, is critical to limit exposure on both ends of the transaction via encryption and finally signing the messages [13,39].

Digital Signatures (DS) are the electronic equivalent of handwritten signatures. A Digital Signature Algorithm (DSA) is the specific mechanism within cryptography that provide the benefits of authentication, data integrity and non-repudiation. Traditionally, handwritten signatures have provided security services because each individual has distinctive handwriting, making their signature unique and hence difficult to forge. Similarly, securing electronic information requires the equivalent of handwritten signature that cannot be duplicated. A public key construct called digital signature, a message that is unique and exclusive to the original signer, provides the solution to this problem. A digital signature's make up is a function of both signer's identity and the data being signed, so that any changes to the message data will effect a detectable change to the digital signature, for more detail info[39].

Two signature schemes are described in ECSS standard. In this scheme the message to be signed is first hashed to a message digest of fixed length and then this digest is signed. Verification of the signature requires both the signature and the original message[39]. Elliptic curve digital signature scheme (ECDSA) is the elliptic curve analogue of the NIST digital signature scheme (DSA)[13,47]. System Setup and Key generation as performed in a similar manner to ECES.

Signature generation for ECSS: Entity A signs a message M for entity B, by performing the following steps:

- Represent the message M as a binary string.
- Use hash algorithm to compute the hash value e: = H(M).
- Select a random number integer k where, $1 \le k \le n-1$.
- Compute its public key R: = kP: = (x_R, y_R).
- Compute r: = x_R + e mod q
- Use the private key k_A to compute s: = k-k_A r mod n
- A sends to B the message M and the signature: (r, s).

Signature verification for ECSS: Entity B verifies A's signature (r, s) for a message M by performing the following steps:

- Look up A's public-key, A.
- Compute the point V = sP+rA: = (x_R, y_R)
- Compute the hash value e: = H(M)
- Compute r′: = x_R+e mod q.
- Accepts A's signature for message M if and only if r: = r′.

Note: the hash value e reduced modulo q and then modulo n. Hence, for security reasons it is necessary that the hash function H, q and n be chosen so that, (H mod q) mod n, is also a cryptographically secure hash function. Similar operation and implementation of elliptic curve digital signature can be done using ECDSA[13] and also EC-ElGamal Signature scheme[15].

SIMPLE IMPLEMENTATION OF (ECES) OVER FINITE FIELD GF(2^4)

System setup: The underlying finite field will be $\mathbb{F}_{2^4}$ and using the generator point P = (g^3, g^5) which has order $\#E(\mathbb{F}_{2^4}) = n = 20$, the data from tables 9 and 10. The system parameters (public information placed in the public-key server), are $(\mathbb{F}_{2^4}, E, P, n)$.

Key generation: Entity A (Alice) performs the following operations:

- A selects a random integer k_A = 9 from[1, n-1].
- A computes the point A = k_AP = 9P = (g^6, 0) = ((0010), (0000)).
- A's public-key is the point A and private-key is the integer k_A = 9.

Encryption process (B): (Entity B sends a message M=10101011 to entity A). Entity B performs the following steps:

- Looks up A's public-key from public-key server: $A = (g^6, 0)$.
- Represents M as a pair of field elements $M = (m_1, m_2) = ((1010), (1011)) = (g^5, g^{14})$
- Selects a random integer $k_B = 14$ from[1,19]
- Computes his public key: $B = k_B P = 14P = (g^8, g^{13}) = ((1001), (1101))$.
- Compute shared key: $S_{BA}: = k_B A = k_B(k_A P) = 14(9)P = 6P = (x_2, y_2) = (g^8, g^3) = ((1001), (0100))$.
- Compute the field element: $x_2^3 = (g^8)^3 = g^9 = (1000)$.
- Now B forms $x_4 = (1000) = g^9$ by concatenating the high order 2-bits of $x_2 = (1001)$ with the low order 2-bits of $x_2^3 = (1000)$. Next B forms $y_4 = (1001) = g^8$ by concatenating the high order 2-bits of $x_2^3 = (1000)$ with low order 2-bits of $x_2 = (1001)$. (Note that in general the probability that $(x_4, y_4) = (g^9, g^8) = ((1000), (1001))$ is a point on the curve, or that x_4, or y_4 equals 0, will be negligibly small.)
- B forms the field elements:

$$c_1 = (m_1 + x_2)x_4 = g^4 g^9 = g^{13} = (1101) \quad (16)$$

$$c_2 = (m_2 + y_2)y_4 = g^0 g^8 = g^8 = (1001) \quad (17)$$

- B transmits the data the ciphertext C: = (B, c_1, c_2) parameters to A, i.e.,:

$$C: = (g^6, 0, g^{13}, g^8) = ((0010), (0000), (1101), (1001))$$

Decryption process (A): Entity A decrypts the ciphertext, C: = (g^6, 0, g^{13}, g^8), received from B, by performing the following steps:

- A computes the session key: $S_{AB}: = k_A B = 14(9P) = 6P = (g^8, g^3) = ((1001), (0100))$..
- A forms $x_4 = (1000) = g^9$ and $y_4 = (1001) = g^8$ just as B did.
- A recovers the pair $(m_1 + x_2, m_2 + y_2)$ by computing:

$$m_1 + x_2 = c_1 x_4^{-1} = g^{13} g^{-9} = g^4 = (0011)$$
$$m_2 + y_2 = c_2 y_4^{-1} = g^8 g^{-8} = g^0 = (1111)$$

- A recovers the message pair: $m_1 = (0011) + g^8 = 1010$ and $m_2 = (1111) + g^3 = 1011$, which is the original message: $M = (m_1, m_2) = (g^5, g^{14})$

Simple implementation of EC Signature Scheme (ECSS)

Signature generation scheme: Entity A signs the message M = 777 = 1100001001. Suppose that the hash value of $\overline{e} = H(M) = 777 \bmod 16 = 9$. A performs the following steps:

- Select a random integer k = 13 in the range[1,n-1].
- Compute $R = 13P = (g^{11}, g^{11}) = (x_R, y_R)$
- Represent $x_R = g^{11} = (1110)$ as the integer: $x_R = 8 + 4 + 2 = 14$
- Compute $r = \overline{x}_R + \overline{e}(\text{mod}\, q) = (14 + 9) \bmod 16 = 7$
- Use its private key k_A=9 to compute: $S = k - k_A r (\text{mod}\, n) = 13 = 9(7) \bmod 20 = 10$
- The signature on message M is: (r, s) = (7, 10).

Signature verification for ECSS: Entity B verifies signature (r, s) = (7, 10) on M as follows:

- B looks up A's public-key: $A = (g^6, 0) = 9P$.
- B computes the point: $V = sP + rA = 10P + 7(9P) = 13P = (g^{11}, g^{11}) = (x_V, y_V)$
- Represent $x_V = g^{11} = (1110)$ as the integer: $x_R = 8 + 4 + 2 = 14$
- B computes $r' = \overline{x}_V + \overline{e}(\text{mod}\, q) = 14 + 9 \bmod 16 = 7$
- B accepts A's signature on M since $r' = 7 = r$.

THE PRACTICAL APPLICATION OF ECC

Implementing ElGamal ECC over finite field GF(2^{163}): Crypto-scientist T. ElGamal was the first mathematician to propose a public-key cryptosystem based on the Discrete Logarithm Problem (DLP) modulo prime p[48]. He in fact proposed two distinct cryptosystems: one for encryption and the other for digital signature scheme in 1984, well before elliptic curves were introduced in cryptography. Since then, many variations have been made on the digital signature system to offer improved efficiency over the original system. In 1991, Claus Schnorr discovered a variant of the ElGamal digital signature scheme, which offers better efficiency, compared to the original systems. In turn, the U.S. government's Digital Signature Algorithm (DSA) is based on ElGamal's work[39] The ElGamal public-key encryption scheme can be viewed as Diffie-Hellman key agreement protocol in key transfer mode[49]. Its security is based on the intractability of the Discrete Logarithm Problem (DLP) and the Diffie-Hellman problem. These systems are the best known of a large number of systems whose security is based on the DLP. The prime p used in the DL systems should also be at least 150 decimal digits (500 bits) in lengthto provide short-term security. The elliptic curve cryptosystems as applied to ElGamal protocols were first proposed in 1985[49].

In implementing ElGamal elliptic curve cryptosystems, suppose that two entities, the eBusiness (B) and the customer Alice (A) wants to communicate between each other over an insecure communication network. Next let's assume that the two entities have decided to use the protocol of EC-ElGamal encryption protocol to implement their secure communication. One basic point to note is unlike ECDH protocol[8,14], this protocol does not create a common key, but using EC-ElGamal protocol a message M = M_1+M_2, not necessarily a point on elliptic curve, can be sent from eBusiness to Alice and vice versa.

Let the fixed point $P = (x_P, y_P) \in GF(2^m)$ be the generator point such that the multiples kP of the generator point P are (for $1 \le k \le n-1$) including point O located at infinity. Now suppose entity A chooses public-key set: $A = k_AP$, where the secret-key k_A is selected from [1, n-1], giving rise to its public-key ring (E, P, A), which is kept in the public-key server. Next entity B selects a random secret-key k_B from [1, n-1] such that: $B = k_BP$ giving rise to its public-key ring (E, P, B) kept in the public-key server.

The sender A encrypts her message $M = M_1 + M_2$ by making use of the key exchange protocol e.g. ECDH key agreement scheme to generate a shared key: $S_{AB} = k_A(k_BP) = k_{AB}P = k_{BA}P = (x_S, y_S)$. She then performs the message encryptions as follows: $C = (c_1, c_2) = MS_{AB}$, preferably, $c_1 = M_1x_S$ and $c_2=M_2x_S$, where $M = M_1+M_2$. The encrypted ciphertext message is then sent as a package: (A, C).

Once the encrypted ciphertext is received by entity B, it looks up A's public-key from the public-key server and computes: $S_{BA} = k_B(k_AP) = k_{BA}P = (x_S, y_S)$. Subsequently $c_1(x_S)^{-1} = M_1x_S(x_S)^{-1} = M_1$ and $c_2(y_S)^{-1} = M_2y_S(y_S)^{-1} = M_2$ and the original plaintext message is $M = M_1+M_2$ is recovered. It should be noted that for the message communication between the parties to take place each must exchange their public key for the purpose of key agreement computation. Secondly, in obtaining the original message, two inversion multiplications are performed. Note that in setting up the parties public keys, we can also compute hk_AP and hk_BP, which can resist the attack on small subgroup. Where, h is a co-factor defined in P1363[19].

Hardware design and implementation for ECC: The current state of global communication coupled with global economy requiring the global prime movers to be in contact at all times with the business point is driving the world of wireless communication powered by embedded constrained environment devices technology to a new high. But the world of embedded constrained systems with its limited power, memories and storage capabilities is limiting effective implementation of better crypto-algorithms. However, with the advent of ECC coupled with AES-a symmetric key crypto, the world of embedded constrained devices is beginning to enjoy better crypto-security network connections, thereby allowing secure business transactions over-the-air on the fly.

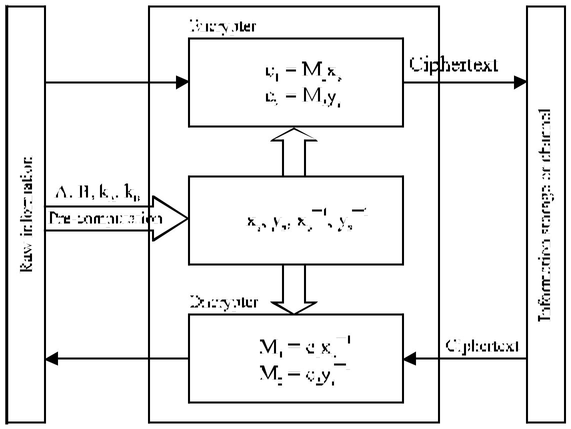

Fig. 8: Encryption and decryption schemes when the encrypter and decrypter are sitting aside

The elliptic curve crypto-schemes offer the highest security per bit ratio compared to any other currently known public-key cryptosystems. This is a plus point for embedded systems where the cost increases significantly with every extra memory chip. ECC hardware implementation uses fewer transistors, e.g., currently implementation of 155-bit ECC uses only 11,000 transistors compared to RSA 512-bits implementation with 50,000 transistors.

The elliptic curve crypto-schemes with key size in the range of 155-210 bits is currently considered acceptable for practical crypto-security protocols which compares favorably with similar security level as the Discrete Logarithm Problem (DLP) and IFP crypto-schemes with the key size of 512-1024 bits. Thus, ECC is superior to DLP/IFP cryptosystems in terms of block length and storage requirements. Our practical implementation is based on the key size of 163-bits where $GF(2^{163})$. The procedure required for successful implementation of hardware based protocols is as follows:

- To minimize the hardware complexity (circuitry complexity, feedback loops and the number of XOR-gates), the irreducible polynomial, $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$, was chosen.
- The integers a, b $\in GF(2^{163})$ required for the obtaining the elliptic curve, E: $y^2+xy = x^3+ax^2+b$ were chosen arbitrarily as:

$a =$ 4 FEEE CE35 5F00 D401 875F E2C0 FFF3 EE2F 64FC B30D
$b =$ 1 31C8 335B 7F5A D1C5 EEA0 8316 0E10 C9C8 C3B3 D39A

- With a and b chosen as above, there could be a possible solution to the elliptic curve E, which yields an arbitrary fixed point $P = (x_P, y_P)$ on the curve:

$x_P =$ 5 5968 5D73 DD75 4D33 7FF7 3319 706C FF36 8212 25AF
$y_P =$ 3 6F4D 089A 3FA0 C2CB F84E 0D6E FCB4 1033 362A 5B97

Which we use as the generator of the point of order #E the number of points on the curve.

- At this point and regardless of the computing complexity at this step, sender chooses her private k_A and looks up at the receivers public-key $B = k_BP$ with k_B being its private key which are given by:

$k_A = 95$ and $k_B = 250$ both from [1, n-1], respectively.

- The encrypter entity A computes, $S_{AB} = k_A(k_BP) = k_{AB}P = (x_S, y_S)$, where:

$x_S =$ 4 A0AF DCE6 AFB4 2309 D155 BFBA 9D49 6178 07B5 885D.
$y_S =$ 459E 2437 4FB9 1C44 D5F4 42E3 57E2 BB32 F6B6 91C9

- The ciphertext messages are then computed as: $c_1 = M_1x_S$ and $c_2 = M_2x_S$ where M_1 and M_2 are consecutive blocks of which the data length is m-bits (163 bits), in this case we have a single block:

$M = M_1 + M_2 =$ 33E1 3366 8832 5566 5CCA + 3FE1 3CE6 8832 5577 5CEA

$c_1 = M_1x_S =$ 1EF DB1F 61D8 AF59 AD11 F520 723C F6C2 AC96 364C
$c_2 = M_2x_S =$ 5 AD61 C6F5 869E BAC1 AB1E F1F1 C272 8FD4 6567 380E

- The encrypted messages are next feed and stored in the array of Flash ROMs, which can be part of an Integrated Chip (IC) module or can be on a separate module.

The last two operations can be repeated many times until all the data messages are encrypted and stored (or sent). To recover the encrypted messages, the receiver performs the following operations:

- The shared key agreement $S_{BA} = k_B(k_AP) = k_{BA}P = (x_S, y_S)$ is already pre-computed as above.
- Step (1) and (2) could be omitted since the encrypter and decrypter are in the same module. From above $(x_P)^{-1}$ and $(y_P)^{-1}$ can be computed as:

$(x_P)^{-1} =$ 5 6468 AAC3 D57A 35E0 E19F BF40 5F1B 18F1 C861 6C5E
$(y_P)^{-1} =$ 7 223D BA66 4DAA 8495 603B C053 B084 2A3A 82C0 5CAE

- Subsequently:

$M_1 = c_1(x_S)^{-1} =$ 33E1 3366 8832 5566 5CCA

$M_2 = c_2(y_S)^{-1} =$ 3FE1 3CE6 8832 5577 5CEA

The last operation is to be repeated until all the plaintext is recovered from the transmitted ciphertext messages in case of large volume message.

The hardware design considerations: The elliptic curve ElGamal cryptosystems discussed above can be implemented in hardware using a simplified circuit in which the encrypter and the decrypter circuit modules are integrated aside within the same IC hardware. Therefore, x_S, y_S, $(x_S)^{-1}$ and $(y_S)^{-1}$ can be pre-computed in advance and stored in the registers. The encrypter task is then just to compute:

$$c_1 = M_1x_S \text{ and } c_2 = M_2y_S$$

At the receiving end, the decrypter has then to compute:

$$c_1(x_S)^{-1} = M_1x_S(x_S)^{-1} = M_1 \text{ and } c_2(y_S)^{-1} = M_2y_S(y_S)^{-1} = M_2$$

To recover the original plaintext. The complete operation is illustrated in the schematic hardware design concept shown in Fig. 8.

The advantage of this coupled encrypter-decrypter architecture approach is that only minimum hardware is required, such that a single fix-coefficient multiplier can be used to perform encryption or decryption depending on the mode of operation the chip is programmed to undertake. The multiplier can also be used in half duplex mode to perform an alternate encryption or decryption. Note that multiplication is performed over Galois field, therefore, the selection of the irreducible polynomial can affect the circuitry complexity. In practice, select the one that has small number of terms (pentanomial, or 5-terms). The IC module capability can be improved using additional circuitry capable of solving elliptic curve equations, inversion and multiplication, thereby enabling it to generate/change the private and/or public keys on-the-fly (Fig. 3 and 4). This additional circuitry no longer allows for a simple circuit and has added complexity which requires VLSI technology which by today's standard is not difficulty to achieve. Hardware based elliptic curve cryptosystems have already been implemented under various constrains and applications. Sutiko *et al*.[50] have implemented the ElGamal ECC processor in 32 bit-wide bus with 155 bit-wide register. Paar[51] looked at the implementation of the multiplier which is applicable to a composite field.

CONCLUSIONS

In this study we have shown how to implement elliptic curve cryptosystems over binary finite field, $\mathbb{F}_{2^m}$ using polynomial basis representation and; normal and optimal normal basis representations. We have shown that the elliptic curve crypto-schemes offer the highest security per bit ratio compared to any other currently known public-key cryptosystems. This is a plus point for embedded systems where the cost increases significantly with every extra memory chip. ECC hardware implementation uses fewer transistors, e.g., currently implementation of 155-bit ECC uses only 11,000 transistors compared to RSA 512-bits implementation with 50,000 transistors. Elliptic curve crypto-schemes offer a lot of promise in terms of security and memory requirement than any other present crypto-schemes. However, more research is needed in this field for better understanding and effective implementation of ECC.

REFERENCES

1. Rabah, K., 2004. Data security and cryptographic techniques: A review. ITJ., 3: 106-132.
2. Data Encryption Standard, 1977. Federal Information Processing Standard (FIPS) Publication 46, National Bureau of Standards, US., Washington D.C.
3. Rabin, M.O., 1979. Digital signature and public-key functions as intractable as factorization. MIT Laboratory of Computer Science, Technical Report, MIT/LCS/TR-212.
4. Rivest, R., A. Shamir and L. Adleman, 1978. A method for obtaining digital signatures and public key cryptosystems. Comm. ACM., 21: 120-126.
5. Menezes, A., P. Van Oorschot and S. Vanstone, 1997. Handbook of Applied Cryptography. CRC Press.
6. Koblitz, N., 1987. Elliptic curve cryptosystems. Mathematics of Computation, 48: 203-209.
7. Miller, V., 1986. Use of elliptic curves in cryptography. In: CRYPTO '85, pp: 417-426.
8. Rabah, K., 2005. Theory and implementation of elliptic curve cryptography. J. Applied Sci., 5: 604-633.
9. Koblitz, N., 1994. A Course in Number Theory and Cryptography. Springer Verlag.
10. Lenstra, A.K. and E.R. Verheul, 1999. Selecting cryptographic key sizes. J. Cryptol., J. Intl. Assoc. Cryptol. Res., Vol: 12.
11. Menezes, A.J., 1993. Elliptic Curve Public Key Cryptosystems . Kluwer Academic Publishers.
12. IEEE P1363a: http://grouper.ieee.org/groups/1363/P1363a/
13. ANSI X9.62, 1999. The Elliptic Curve Digital Signature Algorithm (ECDSA). American Bankers Association.
14. Rabah, K., 2005. Implementation of elliptic curve diffie-hellman and EC encryption schemes. ITJ., 4: 132-139.
15. Rabah, K., 2005. Elliptic curve elgamal signature and encryption schemes. (In Press).
16. Rabah, K., 2005. Theory and implementation of data encryption standard: A review. (In Press).
17. US, Department of Commerce/National Institute of Standards and Technology, 1997. Announcing request for candidate algorithm nominations for the Advanced Encryption Standard (AES). World Wide Web. http://csrc.nist.gov/encryption/aes/pre-round1/aes_9709.htm
18. ISO/IEC: http://grouper.ieee.org/groups/1363/Research/Other.html
19. IEEE P1363, 2000. Standard Specifications for Public-key Cryptography.
20. National Institute of Standards and Technology, 1999. Recommended elliptic curves for federal government use. http://csrc.nist.gov/encryption.
21. Schroeppel, R., H. Orman, S. O'Malley and O. Spatscheck, 1995. Fast Key Exchange with Elliptic Curve Systems. Advances in Cryptology-Crypto '95, LNCS 963, Springer-Verlag, pp: 43-56.
22. Schroeppel, R., C. Beaver, R. Gonzales, R. Miller and T. Draelos, 2002. A low-power design for an elliptic curve digital signature chip. Presented at Cryptographic Hardware and Embedded Systems (CHES).
23. De Win, E., A. Bosselaers, S. Vandenberghe, P. De Gersem and J. Vandewalle, 1996. A Fast Software Implementation for Arithmetic Operations in GF(2^n). Advances in Cryptology-ASIACRYPT '96, LNCS 1163, Springer-Verlag, pp: 65-76.
24. Lenstra, A. and E. Verheul, 2000. Selecting Cryptographic Key Sizes. In: Imai, H. and Y. Zheng, Eds., PKC 2000, volume LNCS 1751, Berlin, Springer-Verlag.
25. Bailey, D. and C. Paar, 2001. Efficient arithmetic in finite field extensions with application in elliptic curve cryptography. J. Cryptol., Vol: 14.

26. Woodbury, A., D.V. Bailey and C. Paar, 2002. Elliptic Curve Cryptography on Smart Cards Without Coprocessors. In: CARDIS 2000, Bristol, UK, September 20th, Kluwer.
27. Beth, T. and F. Schaefer, 1991. Non-Supersingular Elliptic Curves for Public Key Cryptosystems. LNCS 547, Advances in Cryptology-Eurocrypt '91, Brighton, United Kingdom, Davies, D.W. (Ed.), Springer-Verlag, pp: 252-266.
28. Itoh, T. and S. Tsujii, 1988. A fast algorithm for computing multiplicative inverses in GF(2^m) using normal basis. Inform. Comput. J., 78: 171-177.
29. Wu, H., 1999. Low Complexity Bit-parallel Finite Field Arithmetic Using Polynomial Basis. In: Koc, C. and C. Paar, Eds., Workshop on Cryptographic Hardware and Embedded Systems (CHES '99), volume LNCS 1717. Springer-Verlag.
30. Paar, C., P. Fleischmann and P. Soria-Rodriguez, 1999. Fast arithmetic for public-key algorithms in galois fields with composite exponents. IEEE Trans. Computers, 48: 1025-1034.
31. Orlando, G. and C. Paar, 2000. An efficient squaring architecture for GF(2^m) and its applications in cryptographic systems. Elect. Lett., 36: 1116-1117.
32. Weil's Conjecture, Proved by Helmut Hasse in 1934.
33. Silverman, J.H., 1985. The Arithmetic of Elliptic Curves. Springer-Verlag.
34. Georg-Johann, L. and Z. Horst-Günter, 1994. Constructing Elliptic Curves with given Group Order over Large Finite Fields. LNCS 877, Algorithmic Number Theory, The 1st Intl. Symposium, ANTS-I, Ithaca, Springer-Verlag, pp: 250-263.
35. Seroussi, G., 1998. Compact representation of elliptic curve points over GF(2^m). Research Manuscript, Hewlett-Packard Laboratories.
36. Lenstra, H.W.Jr. and R. Schoof, 1987. Primitive normal basis for finite fields. Math. Comput., 48: 217-231.
37. Agnew, G.B., T. Beth, R.C. Mullin and S.A. Vanstone, 1993. Arithmetic operation in GF(2^m). J. Cryptol., 6: 3-13.
38. Mullin, R.C., I. Onyszchuk, S.A. Vanstone and R. Wilson, 1989. Optimal normal basis in GF(p^n). Discrete Applied Math., 22: 149-161.
39. Rabah, K., Secure Implementing Message Digest. Authentication and Digital Signature, In Press.
40. Diffie, W. and M.E. Hellman, 1966. New directions in cryptography. IEEE Tran. Inform. Theory, 22: 644-654.
41. Bellovin, S.M. and M. Merritt, 1992. Encrypted key exchange: Password-based protocols secure against dictionary attacks. Proc. 1992 IEEE Computer Soc. Conf. Res. Security and Privacy, pp: 72-84.
42. Pollard, J.M., 1978. Monte Carlo methods for index computation (mod p). Math. Comput., 32: 918-924.
43. Hellman, M. and J. Reyneri, 1983. Fast Computation of Discrete Logarithms in GF(q). In: Advances in Cryptology: Proc. Crypto '82. Plenum Press.
44. Lim, C. and P. Lee, 1997. A key recovery attack on discrete log-based schemes using a prime order subgroup. Advances in Cryptology-Crypto '97, Lecture Notes in Computer Science, 1294: 249-263.
45. Koyama, K., U.M. Maurer, T. Okamoto and S.A. Vanstone, 1992. New Public-key Scheme Based on Elliptic Curves over the Ring Zn. LNCS 576, Advances in Cryptology-Crypto '91, Barbara, S. and J. Feigenbaum (Eds.), Springer-Verlag, California, pp: 252-266.
46. ANSI key schemes FIPS 140-2. http://cio.doe.gov/Publications/profile2000/Profile 2000_AppendixA.htm
47. Basham, L., D. Johnson and T. Polk, 1999. Representation of Elliptic Curve Digital Signature Algorithm (ECDSA) keys and signatures in internet X.509 public-key infrastructure certificates. Internet Draft, http://www.ietf.org, pp: 252-266.
48. ElGamal, T., 1985. A public-key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Infor. Theory, 31: 469-472.
49. Diffie, W., P.V. Oorschot and M. Wiener, 1992. Authentication and authenticated key exchanges. Designs, Codes and Cryptography, 2: 107-125.
50. Sutikno, S., R. Effendi and A. Surya, 1998. Design and implementation of arithmetic processor F2155 for elliptic curve cryptosystems. In: 1998 IEEE Asia-Pacific Conference on Circuits and Systems, pp: 647-650.
51. Paar, C., 1993. A parallel galois field multiplier with low complexity based on composite fields. 6th Joint Swedish-Russian Workshop on Information Theory, Mölle, Sweden, pp: 320-324.