华侨大学

毕业设计（论文）

题目： 椭圆曲线加密算法的C语言实现

院（系）	信息科学与工程学院
专　　业	电子信息工程
届　　别	07级
学　　号	0715212026
姓　　名	王正辉
指导老师	吴丽丽 讲师

华侨大学教务处印制

2011年5月

摘　要

随着计算机和互联网技术的不断发展、电子商务的广泛应用，信息安全问题变得越来越重要，而网络信息安全的核心在于密码技术。椭圆曲线密码体制(ECC)是一种公钥密码体制，相对于以往基于有限域上离散对数问题或大整数分解问题的传统公钥算法，椭圆曲线密码算法具有安全性高、速度快、密钥短、实现时所需占用资源少的特点。作为迄今为止每比特具有最高安全强度的密码系统，由于其算法的高效安全性，使其成为优于 RSA 的 PKI 体系的核心公钥算法，其 160 位 ECC 算法的安全性相当于 1024 位的 RSA 算法，而 210 位的 ECC 则相当于 2048 位的 RSA，所以 ECC 技术在信息安全领域中的应用将会越来越广泛。本设计正是基于这样的背景，在 Microsoft Visual Studio6.0 下的 Microsoft Visual C++6.0 编译环境中利用标准 C 语言并且借助密码学领域的开放源代码库 OpenSSL 设计与实现基于椭圆曲线的加密系统。

关键字：椭圆曲线， ECC， Microsoft Visual C++6.0， C 语言， OpenSSL，加密系统

ABSTRACT

With the development and application of information technology and the electronic commerce,the problem of information security becomes more and more important,but the network information security core lies in the password technology.Elliptic Curve Cryptography(ECC) systems which is a public-key systems is characterized by higher safety property,faster speed,shorter key lengths and fewer computational resources for implementation thanother former traditional public-key algorithms based on the discrete logarithm infinite fields or the great integer factorization problem.So far,the ECC provides the highest strength-per-bit of any cryptosystem known.Because of its high efficiency of the algorithm,some people think it is the best public-key cryptosystem that is suitable for current use in future.The security of 160-bit ECC is equal to 1024-bit RSA,and the security of 210-bit ECC is equal to 2048-bit RSA.So the application of ECC technology in the field of information security will be more and more widely.Based on this background,This design will use C language with open source library OpenSSL of the field of cryptography to design and realize a complete system of Elliptic curve encryption algorithm.

KEY WORD： Elliptic curve， ECC， Microsoft Visual C++6.0， C language，
OpenSSL， Encryption algorithm system

目　录

第一章 引言

1.1 当前背景

随着计算机和通信技术的高速发展和广泛应用，大量敏感信息常常通过公共通信设施或计算机网络进行交换，特别是 Internet 的广泛应用、电子商务和电子政务的迅速发展，越来越多的个人信息需要严格保密，如：银行账号、个人隐私等，社会的信息化程度越来越高，社会对计算机和网络系统的依赖性也越来越大。因此，如何保护信息的安全使之不被窃取及不至于被篡改或破坏，已成为当今被普遍关注的重大问题。而目前针对网络的攻击手段更是五花八门，攻击者可以窃取网络上的信息，如用户口令、数据库信息等；伪造用户身份、否认自己的签名，这对网络的进一步推广和应用构成了严重的威胁。

为了保护网络通信的安全，促进电子商务和电子政务的快速发展。因此，对网络通信，特别是电子商务中的安全技术进行研究，发展更为安全的加密技术已成为一个迫在眉睫的课题。近几十年来，很多密码学专家和学者致力于密码学的研究，一些安全理论已被广泛应用，为推动网络通信、电子文档的安全传输作出了巨大的贡献。

1.2 密码学现状

密码技术是一门古老的技术，目前已发展成为一门结合数学、计算机科学、电子与通信、微电子和量子物理等技术的交叉学科，使用密码技术不仅可以保证信息的机密性，而且可以保证信息的完整性和确定性，防止信息被篡改、伪造和假冒。

信息安全服务依靠安全机制来完成，而安全机制主要依赖于密码技术，因此密码技术是信息安全的核心。而密码算法又是密码技术的核心。所以密码算法是保障信息安全的核心之核心，其重要性不言自明。

为此，世界各国对密码算法的研制都高度重视，1977 年美国国家标准技术研究所 NIST(National Institute of Standards and Technology)提出数据加密标准 DES(Data Encrytion Standard)，随后多种密码算法相继出现，这些算法有：

RIJINDAEL、MARS、RC6、Twofish、Serpent、IDEA(International Data Encryption Algorithm)、CS2Cipher、MMB、SKIPJACK、Karn 等对称密码算法以及背包公钥密码算法、RSA、EIGamal、椭圆曲线密码算法 ECC(Elliptic Curve CryPtosystem)、NTRU 等非对称密码算法。

以上这些算法随着计算机技术的不断发展有些已经遭到了破译，不断完善和加强密码算法的安全强度，同时又能够为信息产品降低成本的密码技术已经成为当今密码领域的重中之重。

1.2.1 对称密码

对称密码早已被人们使用了数千年，它有各种形式：从简单的替换密码到较复杂的构造方式。对称密码的加密密钥能够从解密密钥中推算出来，同时解密密钥也可以从加密密钥中推算出来。而在大多数的对称算法中，加密密钥和解密密钥是相同的。所以也称这种加密算法为秘密密钥算法或单密钥算法。它要求发送方和接收方在安全通信之前，商定一个密钥。对称算法的安全性依赖于密钥，泄漏密钥就意味着任何人都可以对他们发送或接收的消息解密，所以密钥的保密性对通信性至关重要。

对称系统通常非常快速，却易受攻击，因为用于加密的密钥必须与需要对消息进行解密的所有人一起共享。其优缺点如下：

优点：

(1) 算法实现的效率高、速度快。

(2) 满足大量信息的加密要求。

缺点:

(1) 密钥量问题。在单钥密码系统中，每一对通信者就需要一对密钥，当用户增加时，必然会带来密钥量的成倍增长，因此在网络通信中，大量密钥的产生、存放和分配将是一个难以解决的问题。

(2) 密钥分发问题。单钥密码系统中，加密的安全性完全依赖于对密钥的保护，但是由于通信双方使用的是相同的密钥，人们又不得不相互交流密钥，所以为了保证安全，人们必须使用一些另外的安全通道来分发密钥，例如用专门的信使来传送密钥。这种做法的代价是相当大的，甚至可以说是非常不现实的，尤其在计算机网络环境下，人们使用网络传送加密

的档，却需要另外的安全通道来分发密钥。所以传统的对称密码算法急需改进。

常用的对称加密算法有 DES、DEA 和 AES。

DES 算法是由 IBM 公司开发，并被美国国家标准局于 1977 年 2 月采纳作为“非密级”应用的一个标准，此后，DES 成为全世界使用最广泛的加密标准。DES 算法加密时把明文以 64bit 为单位分成块，采用美国国家安全局精心设计的 8 个 S 盒（S：Substitution）和 P 置换（P：Permutation)，经过 16 轮迭代，最终产生 64 比特密文，每轮迭代使用的 48 比特子密钥由原始的 56 比特产生。DES 的加密与解密的密钥和流程完全相同，区别仅仅是加密与解密使用的子密钥序列的施加顺序正好相反。

DES 算法在历史上曾发挥重要作用，但也存在以下问题：

(1) DES 密钥空间的规模 256 对实际安全而言太小。

(2) DES 的密钥存在弱密钥、半弱密钥和互补密钥。

(3) DES 里的所有计算，除去 S 盒，全是线性的。S 盒的设计对密码算法的安全性至关重要。然而，美国国家安全局并没有公布 s 盒的设计原则，因此，有人怀疑 S 盒罩隐藏了“陷门（trapdoors）”，如果是这样，美国国家安全局就能轻易地解密消息。

此外，由于 DES 的密钥空间小，针对 DES 算法进行穷举攻击就可以取得成功。在 1998 年 7 月，电子前沿基金会(EFF)使用一台 25 万美元的计算机在 56 小时内破译了 DES 密钥。1999 年 1 月 RSA 数据安全会议期间，EFF 通过遍布全世界的 10 万台计算机的协同工作，用 22 小时 15 分钟就宣告破解了一个 DES 的密钥。 可见，在强大的计算机系统面前，DES 的安全性面临极大的挑战。

为了增强 DES 算法的安全性，密码设计者又提出了基于 DES 的 3DES（Triple DES）、独立子密钥方法和推广的 GDES 算法等。这些改变有些作用不大，有些还削弱了 DES 的安全性。

高级加密标准 AES(The Advanced Encryption Standard)是美国国家标准技术研究所(NIST)旨在取代 DES 的 21 世纪的加密标准。AES 的基本要求是：采用对称分组密码体制，密钥长度的最少支持为 128、192、256，分组长度 128 位，算法应易于各种硬件和软件实现。1998 年 NIST 开始 AES 第一轮分析、测试和征集，

共产生了 15 个候选算法。1999 年 3 月完成了第二轮 AES2 的分析、测试。2000 年 10 月 2 日美国政府正式宣布选中比利时密码学家 Joan Daemen 和 Vincent Rijmen 提出的一种密码算法 RIJNDAEL 作为 AES。

1.2.2 公钥密码

由于单钥密码系统存在难以解决的缺点，因此发展一种新的、更有效，更先进的密码体制显得更为迫切和必要。在这种情况下，出现了一种新的公钥密码体制，它突破性地解决了困扰着无数科学家的密钥分发问题。这一全新的思想是本世纪 70 年代，美国斯坦福大学的两名学者 Diffie 和 Hellman 提出的，该体制与单钥密码最大的不同是：在公钥密码系统中，加密和解密使用的是不同的密钥（相对于对称密钥，人们把它叫做非对称密钥），这两个密钥之间存在着相互依存关系：即用其中任一个密钥加密的信息只能用另一个密钥进行解密。这使得通信双方无需事先交换密钥就可进行保密通信。其中加密密钥和算法是对外公开的，人人都可以通过这个密钥加密档然后发给收信者，这个加密密钥又称为公钥；而收信者收到加密档后，它可以使用他的解密密钥解密，这个密钥是由他自己私人掌管的，并不需要分发，因此又称为私钥，这就解决了密钥分发的问题。

用抽象的观点来看，公钥密码就是一种陷门单向函数。我们说一个函数 f 是单向函数，即若对它的定义域中的任意 x 都易于计算 y=f(x)，而当 f 的值域中的 y 为已知时要计算出 x 是非常困难的。若当给定某些辅助信息（陷门信息）时则易于计算出 x，就称单向函数 f 是一个陷门单向函数。公钥密码体制就是基于这一原理而设计的，将辅助信息（陷门信息）作为秘密密钥。这类密码的安全强度取决于它所依据的问题的计算复杂度。

公钥密码体制的思想并不复杂，而实现它的关键问题是如何确定公钥和私钥及加、解密的算法。我们假设在这种体制中，Pk 只是公开信息，用作加密密钥，称为公钥，而 Sk 需要由用户自己保密，用作解密密钥，称为私钥。加密算法 E 和解密算法 D 也都是公开的。虽然 Pk 与 Sk 只是成对出现，但却不能根据 Pk 计算出 Sk。它们须满足条件：

(1) 加密密钥 Pk 对明文 M 加密后，再用解密密钥 Sk 解密，即可恢复出明文，或写为：M=D(Sk,E(Pk,M))。

(2) 加密密钥不能用来解密，即 M≠D(Pk,E(Pk,M))。

(3) 在计算机上可以容易地产生成对的 Pk 和 Sk。

(4) 从已知的最实际上不可能推导出 Sk。

(5) 加密和解密的运算可以对调，即：M=E(Pk,D(Sk,M))。

综上所述，公开密钥密码体制下，加密密钥不等于解密密钥。加密密钥可对外公开，使任何用户都可将传送给此用户的信息用公开密钥加密发送，而该用户唯一保存的私人密钥是保密的，也只有他能将密文复原、解密。虽然解密密钥理论上可由加密密钥推算出来，但这种算法设计在实际上是不可能的，或者虽然能够推算出，但要花费很长的时间而成为不可行的。所以将加密密钥公开也不会危害密钥的安全。

这种体制思想是简单的，但是，如何找到一个适合的算法来实现这个系统却是一个真正困扰密码学家们的难题，因为既然 Pk 和 Sk 是一对存在着相互关系的密钥，那么从其中一个推导出另一个就是很有可能的，如果敌手能够从 Pk 推导出 Sk，那么这个系统就不再安全了。因此如何找到一个合适的算法生成合适的 Pk 和 Sk，并且使得从 Pk 中不可能推导出 Sk，正是迫切需要密码学家们解决的一道难题。这个难题甚至使得公钥密码系统的发展停滞了很长一段时间。为了解决这个问题，密码学家考虑了数学上的陷门单向函数，根据关于陷门单向函数的思想，学者们提出了许多种公钥加密的方法，它们的安全性都是基于复杂的数学难题。根据所基于的数学难题，至少有以下三类系统目前被认为是安全和有效的：即大整数因子分解系统(代表性的有 RSA)、离散对数系统(代表性的有 DSA(Digital Signature Algoritlun)) 和 椭 圆 曲 线 离 散 对 数 系 统 (ECC(Elliptic Curve Cryptosystems))。

RSA 公钥加密算法是 1977 年由 Ron Rivest、Adi Shamirh 和 Len Adleman 在（美国麻省理工学院）开发的。RSA 取名来自开发他们三者的名字。RSA 是第一个能同时用于加密和数字签名的算法，是被研究得最广泛、目前最有影响力的公钥加密算法，已被 ISO 推荐为公钥数据加密标准。RSA 算法基于一个十分简单的数论事实：将两个大素数相乘十分容易，但那时想要对其乘积进行因式分解却极其困难，因此可以将乘积公开作为加密密钥。

RSA 的算法涉及三个参数，n、e1、e2。其中，n 是两个大质数 p、q 的积，n 的二进制表示时所占用的位数，就是所谓的密钥长度。e1 和 e2 是一对相关的值，

e1 可以任意取，但要求 e1 与(p-1)*(q-1)互质；再选择 e2，要求(e2*e1)mod((p-1)*(q-1))=1。(n 及 e1),(n 及 e2)就是密钥对。RSA 加解密的算法完全相同,设 A 为明文，B 为密文：则加密算法为：B=A^e2 mod n；解密算法为：A=B^e1 mod n=A^(e1*e2) mod n。e1 和 e2 可以互换使用，即：A=B^e2 mod n；B=A^e1 mod n。

数字签名（又称公钥数字签名、电子签章）是一种类似写在纸上的普通的物理签名，但是使用了公钥加密领域的技术实现，用于鉴别数字信息的方法。一套数字签名通常定义两种互补的运算，一个用于签名，另一个用于验证。数字签名了的档的完整性是很容易验证的，而且数字签名具有不可抵赖性。简单地说,所谓数字签名就是附加在数据单元上的一些数据,或是对数据单元所作的密码变换。这种数据或变换允许数据单元的接收者用以确认数据单元的来源和数据单元的完整性并保护数据,防止被人(例如接收者)进行伪造。它是对电子形式的消息进行签名的一种方法,一个签名消息能在一个通信网络中传输。基于公钥密码体制和私钥密码体制都可以获得数字签名,目前主要是基于公钥密码体制的数字签名。包括普通数字签名和特殊数字签名。普通数字签名算法有 RSA、ElGamal、Fiat-Shamir、Guillou- Quisquarter、Schnorr、Ong-Schnorr-Shamir 数字签名算法、DES/DSA,椭圆曲线数字签名算法和有限自动机数字签名算法等。数字签名（Digital Signature）技术是不对称加密算法的典型应用。数字签名的应用过程是，数据源发送方使用自己的私钥对数据校验和或其他与数据内容有关的变量进行加密处理，完成对数据的合法“签名”，数据接收方则利用对方的公钥来解读收到的“数字签名”，并将解读结果用于对数据完整性的检验，以确认签名的合法性。

DSA(Digital Signature Algorithm)是一种数字签名算法，它是 Schnorr 和 ElGamal 签名算法的变种，被美国 NIST 作为数字签名标准(DSS(Digital Signature Standard))。它是基于整数有限域离散对数难题的，其安全性与 RSA 相比差不多。DSA 的一个重要特点是两个素数公开，这样，当使用别人的 p 和 q 时，即使不知道私钥，你也能确认它们是否是随机产生的，还是作了手脚。RSA 算法却作不到。

人们对椭圆曲线的研究已有 100 多年的历史，而椭圆曲线密码是 Neal Koblitz 和 Victor Miller 于 1985 年提出来的。目前椭圆曲线密码已成为除 RSA 密码之外呼声最高的公钥密码之一。它可以提供同 RSA 相同的功能。然而它的安全性建立

在椭圆曲线离散对数问题(ECDLP)的困难性之上。现在求解 ECDLP 的最好算法具有全指数时间复杂度，与此不同，整数因子分解问题却有亚指数时间算法，所谓压指数算法就是其算法复杂度还未到达指数级别。这意味着要达到期望的安全强度，椭圆曲线密码可以使用较 RSA 密码更短的密钥。普遍认为 160 位的椭圆曲线密码可提供相当于 1024 位 RSA 密码的安全强度。由于密钥短，所以工程实现加解密速度较快，并且可节省能源、带宽和存储空间。正因为如此，一些国际标准化组织已把椭圆曲线密码作为新的信息安全标准。由于椭圆曲线密码具有上述优点，因此椭圆曲线密码特别适于在航空、航天、卫星及智能卡的系统中的应用。在椭圆曲线密码体制的标准化方面，IEEE、ANSI、ISO、FIPS、SEC 等都作了大量的工作，它们所开发的椭圆曲线标准的文档有：IEEE P1363-2000 和 P1363a、ANSI X9.62 和 X9.63、 ISO/IEC15946、FIPS186-2、SEC1 和 SEC2 等。

公钥密码体制的特点为：

(1) 通信双方不需要事先交换密钥，保密性好。

(2) 可以实现数字签名，保证传输信息的不可否认性。

(3) 算法基于数学难题，加密效率非常低。

(4) 不适合大量信息的加密需求。

(5) 密钥管理、信任关系确立需要权威机构支持。

根据研究，以目前计算整数分解问题、离散对数问题和椭圆曲线离散对数问题的最好算法进行计算，表 1-1 比较了 RSA、DSA 和 ECC 在等价安全强度下所需的密钥尺寸(其中 MIPS(Million Instructions Per Second)年是每秒运算 100 万条指令运行 1 年的时间):

表1-1 三种典型的公钥密码体系性能比较

破解密钥时间（MIPS 年）	RSA、DSA 密钥长度（bit）	ECC 密钥长度（bit）	RSA 与 ECC 密钥长度比
104	512	106	5:1
108	768	132	6:1
1011	1024	160	7:1
1020	2048	210	10:1
1078	21000	600	35:1

表 1-1 说明，在等价安全强度下，ECC 较 RSA 和 DSA 的密钥尺寸小的多，而且随着密钥长度的增加它们的比例将更为悬殊。

以下从几个方面对三种公钥密码体系进行比较：

（1）安全性能

加密算法的安全性能一般通过该算法的抗攻击强度来反映。ECC 和其他几种公钥系统相比，其抗攻击性具有绝对的优势。椭圆曲线的离散对数计算困难性(ECDLP)在计算复杂度上目前是完全指数级，而 RSA 是亚指数级的。这体现 ECC 比 RSA 的每 bit 安全性能更高。

（2）计算量和处理速度

在一定的相同的计算资源条件下，虽然在 RSA 中可以通过选取较小的公钥(可以小到 3)的方法提高公钥处理速度，即提高加密和签名验证的速度，使其在加密和签名验证速度上与 ECC 有可比性，但在私钥的处理速度上(解密和签 名)，ECC 远比 RSA，DSh 快得多。因此 ECC 总的速度比 RSA，DSA 要快得多。同时 ECC 系统的密钥生成速度比 RSA 快百倍以上。因此在相同条件下，ECC 有更高的加密性能。

（3）存储空间

ECC 的密钥尺寸和系统参数与 RSA、DSA 相比要小得多。160 位 ECC 与 1024 位 RSA，DSA 具有相同的安全强度，210 位 ECC 则与 2048 位 RSA，DSA 具有相同的安全强度。意味着它所占的存贮空间要小得多。这对于加密算法在资源受限环境上(如智能卡等)的应用具有特别重要的意义。

（4）带宽要求

当对长消息进行加解密时，三类密码系统有相同的带宽要求，但应用于短消息时 ECC 带宽要求却低得多。而公钥加密系统多用于短消息，例如用于数字签名和用于对称系统的会话密钥传递。带宽要求低使 ECC 在无线网络领域具有广泛的应用前景。

1.3 本设计的主要内容

本设计的目的是在 Microsoft Visual Studio6.0 下的 Microsoft Visual C++6.0 编译环境中利用 C 语言并借助功能非常强大的开放源代码库 OpenSSL 设计与实现椭圆曲线加密和解密算法。关于算法加解密功能的验证将采用结合 VC++6.0 的 dos 窗口打印和文本文档的记录方式。

第二章 椭圆曲线概述

2.1 有限域

在椭圆曲线密码体制中，我们关心的是基于有限域上的椭圆曲线。有限域上的椭圆曲线加密算法涉及数论、群论、有限域理论及椭圆曲线等数学知识。

域是对常见的数系(如有理数域 Q，实数域 R 和复数域 C)及其基本特性的抽象。域由一个集合 F 及两种运算共同组成。这两种运算分别为加法(用+表示)和乘法(用*表示)，且满足下列算术特性：

（1）(F，+)对于加法运算构成加法交换群，单位元用 0 表示。

（2）(F\{0}，*)对于乘法运算构成乘法交换群，单位元用 l 表示。

（3）分配律成立：对于所有的 a，b，c∈F，都有(a+b)*c=a*c+b*c。若集合 F 是有限集合，则称此域为有限域。

有限域算术运算的有效实现是实现椭圆曲线密码的重要先决条件，因为椭圆曲线运算是基于有限域算术运算的。椭圆曲线密码体制的有效实现主要用到两大特征的有限域，它们分别是素域和二进制域。

2.1.1 素域

设 p 是一个素数，以 p 为模，则模 p 的全体余数的集合{0,1,2,...,p-1}关于模 p 的加法和乘法构成一个 p 阶有限域，并用符号 Fp 表示。我们称 p 为 Fp 的模。对于任意的整数 a，a mod p 表示用 p 除 a 所得到的余数 r，0≤r≤p-1,并称这种运算为求模 p 的剩余。Fp 中元素具有以下算术运算：

（1） 加法：如果 a,b∈Fp，则 a+b=r，其中 r 是被 p 除所得的剩余，O≤r≤P 一 1。

（2） 乘法：如果 a,b∈Fp，则 a*b=s，其中 s 是被 p 除所得的剩余，O≤s≤P 一 1。

（3） 求逆：如果 a 是 Fp 中的非零元素，a 模 q 的逆元，记为a^{-1}，是唯一的整数 c ∈Fp，c 满足 a*c=1。

2.1.2 二进制域

阶为2^m的域称为二进制域或特征为 2 的有限域。构成F_{2^m}的一种方法是采用多项式基表示法。在这种方法中，域F_{2^m}的元素是次数最多为 m-1 次的二进制多项式(多项式的系数取自F_2={0，1})，F_{2^m} 的公式见公式 2-1。

$$F_{2^m}=\{a_{m-1}z^{m-1}+a_{m-2}z^{m-2}+\cdots+a_2z^2+a_1z+a_0 : a_i \in \{0.1\}\} \quad \text{公式2-1}$$

选择一个二进制域上的 m 次既约多项式 f(z)（对任意的 m，这样的多项式一定存在，并能够有效产生）f(z)的既约性意味着 f(z)不能被分解成次数低于 m 的二进制域上的多项式的乘积。域元素的加法是两个多项式系数的模 2 相加。域元素的乘法是两个多项式相乘后取模 f(z)。对于任意一个二进制域上的多项式 a(z)，a(z) mod f(z)表示一个次数低于 m 的余式多项式 r(z)。余式多项式 r(z)可用 f(z)辗转相除 a(z)得到，这一运算成为求模 f(z)的余式。

2.2 射影平面和无穷远点

通常情况下两条平行直线永不相交,但可以假设两条平行直线相交于一个 无穷远点，从而可以在原来平面坐标系的基础上建立射影坐标系。

对于普通平面直角坐标系上的点坐标(x，y)做如下变换：

令 x=X/Z，y=Y/Z，Z≠0，则 A 点可以表示为(X，Y，Z)。这样就将二维的坐标变成了三维的坐标，同时，对于平面上的点建立了一个新的坐标体系。同时得到直线的方程 aX+bY+cZ=0(普通平面直角坐标系下直线一般方程是 ax+by+c=0)。无穷远点是两条平行直线的交点，将两条平行线直线对应的方程联立求解：

$$aX+bY+c_1Z=0 \qquad \text{公式 2-2}$$

$$aX+bY+c_2Z=0 \qquad \text{公式 2-3}$$

解得$c_1Z=c_2Z=-aX-bY$,因为$c_1 \neq c_2$，所以 Z=0，即无穷远点可以用(X，Y，0)表示。注意，平常点 Z≠0，无穷远点 Z=0，因此无穷远直线对应的方程是 Z=0。

下面列出无穷远点的几个性质，以统一平行与相交：

（1）直线工上的无穷远点只能有一个。

（2）平面上一组相互平行的直线有公共的无穷远点。

（3） 平面上任何相交的两直线厶，厶有不同的无穷远点。
（4） 平面上全体无穷远点构成一条无穷远直线。
（5） 平面上全体无穷远点与全体平常点构成射影平面。

2.3 椭圆曲线

2.3.1 椭圆曲线定义

域 K 上的椭圆曲线 E 的方程定义见公式2-4：

$$E: y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \qquad \text{公式2-4}$$

其中 $a_1, a_2, a_3, a_4, a_6 \in K$ 且 $\Delta \neq 0$，Δ 是 E 的判别式，具体定义如下:

$$\Delta = -d_2^2 d_8 - 8d_4^3 - 27d_6^2 + 9d_2 d_4 d_6;$$
$$d_1 = a_1^2 + 4a_2;$$
$$d_2 = a_1^2 + 4a_2;$$
$$d_4 = 2a_4 + a_1 a_3;$$
$$d_6 = a_3^2 + 4a_6;$$
$$d_8 = a_1^2 a_6 + 4a_2 a_6 - a_1 a_3 a_4 + a_2 a_3^2 - a_4^2;$$

方程 E 被称为 Weierstrass 方程；我们称 E 是域 K 上的椭圆曲线，这是因为系数 a_1, a_2, a_3, a_4, a_6 均为域 K 的元素；条件 $\Delta \neq 0$ 是确保椭圆曲线是“光滑”的，即曲线的所有点都没有两个或两个以上不同的切线。

椭圆曲线的形状，并不是椭圆的，椭圆曲线的一个典型图形如图 2-1 所示。只是因为椭圆曲线的描述方程，类似于计算一个椭圆周长的方程。定义在实数域 R 上的椭圆曲线 $E_1: y^2 = x^3 - x$，$E_2: y^2 = x^3 + \frac{1}{4}x + \frac{5}{4}$，其点 $E_1(R)\backslash\{\infty\}$ 和 $E_2(R)\backslash\{\infty\}$ 画在图 2-1 中（其中 ∞ 为无穷远点）：

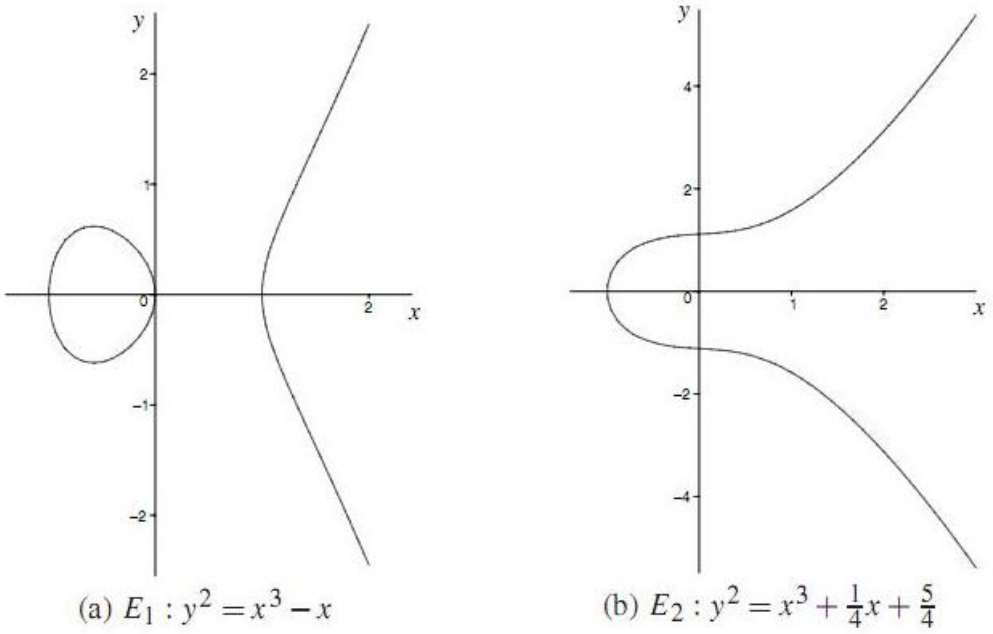

图2-1 R 上的椭圆曲线

若 E 是定义在域F_q上的椭圆曲线，那么椭圆曲线 E（F_q）的点的个数记为#E（F_q）,并称其为域F_q上的椭圆曲线 E 的阶。

若椭圆曲线上的一点 P，存在最小的正整数 n 使得 nP=∞，则称 n 为 P 的阶。若 n 不存在，我们说 P 是无限阶的。事实上，在有限域上定义的椭圆曲线上所有的点的阶 n 都是存在的。

2.3.2 椭圆曲线的运算法则

设 P 和 Q 是椭圆曲线 E 上的两个的点，若是不同的两点，则 L 是 P 和 Q 的连线；若是重合的点，则 L 是该点的切线；L 与曲线相交的另一个点R'，过R'

作 y 轴的平行线，与曲线交另一点 R，定义点加运算 R=P+Q。

不同点和重合点的点加运算的图形描述分别图 2-2 和图 2-3：

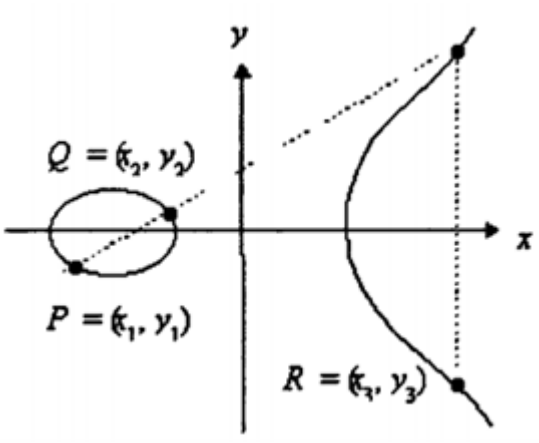

图2-2 不同点相加运算

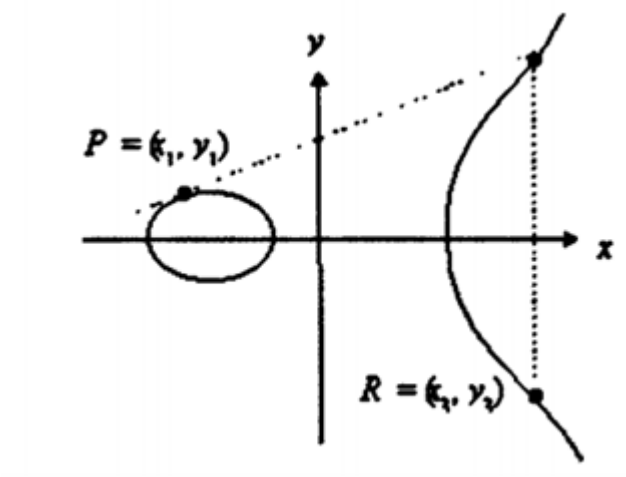

图2-3 重合点倍点运算

椭圆曲线的运算法则如下：

（1） 单位元。对于所有的 P∈E（K），P+∞=∞+P=P。∞点为单位元。

（2） 负元素。若 P=（x,y）∈E（K），则（x,y）+（x,-y）=∞。记点（x,-y）为-P，并称其为 P 的负。注意，-P 也是 E(K)上的一个点，此外-∞=∞。

（3）不同点相加。令 $P=(x_1,y_1)\in E(K)$， $Q=(x_2,y_2)\in E(K)$， $P\neq\pm Q$，则 $P+Q=(x_3,y_3)$。其中：

$$x_3=(\frac{y_2-y_1}{x_2-x_1})^2-x_1-x_2 \quad 和 \quad y_3=(\frac{y_2-y_1}{x_2-x_1})(x_1-x_3)-y_1$$

（4）相同点倍点。令 $P=(x_1,y_1)\in E(K)$， $P\neq -P$，则 $2P=(x_3,y_3)$。其中：

$$x_3=(\frac{3x_1^2+a}{2y_1})^2-2x_1 \quad 和 \quad y_3=(\frac{3x_1^2+a}{2y_1})(x_1-x_3)-2y_1$$

（5）标量乘。椭圆曲线上的标量乘是椭圆曲线密码体制的核心操作。曲线 E 上的一点 P，其标量乘 Q=KP 定义为对 P 进行了（K-1）次的倍点运算。标量乘是椭圆曲线密码体制中最耗时的运算，它决定着椭圆曲线密码体制的运算速度。

2.3.3 射影平面坐标系下的椭圆曲线

椭圆曲线在射影平面上的投影形式为：

$$Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3 \qquad \text{公式 2-5}$$

曲线 E 上唯一的一个无穷远点是（0:1:0）。射影坐标的点加和倍点运算不涉及域上的求逆。首先把点转化为仿射坐标，然后利用其运算法则进行点加，最后去掉分母，便得到射影坐标点的计算公式。

由于在普通平面直角坐标系下，域 K 上的求逆存在比乘法费时的情况，那么采用射影坐标来表示点将是非常好的一种解决办法。

2.4 密码学中的椭圆曲线

椭圆曲线密码体制基于的数学难题是椭圆曲线离散对数问题，前面提及的椭圆曲线是连续的，并不适合加密，所以我们必须把椭圆曲线转化为离散的点，这就需要将椭圆曲线定义在有限域上。

并不是所有的椭圆曲线都适合加密。$y^2 = x^3 + ax + b$ 是一类可以用来加密的椭圆曲线，也是最为简单的一类，本设计正是基于有限域上的这类曲线来实现加解密算法的。下面我们就把 $y^2 = x^3 + ax + b$ 这条曲线定义在有限域 Fp 上：

选择两个满足下列条件的小于 p(p 为素数)的非负整数 a、b 并且满足 $4a^3 + 27b^2 \neq 0 \pmod p$。则满足方程 $y^2 = x^3 + ax + b \pmod p$ 的所有点(x,y)，再加上无穷远点∞ ，构成一条椭圆曲线。其中 x,y 属于 0 到 p-1 间的整数，并将这条椭圆曲线记为 Ep(a,b)。

令 p=29，a=4，b=20，考虑定义在素域 F_{29} 上的椭圆曲线 E:

$$E: y^2 = x^3 + 4x + 20 \qquad \text{公式 2-6}$$

注意，$\Delta = -16(4a^3 + 27b^2) = -176896 \neq 0 \pmod{29}$，所以曲线 E 确实是椭圆曲

线。椭圆曲线 $E(F_{29})$ 的点如下：

∞	(2,6)	(4,19)	(8,10)	(13,23)	(16,2)	(19,16)	(27,2)
(0,7)	(2,23)	(5,7)	(8,19)	(14,6)	(16,27)	(20,3)	(27,27)
(0,22)	(3,1)	(5,22)	(10,4)	(14,23)	(17,10)	(20,26)	
(1,5)	(3,28)	(6,12)	(10,25)	(15,2)	(17,19)	(24,7)	
(1,24)	(4,10)	(6,17)	(13,6)	(15,27)	(19,13)	(24,22)	

椭圆曲线在不同的数域中会呈现出不同的样子，但其本质仍是一条椭圆曲线。椭圆曲线在有限域上的运算法则同实数域上的运算法则类似。

椭圆曲线离散对数问题（ECDLP）定义为:给定定义于有限域 Fq 的椭圆曲线 E，基点 $P \in E(F_q)$，阶为 n，点 $Q \in \langle P \rangle$,寻找一个整数 $l \in [0, n-1]$ 使得 $Q = lP$。整数 l 称为 Q 的基于 P 的离散对数，表示为 $l = \log_P Q$。

椭圆曲线离散对数问题的困难性是所有椭圆曲线密码方案安全性的基础。但需要注意的是，ECDLP 的难解并没有数学上的证明。目前还没有证明不存在求解 ECDLP 的有效算法。然而，通过很多年的考验说明了 ECDLP 的难解性。自从 1985 年椭圆曲线密码被提出来，该问题被许多学者研究，到目前为止还没有发现亚指数时间的通用算法。

第三章 椭圆曲线加解密算法实现

3.1 椭圆曲线的参数选取

3.1.1 参数组 D

为了抵抗所有已知的对 ECDLP 的攻击，密码方案中使用的椭圆曲线的参数应该谨慎选择。最简单的求解 ECDLP 问题的算法是穷举搜索:连续计算一系列点 P,2P,3P,4P,...,直到得到 Q，该方法最坏的情况下为 n 步，平均情况下为 n/2 步。因此可以通过选择参数 n 足够大的椭圆曲线，使得穷举攻击的计算量不可实际实现（如 $n \geq 2^{80}$）。对于 ECDLP，已知最好的通用攻击方法是将 Pohlig-Hellman 算法和 Pollard's rho 相结合，该方法的运行时间为完全指数时间 $O(\sqrt{p})$，其中 p 是 n 的最大素因子。为了抵抗该种攻击，椭圆曲线参数的选择要求 n 含有大素数因子 p， p 应该足够大，以使得 $\sqrt{p}$ 的计算量不可实现（如 $p > 2^{160}$）。另外椭圆曲线的参数选择还要防止其他所有的已知攻击方法。

密码学中，描述一条 F_p 上的椭圆曲线，参数组 D=(p ,FR，S,a,b,G,n,h)。其中: 域的阶 p 、系数 a 和 b 用来确定一条椭圆曲线；FR 即 Fp 中元素的表示；S 是随机生成椭圆曲线系数所需的种子；有穷远点 G 为基点，n 为点 G 的阶，h =$\#E(F_p)/n$，称为余因子。

为了避免针对 ECDLP 的 Pohlig-Hellman 攻击和 Pollard's 攻击，$\#E(F_p)$ 必须能被足够大的素数 n 整除，最小应该保证 $n > 2^{160}$。给定域 F_p，为了最大限度地抵抗 Pohlig-Hellman 攻击和 Pollard's 攻击，应选择 E 使得 $\#E(F_p)$ 为素数或者近似素数，即 $\#E(F_p)$=hn，其中 n 为素数，h 非常小（如 h=1,2,3 或 4）。

这几个参量取值的选择，直接影响了加密的安全性。参量值一般要求满足以下几个条件：

（1） p 当然越大越安全，但越大计算速度越慢，200 位左右可以满足一般安全要求。

（2） p≠n×h。

（3）pt≠1 (mod n)，其中 1≤t<20。

（4） 4a3+27b2≠0 (mod p)。

（5） n 为素数。

（6） h≤4。

为了防止针对特殊类型椭圆曲线的攻击，应随机选择满足$\#E(F_p)$能被大素数整除的椭圆曲线 E。因为随机曲线满足已知的同构攻击的条件的概率非常小而可忽略，因此已知攻击仍可避免。可证明的随机选择一条椭圆曲线，这能保证椭圆曲线的用户不被通过故意构造有潜在弱点的曲线而窃取用户的私钥。

然而随机生成一条满足要求的椭圆曲线，有时需要的时间是很长的。在 FIPS 186-2 标准中，NIST 推荐了美国政府使用的 15 个不同安全级别的椭圆曲线。本设计正是使用了 NIST 推荐的F_p上的随机椭圆曲线 P-192。其参数组如下:

$p = 2^{192} - 2^{64} - 1$； a=-3; h=1;

S = 0x 0345AE6F C8422F64 ED579528 D38120EA E12196D5

r = 0x 3099D2BB BFCB2538 5A2DCD5F B078B6EF 5F3D6FE2 C745DE65

b = 0x 64210519 E59C80E7 0FA7E9AB 72243049 FEB8DEEC C146B9B1

n = 0x FFFFFFFF FFFFFFFF FFFFFFFF 99DEF836 146BC9B1 B4D22831

x = 0x 188DA80E B03090F6 7CBF20EB 43A18800 F4FF0AFD 82FF1012

y = 0x 07192B95 FFC8DA78 631011ED 6B24CDD5 73F977A1 1E794811

其中 p：素域F_p的阶。

S：随机生成椭圆曲线系数所需的种子。

r：单向函数 SHA-1 的输出。

a,b :椭圆曲线 $y^2 = x^3 + ax + b$ 的系数，满足 $rb^2 \equiv a^3 (\mathrm{mod}\ p)$。

n：基点 G 的（素数）阶。

h：余因子。

x,y：基点 G 的 x 和 y 坐标。

3.1.2 密钥对

椭圆曲线密钥对与参数组 D 中的一系列参数相关。其选择过程为:随机选择 $d \in [1, n-1]$ 作为私钥,然后计算 Q=dG 作为公钥。从公钥 Q 中计算私钥 d 的问题（$d = \log_G Q$）显然就是椭圆曲线离散对数问题。因此至关重要的是参数组 D 的选择要使得 ECDLP 不可求解。此外数 d 的生成要是“随机”的，以防止攻击者基于概率搜索策略获得额外信息。

3.2 椭圆曲线加解密算法流程

椭圆曲线本身并不是一种密码算法，但可以将已知密码算法移植到椭圆曲线上。这里选择的是 ELGamal 算法的变种。

ELGamal 是一种基于有限域上离散对数问题的加密、签名算法，其安全性依赖于计算有限域上离散对数这一难题。

ElGamal 用于加密时，首先选择一个素数 p，两个随机数 g 和 x，g, x < p, 计算 $y = g^x \pmod p$，则其公钥为 y, g 和 p，私钥是 x。g 和 p 可由一组用户共享。被加密信息为 M，首先选择一个随机数 k，k 与 p - 1 互质，计算 $a = g^k \pmod p$，$b = y^k M \pmod p$，a 和 b 为密文，是明文的两倍长。解密时计算 $M = \frac{b}{a^x} \pmod p$。

椭圆曲线加解密算法采用 ELGamal 算法的变种，其流程如下:

（1）用户 A 选定一条椭圆曲线 Ep(a,b)，并取椭圆曲线上一点，作为基点 G。

（2）用户 A 选择一个私有密钥 k，并生成公开密钥 K=kG。

（3）用户 A 将 Ep(a,b)和点 K、G 传给用户 B。

（4）用户 B 接到信息后 ，将待传输的明文编码到 Ep(a,b)上一点 M，并产生一个随机整数 r(r<n)，用户 B 计算点 C1=M+rK；C2=rG。

（5）用户 B 将 C1、C2 传给用户 A。

（6）用户 A 接到信息后，计算 C1-kC2，结果就是点 M。因为:C1-kC2 = M+rK-k(rG) = M+rK-r(kG)=M 。

（7）再对点 M 进行解码就可以得到明文。

在这个加密通信中，如果有一个偷窥者 H ，他只能看到 Ep(a,b)、K、G、C1、C2 而通过 K、G 求 k 或通过 C2、G 求 r 都是相当困难的。因此，H 无法

得到 A、B 间传送的明文信息。

这一过程的图形描述如下图 3-1：

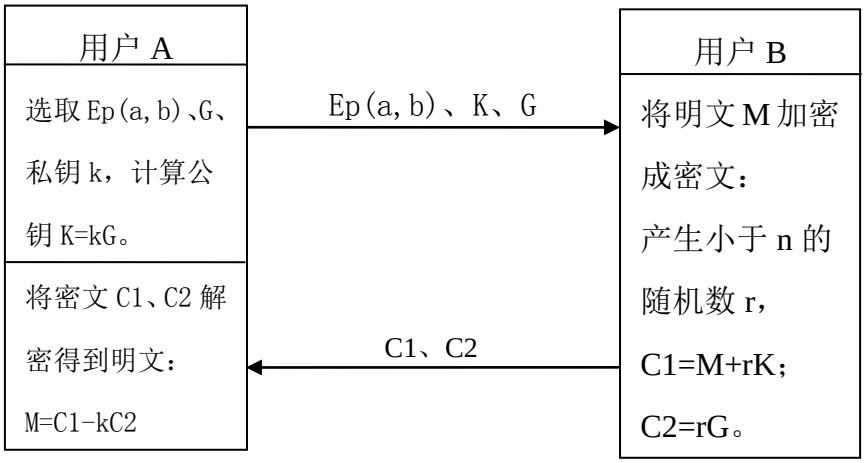

图3-1　ECC 加解密流程

3.3 开放源代码工具 OpenSSL 简介

密码算法几乎都是建立在高精度大整数的运算上，而一般的程序语言都不提供高精度大整数的结构，因此要表示上百位的高精度大整数就必须借助于特殊的含有大整数结构的密码学函数库。

目前，密码学领域的函数库已经有很多了，主要有 cryptlib、Crypto++、GNU MP、Libgcrypt、LiDIA、MIRACL、NTL、OpenSSL 和 PARI-GP 等。这些密码函数库在密码学应用领域发挥着巨大的作用，它们并不都是开放源代码，其中有些是需要收费的。基于各方面的考量，本设计采用 OpenSSL 函数库。

OpenSSL 是一个开放源代码的 SSL 协议的产品实现，它采用 C 语言作为开发语言，具备了跨系统的性能。OpenSSL 项目最早由加拿大人 Eric A. Yang 和 Tim J. Hudson 开发，现在由 OpenSSL 项目小组负责改进和开发，这个小组是由全球的一些技术精湛的志愿技术人员组成，他们的劳动都是无偿的，在此应该向他们表示崇高的敬意。

OpenSSL 最早的版本在 1995 年发布，1998 年后开始由 OpenSSL 项目组维护和开发。它完全实现了对 SSLv1、SSLv2、SSLv3 和 TLS 的支持。OpenSSL 的源代码库可以从 OpenSSL 的官方网站 www.openssl.org 自由下载，并可以免费用于任何商业或非商业的目的。由于采用 C 语言开发，OpenSSL 的源代码库具有良好的跨平台性能，支持 Linux、Unix、Windows、Mac 和 VMS 等多种平台。目前，OpenSSL

已经得到了广泛的应用，许多类型的软件中的安全部分都使用了 OpenSSL 的库，如 VOIP 的 OpenH323 协议、Apache 服务器、Linux 安全模块等。我们有理由预期 OpenSSL 和其所倡导的开放源码的思想必将被众多的支持者发扬光大。

虽然 OpenSSL 使用 SSL 作为其名字的重要组成部分，但其实现的功能却远远超出了 SSL 协议本身。OpenSSL 事实上包括了三部分：SSL 协议、密码算法库和应用程序。

SSL 协议部分完全实现和封装了 SSL 协议的三个版本和 TLS 协议，SSL 协议库的实现是在密码算法库的基础上实现的。使用该库，你完全可以建立一个 SSL 服务器和 SSL 客户端。该部分在 Linux 下编译会形成一个明文 libssl.a 的库，在 Windows 下则是名为 ssleay32.lib 的库。

密码算法库是一个强大完整的密码算法库，它是 OpenSSL 的基础部分，也是很值得一般密码安全技术人员研究的部分，它实现了目前大部分主流的密码算法和标准。主要包括公开密钥算法、对称加密算法、散列函数算法、X509 数字证书标准、PKCS12、PKCS7 等标准。事实上，OpenSSL 的 SSL 协议部分和应用程序部分都是基于这个库开发的。目前，这个库除了可以使用本身的缺省算法外，在 0.9.6 版本之后，还提供了 Engine 机制，用于将如加密卡这样外部的加密算法实现集成到 OpenSSL 中。密码算法库在 Linux 编译后其库文件名称为 libcrypto.a，在 Windows 下编译后其库文件为 libeay32.lib。

应用程序部分是 OpenSSL 最生动的部分，也是 OpenSSL 使用入门部分。该部分基于上述的密码算法库和 SSL 协议库实现了很多实用和范例性的应用程序，覆盖了众多的密码学应用。主要包括了各种算法的加密程序和各种类型密钥的产生程序(如 RSA、Md5、Enc 等等)、证书签发和验证程序（如 Ca、X509、Crl 等）、SSL 连接测试程序（如 S_client 和 S_server 等）以及其它的标准应用程序（如 Pkcs12 和 Smime 等）。在某些时候，不需要做二次开发，仅仅使用这些应用程序便能得到我们的应用要求，比如采用 Ca 程序就能基本上实现一个小型的 CA 功能。这些应用程序同时也是很好的使用 OpenSSL 加密算法库和 SSL 协议库的优秀例子，比如 Ca、Req 和 X509 程序就是使用 OpenSSL 的库开发一个 CA 中心服务器的优秀例子，又如 S_client 和 S_server 程序就是利用 SSL 协议库建立 SSL 安全连接的优秀例子。对于初学者来说，研读这些应用程序的源码通常是最好的入门途径

与其它的一些同类型密码库相比，OpenSSL 具有以下优点：

（1）采用 C 语言开发，支持多种操作系统，可移植性好。

（2）功能全面，支持大部分主流密码算法、相关标准协议和 SSL 协议。

（3）开放源代码，可信任，能够根据自己的需要进行修改，对技术人员有借鉴和研究的价值。

（4）具备应用程序，既能直接使用，也可以方便进行二次开发。

（5）免费使用，能够用于商业和非商业目的。

OpenSSL 的一些缺点：

（1）采用非面向对象的 C 语言开发，对于初学者来说有一定的困难，也不利于代码的剥离。

（2）文档不全面，增加了使用的困难性。

但总的来说，OpenSSL 是一个非常优秀的软件包，很值得密码安全技术人员研究和使用。

OpenSSL 里的大数表示:

```
struct bignum_st
{
    BN_ULONG *d;/* Pointer to an array of 'BN_BITS2' bit chunks. */
    int top;  /* Index of last used d +1. */
    /* The next are internal book keeping for bn_expand. */
    int dmax;     /* Size of the d array. */
    int neg;  /* one if the number is negative */
    int flags;
};
```

其中:

d : BN_ULONG（因系统而异，Win32 下为 4 个字节）数组指针首地址，大数就存放在这里面，但是是倒着放的。比如:要存放的大数为 12345678000（通过 BN_bin2bn 放入），则 d 的内容为:0x30 0x30 0x30 0x38 0x37 0x36 0x35 0x34 0x33 0x32 0x31。

top : 用来指明大数占多少个 BN_ULONG 空间，上例中 top 为 3。

dmax : d 数组的大小。

neg : 1 是负数；0 是正数。

flags : 用来存放有些标记，比如 flags 含有 BN_FLG_STATIC_DATA 时，表明 d 的内存是静态分配的；含有 BN_FLG_MALLOCED 时，d 的内存是动态分配的。

OpenSSL 的大数函数如下:

(1) BN_rand/BN_pseudo_rand

生成一个随机的大数。

(2) BN_rand_range/BN_pseudo_rand_range

根据给出随机数的范围，生成随机数。

(3) BN_dup

大数复制。

(4) BN_generate_prime

生成素数。

(5) int BN_add_word(BIGNUM *a, BN_ULONG w)

给大数 a 加上 w，成功则返回 1。

(6) BIGNUM *BN_bin2bn(const unsigned char *s, int len, BIGNUM *ret)

将内存中的数转化为大数。s 为内存地址，len 为数据长度，ret 为返回值。

(7) int BN_bn2bin(const BIGNUM *a, unsigned char *to)

将大数 a 转化为内存形式。to 为输出缓冲区地址。缓冲区需预先分配，返回值为缓冲区长度。

(8) char BN_bn2dec(const BIGNUM *a)

将大数 a 转化为整数字符串。返回值为存放整数字符串的地址，它由内部空间分配，用户必须在外部使用 OPENSSL_free 函数释放该空间。

(9) char *BN_bn2hex(const BIGNUM *a)

将大数 a 转化为十六进制字符串。返回值为生成字符串的地址，外部需要 OPENSSL_free 函数释放该空间。

(10) BN_cmp

比较两个大数。

(11) BIGNUM *BN_mod_inverse(BIGNUM *in,const BIGNUM *a,const BIGNUM *n, BN_CTX *ctx)

计算 ax=1(mod n)。

OpenSSL 不仅提供了高精度大整数的表示和运算，还提供了各种密码算法的实现。OpenSSL 中涉及椭圆曲线算法的一些结构和库函数将在下面的内容中给出介绍。

3.4 基于 OpenSSL 的椭圆曲线加解密算法实现

本设计的编译环境是 Microsoft Visual Studio6.0 下的 Microsoft Visual C++6.0，并将 OpenSSL 链接到 Microsoft Visual C++6.0 中。关于 OpenSSL 在 windows 下的编译，以及 Microsoft Visual C++6.0 中如何使用 OpenSSL 将附录中给出。

OpenSSL 中关于椭圆曲线的密钥数据结构如下:

```
struct ec_key_st
{
    int version;
    EC_GROUP *group;
    EC_POINT *pub_key;
    BIGNUM      *priv_key;
    unsigned int enc_flag;
    point_conversion_form_t conv_form;
    int   references;
    EC_EXTRA_DATA *method_data;
} /* EC_KEY */;
```

其中 priv_key 为密钥，定义为一个大数；pub_key 为公钥，定义为一个曲线点。group 是椭圆曲线参数组，里面包含椭圆曲线系数 a,b、基点 G、阶 n 和余因子 h 等，定义为结构体 EC_GROUP。

3.4.1 密钥对的生成

椭圆曲线密钥对的生成实现在 OpenSSL 目录下的 crypto/ec/ec_Key.c 中，密钥对的生成首先需要选定一条椭圆曲线(在 OpenSSL 的 crypto/ec_curve.c 中内置了 67 种椭圆曲线，可用 EC_get_builtin_curves 获取该列表)，然后根据选择的椭圆曲线计算密钥生成参数，最后根据密钥参数来生成密钥对。

nid=NID_X9_62_prime192v1; // 曲线参数名

b = EC_KEY_new_by_curve_name(nid); // 根据指定的椭圆曲线生成密钥参数

EC_KEY_generate_key(b); // 根据密钥参数生成 ECC 公私钥

生成的公私钥后对公私钥的表示:

EC_KEY_get0_public_key(b)为公钥;

EC_KEY_get0_private_key(b)为私钥。

3.4.2 OpenSSL 中关于椭圆曲线的主要函数

(1) EC_get_builtin_curves

获取曲线列表

(2) EC_KEY_new_by_curve_name

根据指定的椭圆曲线来生成密钥参数

(3) int EC_KEY_generate_key

根据密钥参数生成 ECC 公私钥

(4) int EC_KEY_check_key

检查 ECC 密钥

(5) int ECDSA_size

获取 ECC 密钥字节大小

(6) ECDSA_sign

数字签名，成功则返回 1

(7) ECDSA_verify

验证签名，合法则返回 1

(8) EC_KEY_get0_public_key

获取公钥

(9) EC_KEY_get0_private_key

获取私钥

(10) ECDH_compute_key

生成共享密钥

3.4.3 软件流程图

椭圆曲线各参数生成流程图如图 3-2:

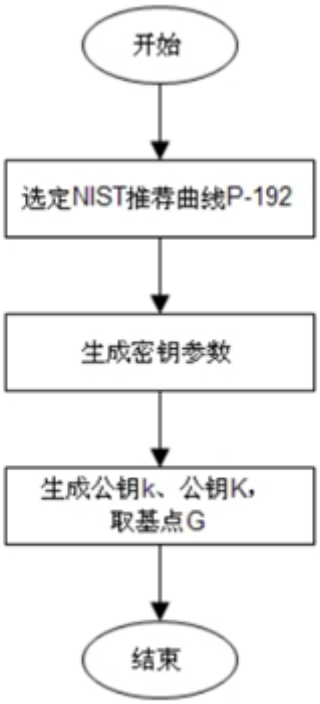

图3-2 椭圆曲线各参数生成流程图

加密流程图和解密流程图分别如图 3-3 和图 3-4:

图3-3 加密流程图

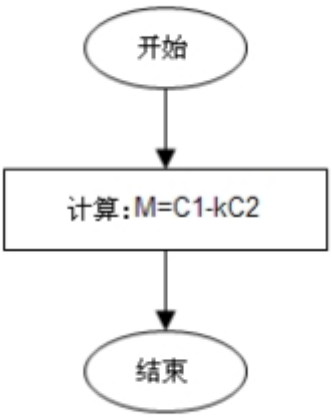

图3-4 解密流程图

整体的软件流程图如图 3-5:

图3-5 系统软件流程图

第四章 加解密功能的验证

Microsoft Visual C++6.0下执行窗口如图3-6:

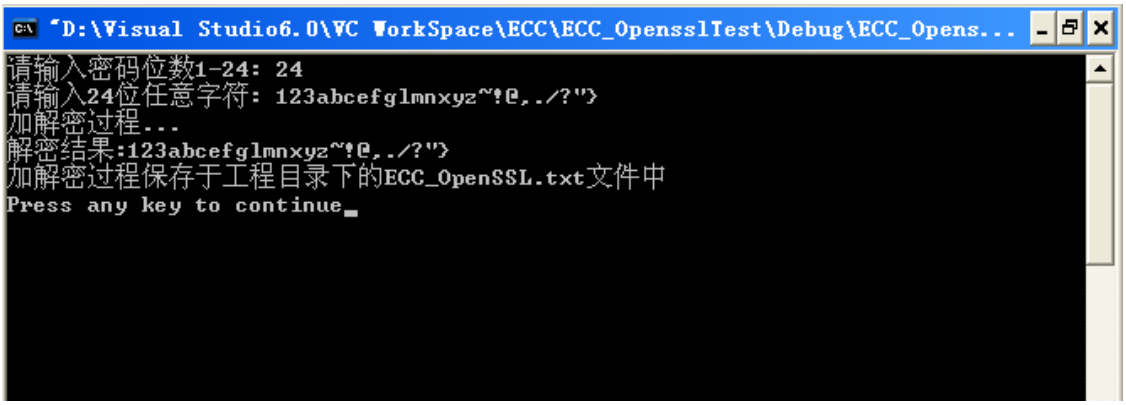

图3-6 VC++6.0执行窗口

加解密过程记录文件 ECC_OpenSSL.txt 文件内容如图3-7:

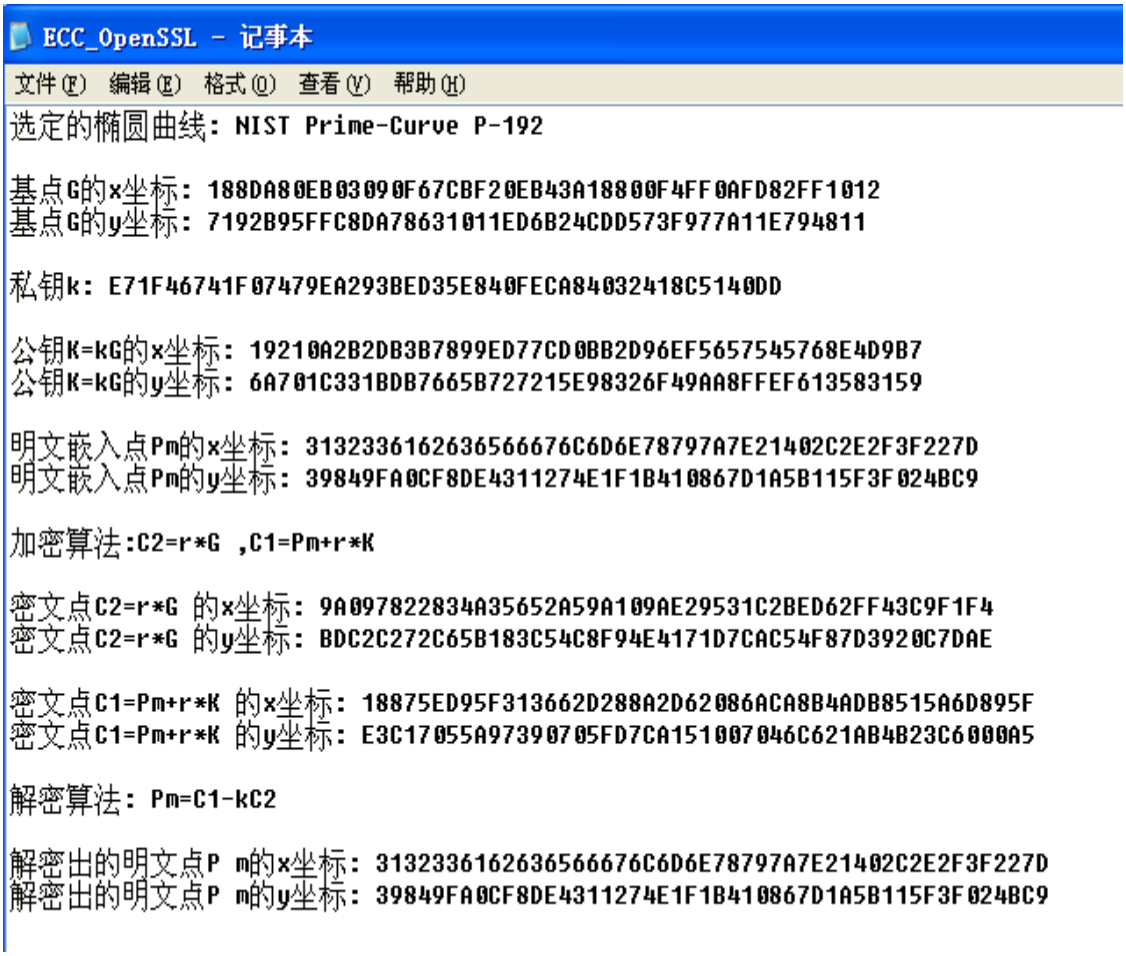

图3-7 加解密过程

密码123abcefglmnxyz~!@,./?"}的 ASCII 码经查表如下（以字符串形式）：313233616263656667 6C6D6E78797A7E21402C2E2F3F227D。

经过多次测试的结果验证了加解密功能的正确性。

第五章 结论和感想

本设计是借助 OpenSSL 在 Microsoft Visual C++6.0环境下用 C 语言实现椭圆曲线加解密算法。经验证，能够很好的完成其加解密算法的功能。从对密码体系的一窍不通到逐步了解的过程让我受益匪浅。椭圆曲线密码体系还被用于数字签名(ECDSA)和密钥交换(ECDH),也出现了不要求明文进行嵌入的加密算法 ECIES。本设计学到的东西将是我不断学习椭圆密码体制的基础。

致谢语

在这里，我要特别感谢郭荣新老师、吴丽丽老师和苏文龙同学给予的帮助。你们的支持是我完成本次设计源源不断的动力！

参考文献

1.Darrel Hankerson、Alfred Menezes、Scott Vanstone，张焕国等翻译.《椭圆曲线密码学导论》[M]．电子工业出版社，2000 年.

2.祝跃飞、张亚娟.《椭圆曲线公钥密码导论》[M]．电子工业出版社，2006 年.

3.胡冠章.《应用近世代数》（第二版）[M]．清华大学出版社，2004 年.

4.Bruce Schneier，吴世忠等翻译.《应用密码学(协议算法与 C 源程序) 》[M]．机械工业出版社，2000 年.

5.William Stallings.《密码编码学与网络安全---原理与实践》(第三版)[M]．电子工业出版社，2004 年.

6.谭浩强.《C 程序设计》（第三版）[M]．清华大学出版社，2005 年.

7.邓安文.《密码学-加密演算法》[M]．中国水利水电出版社，2006 年.

8.赵春平.《OpenSSL 编程》[J].OpenSSL 说明性文档.

9.张丞.《基于 AES 和 ECC 的混合密码体制研究及应用》[J].硕士论文.

10.武汉大学密码研究中心.http://www.eccsdk.com/bbs/.

附录

windows 下 OpenSSL 的安装及使用（以 Perl-5.8 和 openssl-0.9.8k 为例）

1、下载 Perl-5.8，安装后重启系统。

http://downloads.activestate.com/ActivePerl/Windows/5.8/ActivePerl-5.8.8.822-MSWin32-x86-280952.zip

2、下载 openssl 并解压到 C:\openssl-0.9.8k。

http://www.openssl.org/source/

参考 openssl 目录下的 install.win32 说明进行安装：

1、进入解压目录。

>cd C:\openssl-0.9.8k

2、运行 Configure。

>perl Configure VC-WIN32

如不成功会有明显提示。

3、创建 Makefile 文件。

>ms\do_ms

推荐使用这种方式，另外两种方式 如果使用也必须保证本机有编译器才能使用。

:ms\do_masm（默认 vc 自带的编译器；也也以自己下载安装）

:ms\do_nasm（需要自己下载）

4、配置 VC 环境变量。

>cd C:\Program Files\Microsoft Visual Studio\VC98\Bin

>vcvars32.bat

5、编译动态链接库。

>cd C:\openssl-0.9.8k

>nmake -f ms\ntdll.mak

如果编译成功，最后的输出都在 out32dll 目录下：包括可执行文件、两个 dll（ssleay32.lib, libeay32.lib）和两个 lib 文件（ssleay32.dll, libeay32.dll）。

6、为 VC 添加头文件和静态链接库路径。

Tools àOptions àDirectores，在 Include files 中增加 C:\openssl-0.9.8k \inc32 目录；在

Libray files 中增加 C:\openssl-0.9.8k\out32dll。

7、编写 OpenSSL 程序，可参考 C:\openssl-0.9.8k\demos

(1)包含相应头文件

#include<openssl/***.h>

(2)添加静态链接库

#pragmacomment(lib,"libeay32.lib")

#pragmacomment(lib,"ssleay32.lib")

或 Project àSettings àLink àObject\library modules 填写 libeay32.lib ssleay32.lib。

(3)将动态链接库 ssleay32.dll, libeay32.dll 复制到 C:\WINDOWS\system32 或 Debug 目录下，确保动态链接库在正确的路径。

英文文献翻译

Elliptic curve cryptosystems

Elliptic curve cryptosystems (ECCs) were developed independently in 1985 by Neal Koblitz and Victor Miller. ECCs do not use new cryptographic algorithms; they use existing algorithms; e.g., the ElGamal cryptosystem.What changes is the operation.

Recall that for the ElGamal cryptosystem the exponentiation operation is based upon multiplication modulo a prime p .

$$b = a^n \bmod p = a \cdot a \cdots a \bmod p \text{ (n factors a)}$$

For an ECC the exponentiation operation is based upon ⊗, the multiplication of points on an elliptic curve that is defined over a finite field. What does all this mean?

First, it should be noted that an ellipse is not an elliptic curve. Elliptic curves are related to integrands of elliptic integrals, and elliptic integrals first occurred in the calculation of the arc length of an ellipse. (Elliptic integrals have been explored since early in the Nineteenth Century.)

For ECCs, the elliptic curves are typically of the form $y^2 = x^3 + ax + b$ (with some conditions on a and b). To say that the curve is defined over a finite field just means that the input x (and the output y) come from a finite field. (See the appendix about finite fields.) So, instead of being a continuous curve, the ECC curve over a finite field is a collection of points.

But, these points have the property that two points on the curve can be multiplied (we will denote the operation ⊗), and the result is a point on the curve (i.e., the points on the curve form a group under ⊗).Elliptic curves have the property that most straight lines meet an elliptic curve in three points.So, given two points on an elliptic curve, draw a line through the points.That line should meet the curve in one point – the product. The fact that the points on an elliptic curve over a finite field form a group has been well- known to algebraic geometers.

For a given elliptic curve, an ECC based upon the ElGamal cryptosystem replaces

the usual multiplication with ⊗ and the usual exponentiation with

$$b = a^n \bmod p = a \otimes a \otimes \cdots \otimes a \bmod p$$

where a is a point on the elliptic curve. The algorithm is otherwise not changed. There is a little bit of work to do to implement the ECC however because it must be arranged that blocks of a plaintext message m can be converted to and from points on the curve. (See, for example, Klima, Sigmon, and Stitzinger, below.)

Why should ECC be considered?Since factoring and the discrete logarithm problem were implemented in public-key cryptosystems in the 1970s, much attention has been paid to the underlying mathematical problems. Steady progress has been made on factoring and on solving for discrete logarithms, but neither problem is solved. Cryptographers have countered mathematical successes by increasing key sizes. On the other hand, little progress seems to have been made on the elliptic curve discrete logarithm problem. Rather than continuing to increase key sizes, one can switch to "second generation" public-key cryptosystems using elliptic curves and use smaller keys.The NSA now recommends that vendors who incorporate public-key cryptosystems consider the use of ECC. In 2005 the NSA announced suite B which uses ECC. Suite B may be used for both unclassified and classified data.

References

http://www.certicom.com/index.php?action=ecc,home Certicom is a developer of ECCs. Their website has tutorials about elliptic curves and ECCs. Their discussion of adding points on an elliptic curve includes some applets that allow the operation to be visualized.

Tony Horowitz, "Elliptic Curves," The Journal of Undergraduate Mathematics and Its Applications, 8 (2), 1987. This is an excellent introduction to elliptic curves and adding points on an elliptic curve.

Richard Klima, Neil Sigmon, and Ernest Stitzinger, Applications of Abstract Algebra with MAPLE, CRC Press, 2000.Chapter 8 is a nice discussion of a simplified ECC.

椭圆曲线密码体制

椭圆曲线密码体制（ECCs）由 Neal Koblitz 和 Victor Miller 于 1985 年分别独立提出。ECCs 并没有使用新的密码学算法，而是使用已经存在的算法，例如:ElGamal 加密体系，改变的是运算。

回想一下，ElGamal 加密体系的指数运算是基于以素数 p 为模的乘法。

$$b = a^n \bmod p = a \cdot a \cdots a \bmod p \text{ (n factors a)}$$

ECC 的指数运算是基于 ⊗，它是定义在有限域上的椭圆曲线上的点的乘法。这究竟是什么意思呢？

首先，应当指出椭圆并不是椭圆曲线，椭圆曲线和椭圆积分的被积函数有关，椭圆积分最早是在计算椭圆的弧长时被发现的（早在十九世纪椭圆积分就已经被研究）。

对于 ECCs，椭圆曲线的典型形式是 $y^2 = x^3 + ax + b$ （a 和 b 满足某种条件）。说椭圆曲线是定义在有限域上意思是输入（和输出 y）来自一个有限域。所以，不同于连续的曲线，定义在有限域上的椭圆曲线是一个点的集合。

但是，这些点都有这样的性质:两个曲线上的点相乘（我们记为运算 ⊗）,它的结果是这个曲线上的另一点（这个点来自于 ⊗ 群）。椭圆曲线有一个性质：大部分的直线与椭圆曲线相交的点有 3 个。所以，给定两个椭圆曲线上的点，连接两点画一条直线。这条直线相交于椭圆曲线的另一个点—就是它的乘积。事实上，来自群的基于有限域的椭圆曲线上的点已经在代数几何中广为人知。

给定一条椭圆曲线，基于 ElGamal 加密体系的 ECC 用 ⊗ 代替了普通的乘法和指数运算:

$$b = a^n \bmod p = a \otimes a \otimes \cdots \otimes a \bmod p \text{ (n factors a)}$$

其中 a 是椭圆曲线上的一个点。该算法在其他方面并没有改变。但实现 ECC 仍然有一些工作要做，那是因为必须把明文块 m 嵌入到曲线上的一点。为什么要考虑到 ECC？自从 20 世纪 70 年因子分解和离散对数问题被实现在公钥密码体制中，相关的数学难题受到了更多的关注。因子分解和离散对数的求解已取得稳步进展，但是这两个问题都没有得到解决。密码学者反驳说可以通过增加密钥长度来对抗攻击。在另一方面，对于椭圆曲线离散对数问题的求解似乎没有什么进展。

不同于不断增加密钥长度，可以切换到第二代公钥密码体制使用椭圆曲线并且使用更短的密钥。现在美国国家安全局推荐成立公钥密码体制的信息工业产品厂家将 ECC 纳入参考。2005 年，现在美国国家安全局宣布 Suit B 使用 ECC。Suit B 可被用于非保密级数据和保密级数据。

参考文献

http://www.certicom.com/index.php?action=ecc,home

Tony Horowitz,“Elliptic Curves,”The Journal of Undergraduate Mathematics and Its Applications, 8 (2), 1987.

Richard Klima, Neil Sigmon, and Ernest Stitzinger, Applications of Abstract Algebra with MAPLE, CRC Press, 2000.