**第** **1** **章** **密码学概述**

密码学的历史极为久远，其起源可以追溯到远古时代，人类有记载的通信密码始于公元前 400年。虽然密码是一门古老的技术，但自密码诞生直至第二次世界大战结束，对于公众而言， 密码始终处于一种未知的黑暗中，常常与军事、机要、间谍等工作联系在一起，让人在感到神 秘之余，又有几分畏惧。信息技术的发展迅速改变了这一切。随着计算机和通信技术的迅猛发 展，大量的敏感信息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。正是这种对信息的秘密性与真实性的需求，密码学才逐渐揭去了神秘的面纱，走进公众的 日常生活中。

**1.1 玛丽女王的密码**

密码学是一门神秘而古老的学科，古今中外，刀光剑影，经常能看到密码学的影子。1586 年10月15日上午，玛丽女王走进佛斯林费堡挤满人的法庭。由于多年囚禁与风湿病的折磨， 她憔悴不堪，但她依旧高贵冷静地展现着帝王风范，在医生的协助下，缓缓走近位于这狭长审 判室中间的御座。玛丽以为这御座表示她赢得了应有的敬意。她错了。这御座代表缺席的伊丽 莎白女王 玛丽的仇敌与起诉人。玛丽被和缓地带离御座，走到审判室另一边的被告席上， 那把猩红色丝绒椅才是她的座位。

苏格兰的玛丽女王在此接受叛逆罪的审判。她被控密谋行刺伊丽莎白女王以夺取英格兰王 位。伊丽莎白的国务大臣弗朗西斯 ·沃尔辛厄姆爵士已捕获其他共犯，取得供词，并将他们处 决了。现在，他要证明玛丽是这宗阴谋的核心人物，一样罪当处死。

如图1-1所示，上方人物是伊丽莎白一世，下方头像是玛丽女王。



**Windows C/C++加密解密实战**



图1-1

这宗叛逆阴谋是一群年轻的英格兰天主教贵族策划的。他们意图除掉伊丽莎白这个新教徒， 让同为天主教徒的玛丽取而代之。法庭认为，玛丽显然是这群叛匪的名义领袖，但不确定她是 否承认这项罪责。事实上，玛丽的确授意了此次行动。沃尔辛厄姆面临的挑战是：他必须证实 玛丽和这群叛匪之间确有关系。

审判日当天，玛丽独坐在被告席上。被控叛逆罪的嫌犯不得请辩护律师，也不准传唤证人。 不过，玛丽还未身陷绝境 当初她很谨慎地用密码与叛匪通信，她用密码系统把信息转换成 一串无意义的符号。玛丽相信，就算沃尔辛厄姆搜出这些信件，也读不出什么名堂来。这些信 件的内容既然无解，也就不能成为呈堂证据。

不幸的是，沃尔辛厄姆不仅拦截到玛丽送给叛匪的信件，还知道谁能破解这些密码。托马 斯 ·菲利普是英格兰破解密码的第一高手。他若能破解玛丽授意叛匪谋逆的信息，她就难逃一 死了。

1542年11月24日，亨利八世于索维莫斯一役击溃苏格兰大军。亨利八世征服苏格兰、 夺取詹姆斯五世王位的野心，眼看就要实现了。经过这场战役，深受打击的苏格兰国王身心完 全崩溃，退居在福克兰的宫殿里。就连两周之后，女儿玛丽的诞生，也无法使这位病怏怏的国 王振作起来。他似乎就等着继承人诞生，确定责任已了，即可平静地离开人世。玛丽诞生一个

**第1章密码学概述**

星期后，年仅30岁的詹姆斯五世随即驾崩。1543年9月9日，9个月大的玛丽在斯特灵城堡 的礼拜堂接受加冕。正因玛丽女王太过年幼，英格兰反而暂缓侵犯苏格兰。亨利八世顾虑，若 于此刻出兵进犯一个新王只是女婴的国家，会被讥为没有骑士风度。因此，英格兰国王改用怀 柔政策，想安排玛丽与他的儿子爱德华成亲，借此将苏格兰纳入都铎王室的统治之下。

可是，苏格兰拒绝了亨利八世的提议。他们宁愿让玛丽和法国皇太子弗朗西斯缔结婚约。 1548年8月7日，6岁的玛丽前往法国。16岁时，玛丽与弗朗西斯完婚，并在1559年成为法 国王后。至此事事顺遂，玛丽似乎可以意气风发地返回苏格兰了。没想到，一向孱弱的弗朗西 斯病倒了。他在儿时感染的耳疾忽然恶化，发炎的部位扩散到脑部，引发脓疮。1560年，登 基未满一年的弗朗西斯撒手人寰，玛丽成为寡妇。

从此，玛丽陷入一场又一场的悲剧。1561 年，她回到苏格兰。在接下来的两场失败婚姻 中，玛丽女王被卷进衰败的旋涡。1567年，苏格兰的新教贵族对他们的天主教女王不再抱任 何希望，于是囚禁玛丽，强迫她让位给14个月大的儿子詹姆斯六世。1568年，玛丽逃出囚房， 向南朝英格兰走去，寄望于她的表姑伊丽莎白一世能提供庇护。

玛丽做了一个可怕的错误判断。伊丽莎白提供给她的不过是另一座监牢。玛丽遭受囚禁的 原因是她对伊丽莎白构成了威胁。玛丽的祖母玛格丽特 ·都铎是亨利八世的姐姐，所以她有权 继承英格兰王位，只不过亨利八世仅存的子嗣伊丽莎白的顺位排在她之前。但是英格兰的天主 教徒认为，英格兰的国君应该是信奉天主教的玛丽。

1586年1月6日，被囚禁了18年的玛丽惊愕地收到一批信。

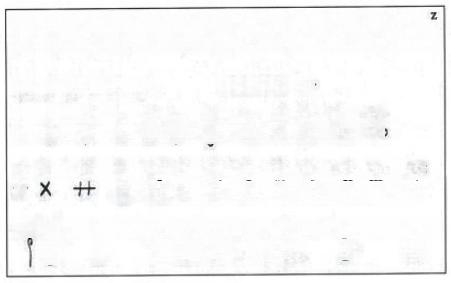
这些信来自玛丽的支持者，是吉尔伯特 ·基弗偷运进来的。基弗是天主教徒，1577年离 开英格兰，在罗马接受神学教育。他于1585年回到英格兰，急于为玛丽效劳。他来到位于伦 敦的法国大使馆，那里积放了一大批寄给玛丽的书信。基弗宣称他有办法把这些信件偷运进囚 禁玛丽的查特里宅邸，而他也真的办到了。这只是一个开始。基弗开始担任秘密信差，送信给 玛丽，并带出她的回信。他用皮革把信裹起来，再把包裹藏在封塞啤酒桶的空心木塞里。酿酒 商把酒送进查特里宅邸，玛丽的仆人打开木塞，取出藏在里面的信交给玛丽。将信息带出查特 里宅邸也是用同样的方法。

此时，一项营救玛丽女王的计划正在伦敦的一个酒馆里酝酿着。该计划的中心人物是天主 教贵族青年安东尼 · 贝平顿。这些谋逆分子一致认为，这项后来被称为“贝平顿阴谋”的计划 需要玛丽的首肯才能进行下去。问题是，他们找不到与她通信的途径。1586年7月6日，基 弗来到贝平顿的门前。他送来一封玛丽的信，说她在巴黎的支持者提到贝平顿，她很期待贝平 顿的来信。贝平顿随即写了一封长信，描述计划的轮廓。

一如既往，基弗把这封信放进啤酒桶的木塞里，蒙混过看守玛丽的人。贝平顿还采取了额 外的措施，把信转换成密码——万一密函被玛丽的看守人拦截，他也无法解读内容。他所用的 密码如图1-2所示。他用了23个符号来代替英文字母(不包括j、v、w), 另有36个符号来 代替单词或词组。此外，还有4个虚元(不代表任何字母，像空格一样不具备任何意义的符号), 以及一个重复符号，这个重复符号表示下一个符号代表两个相同的字母。



**Windows C/C++加密解密实战**

a0Ac 世daecθf g h1i6kxI1 mp⁵oM.pf△qrsEC8y9

虚元 ff. 一.一.d. 重复符号σ

and for with that if but where as of the from by

2344430 M 8 火

8 d 5

not when there this in wich is what say me my wyrt

06x 古 6 m 方 md

send Ilre receave bearer I pray you Mte yourname myne

√ 古 T l F一R 子 SS

图1-2

虽然基弗是一个小伙子，比贝平顿还年轻，这件传递信息的差事却做得从容自在、游刃有 余。可是，每趟来回查特里宅邸的途中，他都会多拐一个弯。表面上他是玛丽的特务，事实上 他是双面间谍。且回到1585年，基弗在回到英格兰之前，写了一封信给沃尔辛厄姆爵士，向 他毛遂自荐。基弗意识到，他的天主教背景是打进反伊丽莎白女王密谋核心的最佳面具。

得到贝平顿与玛丽的往来密函后，沃尔辛厄姆的第一目标就是解译它们的内容。全然了解 密码学价值的他，曾在伦敦设立了一所密码学校，并聘任托马斯 ·菲利普当他的密码秘书。菲 利普是语言学家，通晓法语、意大利语、西班牙语、德语，更重要的是，他是欧洲优秀的密码 分析家之一。寄给或出自玛丽之手的信函，都被菲利普一一掌握。他是频率分析法大师，破解 密码是迟早的事。

所谓频率分析法，以英文为例，首先，我们必须分析一长篇甚至数篇普通的英文文章，以 确立每个英文字母的出现频率。据统计，英文字母中出现频率最高的是e, 接下来是t, 然后是 a… 然后，检视我们要处理的密码文，并把每个字母的出现频率整理出来。假设密码文内出现 频率最高的字母是O, 那么它很可能就是e 的替身；如果密码文内出现频率次高的字母是X,

那它可能就是t 的替身；如果密码文内出现频率第三高的字母是P, 那它可能就是a 的替身。

在此情况下，我们需要一种更精细的频率分析法，才能有把握地继续下去，判别出这3 个常用的字母0、X、P的真实身份。我们可以把观察焦点转向它们跟其他字母相邻的频率上。 例如，字母O 是否出现在许多字母之前或之后?还是它只出现在某些特定的字母旁边?这些 问题的答案可以进一步告诉我们O 所替代的字母是元音还是辅音。如果O 所替代的字母是元 音，跟它相邻的字母应该很多；如果它所替代的字母是辅音，有很多字母可能没有机会跟它相 邻。例如，字母e 几乎可以出现在任何字母的前面或后面，但字母t 就不太可能与b 、d、g、j k 、m 、q 、v相邻。

一旦破译出几个字母后，密码分析的工作就可以快速地开展下去了。

玛丽在7月17日函复贝平顿时，实质上等于签下了自己的死刑判决书。她于信中提到这 个“计划”,尤其希望他们在刺杀伊丽莎白时，能先派人救出她。这封信在交给贝平顿之前， 依例拐了个弯，来到菲利普手上。他很快就破译出这封信的内容。读毕，菲利普在信上标了一 个绞刑架的符号。

8月11日，玛丽女王和她的侍从被特许在查特里宅邸的属地骑马。当玛丽来到 一 片荒野

**第1章密码学概述**

上时，瞥见一群人骑着马过来。她的第一个念头是，必定是贝平顿的人来救她了。但她很快就 明白，这些人是来带她去接受审判的。玛丽女王如图1-3所示。



图1-3

审判在10月15日开庭，现场有两位首席法官、4位陪审人员、法务大臣、财政大臣以及 沃尔辛厄姆爵士等。菲利普坐在旁听席，静静看着他们呈示他从加密信函中掘出的证据。

审判延续到次日。审判结束时，玛丽把命运交给法官。10天后，星室法庭在威斯敏斯特 聚会，认定玛丽“图谋、设想各种能致使英格兰女王死亡、毁灭的事件”,因此判决有罪。他 们建议处以死刑。伊丽莎白一世签署了死刑判决书。

从这个历史真事可以看出，密码学非常重要，关乎生死。至于在战争年代，密码学就更加 重要了。情报永远要在密码的保护下发送出去。

**.** **密码学简史**

密码学是研究通信安全保密的科学，其目的是保护信息在信道上传输的过程中不被他人窃 取、解读和利用，它主要包括密码编码学和密码分析学两个相互独立又相互促进的分支。前者 研究将发送的信息(明文)变换成没有密钥不能解或很难解的密文的方法，而后者则研究分析 破译密码的方法。其发展经历了相当长的时期。第一次世界大战之前，密码学的重要进展根本 是不为人知的，很少有文献披露这方面的信息。直到1918年，由W.F.Friendnmn论述了重合 指数及其在密码学中的应用以及转轮机专利的发表才引起了人们的重视，但仅仅由军事和秘密 部门所控制。

第一次世界大战之后到20世纪40年代末期，密码学家们将信息论、密码学和数学结合起



**Windows CIC++加密解密实战**

来研究，使信息论成为研究密码编码学和密码分析学的重要理论基础。完全处于秘密工作状态 的研究机构，开始在密码学方面取得根本性的进展，具有代表性的有 Shannon(香农)的论文 ——《保密系统的通信理论》和《通信的数学理论》,他将安全保密的研究引入了科学的轨道， 从而创立了信息论的一个新学科。

从20世纪50年代初期到60年代末期的20年中，在密码学的研究方面公开发表的论文极 少，但 David Kahn于1967年出版的著作——《破译者》使密码学的研究涉及了相当广泛的 领域，使不知道密码学的人了解了密码学，因此密码学的研究有了新的进展。

自20世纪70年代初期到现在，随着计算机科学与技术的发展，促进了密码学研究的兴起 和发展，人们使用密码学技术来保护计算机系统中信息的安全。因此，在密码学的研究和应用 等方面取得了许多惊人的成果和理论。具有代表性的有：

(1)Diffie和Hellman于1976年发表的“密码学的新方向”一文提出了公开密钥密码学 (公开密钥或双密钥体制),打破了长期沿用单密钥体制的束缚，提出了一种新的密码体制。 公开密钥体制可使收、发信息的双方无须事先交换密钥就可以秘密通信。

(2)Horst Festal 研究小组于20世纪70年代初着手研究美国数据加密标准 (Data Encryption Standard,DES),并于1973年发表了“密码学与计算机保密”等有价值的论文， 该文论述了他们的研究成果并被美国标准局(National Bureau of Standards,NBS)采纳，于 1977年正式公布实施为美国数据加密标准，并被简称为DES 标准。

上述密码学的发展可粗略地划分为三个阶段：第一阶段(1949年之前)的密码学可以说 不是什么学科，仅为一门艺术；第二阶段(1949年到1975年)可以说是密码学研究的“冬天”, 成果和论文少且为单密钥体制，但在这一阶段有如 Shaman 的理论和 David Kahn的著作并为 密码学奠定了坚实的理论基础；第三阶段(1976年到现在)可以说是密码学研究的“春天”, 密码学的各种理论和观点百花齐放，应用硕果累累。

我们也可以看一下密码历史长河中的年份流水表： 公元前400年，希腊人发明了置换密码。

1881年，世界上第一个电话保密专利出现。

二战期间，德国军方启用“恩尼格玛”密码机。

1976年，由于对称加密算法已经不能满足需要，Diffie 和 Hellman发表了一篇叫《密码 学新动向》的文章，介绍了公钥加密的概念，由Rivet、Shamir、Adelman提出了RSA 算法。

1985年，N.Koblitz和Miller 提出将椭圆曲线用于密码算法，根据是有限域上的点群中的 离散对数问题ECDLP, 它比因子分解更难(指数级)。

ECC 产生背景：随着分解大整数方法的进步和完善、计算机速度的提高以及计算机网络 的发展，RSA 的密钥需要不断增加长度才能保证数据安全。但是，这导致了RSA 加密速度大 为降低，对使用RSA 的应用带来了很大的负担，需要一种新的算法来替代RSA。

1993年，美国国家标准和技术协会(National Instituteof Standards and Technology,NIST) 提出安全散列算法 (SHA)。

1995年，又发布了修订版FIPS PUB 180-1,通常称之为SHA-1。

**第1章密码学概述**

1997年，美国国家标准局公布实施了美国数据加密标准 (DES)。

1997年，利用各国7万台计算机历时96天破解了 DES 的密钥。

1998年，电子边境基金会(Electronic Frontier Foundation,EFF)用25万美元制造的专用 计算机花费56小时破解了DES 的密钥。

1999年，EFF 用22小时15分完成了DES 的破解工作。

1999年年底，有人把512位的整数分解因子，512位的RSA 密钥被破解。

2000年10月，美国国家标准和技术协会(NIST) 宣布选择Rijndael作为将来的AES。注： Rijndael是在1999年由研究员Joan Daemen和 Vincent Rijmen 创建的。

2004年，在国际密码学会议(Crypto'2004) 上，来自山东大学的王小云教授的报告介绍 了破译MD5 、HAVAL-128 、MD4和 RIPEMD算法。随后SHA-1也被宣告破解。

2009年年底，768位的整数也被成功分解，威胁到了现在流行的1024位密钥的安全性。





 **密码学的基本概念**

**1.3.1** **基本概念**

密码学作为数学的一个分支，是研究信息系统安全保密的科学，是密码编码学和密码分析 学的统称。

密码编码学是关于消息保密的技术和科学。密码编码学是密码体制的设计学，即怎样编码， 采用什么样的密码体制保证信息被安全地加密。从事此行业的人员被称为密码编码者 (Cryptographer)。

密码分析学是与密码编码学相对应的技术和科学，即研究如何破译密文的科学和技术。密 码分析学是在未知密钥的情况下从密文推演出明文或密钥的技术。密码分析者(Cryptanalyst) 是从事密码分析的专业人员。

**1.3.2** **密码学要解决的5大问题**

密码学主要是为了解决信息安全的5大问题，即机密性、可用性、完整性、认证性、不可 否认性。

机密性指保密信息不会透露给非授权用户或实体，确保存储的信息或传输的信息仅能被授 权用户获取到，而非授权用户获取到也无法知晓信息内容。解决方案是使用密码算法对需要保 密的信息进行加密。

可用性指保障信息资源随时可提供服务的能力特性。

完整性指信息在生成、传输、存储和使用过程中发生的人为或非人为的非授权篡改均可以 被检测到。解决方案是利用密码函数生成信息“指纹”,实现完整性检验。

认证性指一个消息的来源和消息本身被正确地标识，同时确保该标识没有被伪造。解决方



**Windows C/C++加密解密实战**

案是利用密钥和认证函数相结合来确定信息的来源。

不可否认性是指用户无法在事后否认曾经进行信息的生成、签发、接收行为。解决方案是 对信息进行数字签名。

**1.3.3** **密码学中的五元组**

在密码学中，有一个五元组：明文(Plaintext) 、 密文 (Ciphertext) 、 密钥(Key) 、 加 密算法(Encryption Algorithm)、解密算法(Decryption Algorithm)。对应的加密方案称为密 码体制。

明文是作为加密输入的原始信息，即消息的原始形式，通常用m 或 p 表示。所有可能的 明文构成的有限集称为明文空间，通常用 M 或 P 来表示。

密文是明文经加密变换后的结果，即消息被加密处理后的形式，通常用c 表示。所有可 能的密文构成的有限集称为密文空间，通常用C 来表示。

密钥是参与密码变换的参数，通常用k 表示。一切可能的密钥构成的有限集称为密钥空问， 通常用 K 表示。

加密算法是将明文变换为密文的变换函数，相应的变换过程称为加密，即编码的过程，通 常用E 表示，即c=Ek(p)。

解密算法是将密文恢复为明文的变换函数，相应的变换过程称为解密，即解码的过程，通 常用 D 表示，即p=Dk(c)

对于有实用意义的密码体制而言，总是要求它满足：p=Dk(Ek (p)),即用加密算法得到的 密文。同样，总是能用一定的解密算法恢复出原始的明文来。

**1.3.4** **加解密算法的分类**

通常可以将加解密算法分为对称算法和非对称算法。

对称算法使用的密钥必须完全保密，且加密密钥和解密密钥相同。对称算法的优点：(1)运 算速度快，具有较高的吞吐率；(2)对称密码体制中的密钥相对较短；(3)对称保密体制的 密文长度往往和明文长度相同，或扩张较小。对称算法的缺点：(1)密钥分发需要安全通道；

(2)密钥量大，难以管理；(3)难以解决不可否认问题。

非对称算法又称为公钥算法，它有两个密钥，一个是对外公开的公钥，可以像电话号码一 样注册；另一个是必须保密的私钥，只有拥有者才知道。非对称加密是为了解决对称加密体制 的缺陷而提出的，一个是密钥的分发和管理问题；另一个是不可否认问题。非对称算法的优点 是：(1)密钥分发相对容易；(2)密钥管理简单；(3)可以有效地实现数字签名。非对称 算法的缺点是：(1)运算速度较慢；(2)同等安全强度下，非对称密码体制要求的密钥位数 要多些；(3)非对称保密体制中，密文的长度往往大于明文的长度。

**第** **2** **章**

**搭建C和C**++**密码开发环境**

**2.1 密码编程的两个重要的国际库**

密码编程如果所有事情都要从头开始写，那结果将是灾难性的。幸亏国际开源界已经为我 们提供了两个密码学相关的函数库：OpenSSL 和 Crypto++。从功能上来讲，OpenSSL 更为强 大，不但提供了编程用的 API 函数，还提供了强大的命令行工具，可以通过命令来进行常用 的加解密、签名验签、证书操作等功能。Crypto++纯粹是用C++写的，适合 C++洁癖患者， OpenSSL 是用C 语言写的，也可以在C++ 程序中调用。

友情提醒，一线密码应用开发中，OpenSSL 用得多些，建议掌握。

**2.2 C/C++密码库OpenSSL**

Crypto++虽好，但功能不如 OpenSSL。一线开发中，用得更多的是OpenSSL。虽然OpenSSL 是用C 语言写的，但在C++ 程序中使用完全没有问题。何况，OpenSSL 很多地方利用了面向 对象的设计方法与多态来支持多种加密算法。所以，学好 OpenSSL, 甚至分析其源码，对我 们提高面向对象的设计能力大有帮助。很多著名的开源软件，比如内核XFRM框架、VPN 软 件 StrongSwan等都是用C 语言来实现面向对象设计的。因此，我们会对OpenSSL 叙述的更为 详细些，因为一线实践开发中，经常会碰到这个库的使用(很多C# 开发的软件，底层的安全 连接也会用VC 封装OpenSSL为控件后供C# 界面使用，更不要说Linux的一线开发了),希 望大家能预先掌握好。

随着 Internet 的迅速发展和广泛应用，网络与信息安全的重要性和紧迫性日益突出。 Netscape公司提出了安全套接层协议(Secure Socket Layer,SSL),该协议基于公开密钥技术， 可保证两个实体间通信的保密性和可靠性，是目前Internet上保密通信的工业标准。

Eric A.Young和 Tim J.Hudson自1995年开始编写后来具有巨大影响力的OpenSSL 软件 包，这是一个没有太多限制的开放源代码的软件包，可以利用这个软件包做很多事情。1998 年 ，OpenSSL 项目组接管了OpenSSL的开发工作，并推出了OpenSSL 的0.9.1版，到目前为



**Windows C/C++加密解密实战**

止 ，OpenSSL 的算法已经非常完善，对SSL 2.0、SSL 3.0以 及TLS 1.0都支持。OpenSSL 目 前新的版本是1.1.1版。

OpenSSL 采用C 语言作为开发语言，使得OpenSSL 具有优秀的跨平台性能，可以在不同 的平台使用。OpenSSL支持Linux 、Windows 、BSD 、Mac等平台，OpenSSL 具有广泛的适用 性。OpenSSL 实现了8种对称加密算法，如AES、DES、Blowfish、CAST、IDEA、RC2、RC4、 RC5, 实现了4种非对称加密算法，如DH 、RSA 、DSA和 ECC, 实现了5种信息摘要算法， 如 MD2 、MD5 、MDC2 、SHA1和RIPEMD。此 外 ，OpenSSL还实现了密钥和证书的管理。

OpenSSL的 License (许可证)是SSLeay License和 OpenSSL License的结合，这两种许 可证实际上都是BSD类型的许可证，依照许可证里面的说明，OpenSSL可以被用作各种商业、 非商业的用途，但是需要相应地遵守一些协定，其实这都是为了保护自由软件作者及其作品的 权利。

**2.2.1** **OpenSSL源代码模块结构**

OpenSSL 整个软件包大概可以分成三个主要的功能部分：密码算法库、SSL 协议库以及 应用程序。OpenSSL 的目录结构也是围绕这三个功能部分进行规划的，具体可见表2-1。

**表2-** **1** **OpenSSL的目录结构及功能**

|  |  |
| --- | --- |
| **目** **录** **名** | **功能描述** |
| Crypto | 所有加密算法源码文件和相关标准(如X.509源码文件)是OpenSSL中重要的目录，包含 OpenSSL密码算法库的所有内容 |
| SSL | SSL存放OpenSSL中SSL协议各个版本和TLS 1.0协议的源码文件，包含OpenSSL协议库的 所有内容 |
| Apps | 存放OpenSSL中所有应用程序的源码文件，如CA、X509等应用程序的源码文件就存放在这里 |
| Docs | 存放OpenSSL中所有的使用说明文档，包含三个部分：应用程序说明文档、加密算法库API 说明文档以及SSL协议API说明文档 |
| Demos | 存放一些基于OpenSSL的应用程序例子，这些例子一般都很简单，演示怎么使用OpenSSL中 的一个功能 |
| Include | 存放使用OpenSSL的库时需要的头文件 |
| Test | 存放OpenSSL自身功能测试程序的源码文件 |

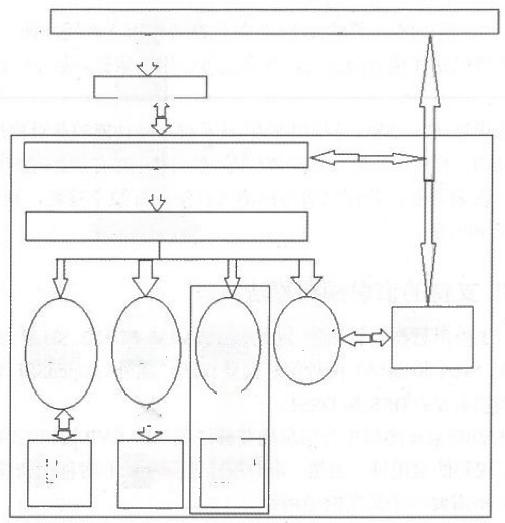
OpenSSL 的算法目录 Crypto 目录包含 OpenSSL 密码算法库的所有源代码文件，是 OpenSSL中重要的目录之一。OpenSSL的密码算法库包含OpenSSL中所有密码算法、密钥管 理和证书管理相关标准的实现。

**2.2.2** **OpenSSL** **加密库调用方式**

OpenSSL是全开放的和开放源代码的工具包，实现安全套接层协议(SSL v2/v3) 和传输 层安全协议(TLS v1) 以及形成一个功能完整的、通用目的的加密库SSLeay。应用程序可以 通过三种方式调用SSLeay, 如图2-1所示。



第 2 章 搭 建C 和 C++ 密码开发环境



应用程序

食

其他层次

openSSL 加密库接口(如：RSA EVP)



Engine 安全平台

OpenSSL SSLeay

engine

engine



Ubsec CSP

Vestone

engine

父 公

Ubsec engine

Testor CSP

Aep

Ae Cs



图2-1

一是直接调用，二是通过OpenSSL 加密库接口调用，三是通过Engine 平台和OpenSSL 对象调用。除了SSLeay 外，用户还可以通过 Engine 安全平台访问CSP。

使用Engine技术的OpenSSL已经不仅仅是一个密码算法库，而是一个提供通用加解密接 口的安全框架，在使用时只要加载了用户的Engine模块，应用程序中所调用的OpenSSL 加解 密函数就会自动调用用户自己开发的加解密函数来完成实际的加解密工作。这种方法将底层硬 件的复杂多样性与上层应用分隔开，大大降低了应用开发的难度。

**2.2.3** **OpenSSL支持的对称加密算法**

OpenSSL一共提供了8种对称加密算法，其中7种是分组加密算法，仅有一种流加密算 法是RC4。这7种分组加密算法分别是AES 、DES 、Blowfish 、CAST 、IDEA 、RC2 、RC5, 都支持电子密码本模式(ECB) 、 加密分组链接模式(CBC) 、 加密反馈模式 (CFB) 和输出 反馈模式(OFB)4 种常用的分组密码加密模式。其中，AES 使用的加密反馈模式(CFB) 和 输出反馈模式(OFB) 分组长度是128位，其他算法使用的则是64位。事实上，DES 算法里 面不仅仅是常用的DES 算法，还支持三个密钥和两个密钥3DES算法。OpenSSL还使用EVP 封装了所有的对称加密算法，使得各种对称加密算法能够使用统一的API 接口EVP\_Encrypt 和 EVP\_Decrypt进行数据的加密和解密，大大提高了代码的可重用性能。

**2.2.4** **OpenSSL支持的非对称加密算法**

OpenSSL实现了4种非对称加密算法，包括DH算法、RSA 算法、DSA算法和ECC 算法。



**Windows C/C++加密解密实战**

DH 算法一般用于密钥交换。RSA 算法和 ECC 算法既可以用于密钥交换，又可以用于数字签 名，当然，如果你能够忍受其缓慢的速度，那么也可以用于数据加解密。DSA 算法则一般只 用于数字签名。

跟对称加密算法相似，OpenSSL也使用EVP技术对不同功能的非对称加密算法进行封装， 提供了统一的API接口。若使用非对称加密算法进行密钥交换或者密钥加密，则使用EVPSeal 和 EVPOpen 进行加密和解密；若使用非对称加密算法进行数字签名，则使用 EVP\_Sign 和 EVP\_Verify进行签名和验证。

**2.2.5 OpenSSL支持的信息摘要算法**

OpenSSL实现了5种信息摘要算法，分别是MD2、MD5、MDC2、SHA(SHA1)和RIPEMD。 SHA算法事实上包括SHA 和 SHA1 两种信息摘要算法，此外，OpenSSL 还实现了DSS 标准 中规定的两种信息摘要算法：DSS 和 DSS1。

OpenSSL采用EVPDigest接口作为信息摘要算法统一的EVP 接口，对所有信息摘要算法 进行了封装，提供了代码的重用性。当然，跟对称加密算法和非对称加密算法不一样，信息摘 要算法是不可逆的，不需要一个解密的逆函数。

**2.2.6 OpenSSL密钥和证书管理**

OpenSSL实现了ASN.1的证书和密钥相关标准，提供了对证书、公钥、私钥、证书请求 以及CRL 等数据对象的DER、PEM和 BASE64的编解码功能。OpenSSL提供了产生各种公 开密钥对和对称密钥的方法、函数和应用程序，同时提供了对公钥和私钥的DER 编解码功能， 并实现了私钥的PKCS#12和 PKCS#8的编解码功能。OpenSSL 在标准中提供了对私钥的加密 保护功能，使得密钥可以安全地进行存储和分发。

在此基础上，OpenSSL 实现了对证书的X.509 标准编解码、PKCS#12 格式的编解码以及 PKCS#7格式的编解码功能，并提供了一种文本数据库，支持证书的管理功能，包括证书密钥 产生、请求产生、证书签发、吊销和验证等功能。

事实上，OpenSSL 提供的CA 应用程序就是一个小型的证书管理中心 (CA), 实现了证 书签发的整个流程和证书管理的大部分机制。

**2.2.7 面向对象与OpenSSL**

OpenSSL支持常见的密码算法。OpenSSL 成功地运用了面向对象的方法与技术，才使得 它能支持众多算法并能实现SSL 协议。OpenSSL的可贵之处在于它利用面向过程的C 语言去 实现面向对象的思想。

面向对象方法是一种运用对象、类、继承、封装、聚合、消息传递、多态性等概念来构造 系统的软件开发方法。

面向对象方法与技术起源于面向对象的编程语言(Object-Oriented Programming Language, 0OPL) 。 但是，面向对象不仅是一些具体的软件开发技术与策略，而且是一整套关于如何看



**第** **2** **章** **搭** **建C** **和** **C++** **密码开发环境**

待软件系统与现实世界的关系、以什么观点来研究问题并进行求解，以及如何进行系统构造的 软件方法学。概括地说，面向对象方法的基本思想是，从现实世界中客观存在的事物(对象) 出发来构造软件系统，并在系统构造中尽可能运用人类的自然思维方式。面向对象方法强调直 接以问题域(现实世界)中的事物为中心来思考问题、认识问题，并根据这些事物的本质特征， 把它们抽象地表示为系统中的对象，作为系统的基本构成单位。这可以使系统直接地映像问题 域，保持问题域中的事物及其相互关系的本来面貌。

结构化方法采用了许多符合人类思维习惯的原则与策略(如自顶向下、逐步求精)。面向 对象方法则更加强调运用人类在日常的逻辑思维中经常采用的思想方法与原则，例如抽象、分 类、继承、聚合、封装等。这使得软件开发者能够更有效地思考问题，并以其他人也能看得懂 的方式把自己的认识表达出来。具体地讲，面向对象方法有如下一些主要特点：

(1)从问题域中客观存在的事物出发来构造软件系统，用对象作为这些事物的抽象表示， 并以此作为系统的基本构成单位。

(2)事物的静态特征(可以用一些数据来表达的特征)用对象的属性表示，事物的动态 特征(事物的行为)用对象的服务表示。

(3)对象的属性与服务结合成一体，成为一个独立的实体，对外屏蔽其内部细节(称作 封装)。

(4)对事物进行分类。把具有相同属性和相同服务的对象归为一类，类是这些对象的抽 象描述，每个对象是它的类的一个实例。

(5)通过在不同程度上运用抽象的原则(较多或较少地忽略事物之间的差异),可以得 到较一般的类和较特殊的类。子类继承超类的属性与服务，面向对象方法支持对这种继承关系 的描述与实现，从而简化系统的构造过程及其文档。

(6)复杂的对象可以用简单的对象作为其构成部分(称作聚合)。

(7)对象之间通过消息进行通信，以实现对象之间的动态联系。

(8)通过关联表达对象之间的静态关系。

概括以上几点可以看到，在使用面向对象方法开发的系统中，以类的形式进行描述并通过 对类的引用而创建的对象是系统的基本构成单位。这些对象对应着问题域中的各个事物，它们 内部的属性与服务刻画了事物的静态特征和动态特征。对象类之间的继承关系、聚合关系、消 息和关联，如实地表达了问题域中事物之间实际存在的各种关系。因此，无论是系统的构成成 分，还是通过这些成分之间的关系而体现的系统结构，都可以直接地映像问题域。

面向对象方法代表了一种贴近自然的思维方式，它强调运用人类在日常的逻辑思维中经常 采用的思想方法与原则。面向对象方法中的抽象、分类、继承、聚合、封装等思维方法和分析 手段，能有效地反映客观世界中事物的特点和相互的关系。而面向对象方法中的继承、多态等 特点可以提高过程模型的灵活性、可重用性。因此，应用面向对象的方法将降低工作流分析和 建模的复杂性，并使工作流模型具有较好的灵活性，可以较好地反映客观事物。

在OpenSSL 源代码中，将文件及网络操作封装成 BIO。BIO 几乎封装了除了证书处理外 的 OpenSSL 所有的功能，包括加密库以及SSL/TLS 协议。当然，它们都只是在OpenSSL 其



**Windows C/C++加密解密实战**

他功能之上封装搭建起来的，但却方便了不少。 OpenSSL 对各种加密算法封装，就可以使用 相同的代码但采用不同的加密算法进行数据的加密和解密。

**2.2.8** **BIO接口**

在 OpenSSL 源代码中，I/O 操作主要有网络操作和磁盘操作。为了方便调用者实现其I/O 操作，OpenSSL源代码中将所有的与I/O操作有关的函数进行统一封装，即无论是网络还是磁 盘操作，其接口是一样的。对于函数调用者来说，以统一的接口函数去实现其真正的I/O 操作。

为了达到此目的，OpenSSL 采用BIO 抽象接口。BIO 是在底层覆盖了许多类型 I/O接口 细节的一种应用接口，如果在程序中使用BIO, 就 可 以 和 SSL 连接、非加密的网络连接以及 文 件IO 进行透明的连接。BIO 接口的定义如下：

};

structbio st {

BIO\_METHOD \*method;

其 中 ，BIO\_METHOD 结构体是各种函数的接口定义。如果是文件操作，此结构体如下：

static BIO\_METHOD methods\_filep=

{

BIO\_TYPE\_FILE

"FILE pointer",

file\_write file\_read, file\_puts, file gets file\_ctrl, file new file\_free,

NULL, };

以上定义了7个文件操作的接口函数的入口。这7个文件操作函数的具体实体与操作系统 提 供 的API 有 关 。BIO\_METHOD 结构体如果用于网络操作，其结构体如下：

staitc BIO\_METHOD methods\_sockp= {

BIO\_TYPE\_SOCKET, "socket"

sock\_write, sock\_read, sock\_puts, sock\_ctrl, sock\_new sock\_free, NULL,



**第** **2** **章** **搭** **建C** **和C++** **密码开发环境**

};

它跟文件类型BIO 在实现的动作上基本上是一样的。只不过是前缀名和类型字段的名称 不一样。其实在像Linux这样的系统中，Socket 类型跟fd 类型是一样的，它们是可以通用的， 但是，为什么要分开来实现呢?那是因为有些系统(如Windows 系统)的Socket跟文件描述 符是不一样的，所以，为了平台的兼容性，OpenSSL 就将这两类分开来了。

**2.2.9 EVP 接口**

EVP 系列的函数定义包含在 evp.h里面，这是一系列封装了 OpenSSL 加密库里面所有算 法的函数。通过这样统一的封装，使得只需要在初始化参数的时候做很少的改变，就可以使用 相同的代码但采用不同的加解密算法进行数据的加密和解密。

EVP 系列函数主要封装了三大类型的算法，即公开密钥算法(也称非对称加密算法)、 数字签名算法和对称加密算法(业内一般讲加密算法就是指加解密算法),要支持这些算法， 需要调用OpenSSL\_addall\_algorithms函数。

**1.公开密钥算法**

函数名称：EVPSeal\*..\* 、EVPOpen\*.\*。

功能描述：该系列函数封装提供了公开密钥算法的加密和解密功能，实现了电子信封的功 能。

相关文件：p\_seal 、p\_open.c

**2.数字签名算法**

函数名称：EVP\_Sign\*..\* 、EVP\_Verify\*...\*。

功能描述：该系列函数封装提供了数字签名算法的功能。

相关文件：p\_sign.c 、p\_verify.c。

**3.对称加密算法**

函数名称：EVP\_Encrypt\*..\*。

功能描述：该系列函数封装提供了对称加密算法的功能。

相关文件：evp\_enc.c 、p\_enc.c 、p\_dec.c 、e\_\*.c。

**4.信息摘要算法**

函数名称：EVPDigest\*..\*。

功能描述：该系列函数封装实现了多种信息摘要算法。

相关文件：digest.c 、m\_\*.c。

**5.信息编码算法**

函数名称：EVPEncode\*...\*。

功能描述：该系列函数封装实现了ASCIⅡ码与二进制码之间的转换函数的功能。



Windows C/C++加密解密实战

**2.2.10** **关于版本和操作系统**

本书既要照顾老项目维护者，又要照顾新项目开发者。因此，笔者会选择目前新的版本和 较老但使用较多且稳定的版本同时介绍，两个版本各有千秋，新版本会引进不少新技术和新算 法，比如在新版本中加入了国密算法SM2/3/4,老版本主要用来兼容老项目，建议开发新项目 还是用新版的OpenSSL。目前新的版本是OpenSSL 1.1.1b,它是在2019年2月26日发布的。 我们选择的老版本是 OpenSSL-1.0.2m。另外要注意的是，OpenSSL 官方现在已停止对0.9.8 和1.0.0两个版本的升级维护，所以大家选择的老版本也别太老了。

至于操作系统的选择，当前密码应用开发在 Linux和Windows 下都开展得如火如荼，因 此也会介绍在这两个系统下的安装和使用。笔者选择的操作系统是Windows 7和 CentOS 7。

**2.2.11 在** **Windows下编译** **OpenSSL 1.1.1**

OpenSSL 是一个开源的第三方库，它实现了SSL(Secure Socket Layer,安全套接层)和 TLS(Transport Layer Security,安全传输层)协议，被企业应用广泛采用。对于一般的开发人 员而言，在Win32 OpenSSL上下载已经编译好的OpenSSL库是省力省事的好办法。对于高级 的开发用户，可能需要适当地修改或者裁剪 OpenSSL, 那么编译它就成为一个关键问题。考 虑到我们早晚要成为高级开发用户，所以掌握OpenSSL 的编译是早晚的事。下面主要讲述如 何在Windows上编译OpenSSL库。

前面讲了不少理论知识，虽然枯燥，但可以从宏观层面上对OpenSSL进行高屋建瓴地了 解，这样以后走迷宫时不至于迷路。下面即将进入实战环节。废话不多说，打开官网下载源码。 OpenSSL的官网地址是https:/www.openssl.org。这里使用的是新版本OpenSSL 1.1.1,在学习 的时候我们要勇于尝试新版本。另外要注意的是，OpenSSL 官方现在已停止对0.9.8和1.0.0 两个版本的升级维护，还在维护老代码的同志要注意了，升级是早晚的事情。这里会下载两个 版本进行演示，为了照顾喜欢尝鲜的读者，先下载目前新的版本1.1.1,下载下来的压缩文件 是 openssl-1.1.1.tar.gz。后面会下载一个推荐用于实际开发的版本，即1.0.2m, 下载下来的压 缩文件是 openssl-1.0.2m.tar.gz,不求最新，但求稳定，这是一线开发的原则。另外，本书也涉 及一些CentOS7 下 OpenSSL的使用，使用的版本是CentOS7 自带的，也是为了稳定。

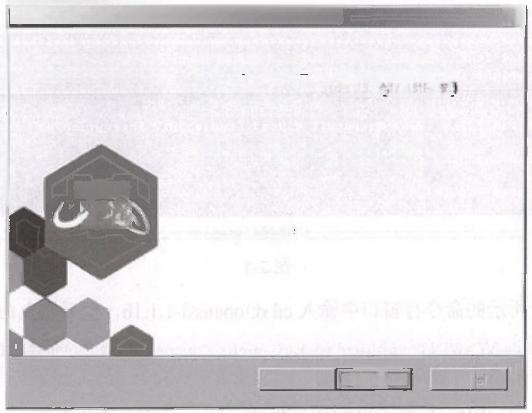
**1.安装ActivePerl解释器**

因为编译OpenSSL源码的过程中会用到Perl 解释器，所以在编译OpenSSL库之前，还需 要下载一个 Perl 脚本解释器，这里选用大名鼎鼎的 ActivePerl,我们可以从其官方网站 (<https://www.activestate.com/>)下载。这里下载后的文件为ActivePerl-5.26.1.2601-MSWin32- x64-404865.exe。

下载完毕后，就可以开始安装了。直接双击即可开始安装，安装时间有点长，要有点耐心， 最终会提示安装成功。安装完成的界面如图2-2所示。



**第** **2** **章** **搭** **建C** **和C++** **密码开发环境**



Activerer15.26.1 Bail42601 (64-bit)Set ×

**Completingthe ActivePerl**

**5.26.1 Build 2601(64-bit) Setup Wizard**

Ock the Tnish"button to estt the Setop Wzard.

\* 大

风 Vew readme fle

n

**ActiveState**



<Bak

cang

图2-2

**2.下载OpenSSL 1.1.1**

目 前 ，OpenSSL 1.1.1b 是新的版本，我们可以去官方网站下载，下载地址为 <https://www.openssl.org/source/>。下载下来的文件名是 openssl-1.1.1b.tar.gz。

下面开始我们的编译之旅。为什么要编译?因为编译后会生成库，这个库可以放到工程环 境中使用。

**3.安装VC**

在Windows平台上，用微软Visual Studio中的C编译器生成OpenSSL的静态库或动态库。 我们要在VC 命令行下编译，所以需要先安装VC, 至少要VC 2008,本书采用的是VC 2017, 建议大家使用这个版本，以后看书的过程中有问题方便联合作者一起排查。

**4.编译出32位的Debug** **版本的静态库**

安装完 ActivePerl 后，就可以正式编译安装 OpenSSL 了。我们下载下来的 openssl-1.1.1.tar.gz 是一个源码压缩包，需要对其进行解压，然后编译出开发所需要的静态库 或动态库。

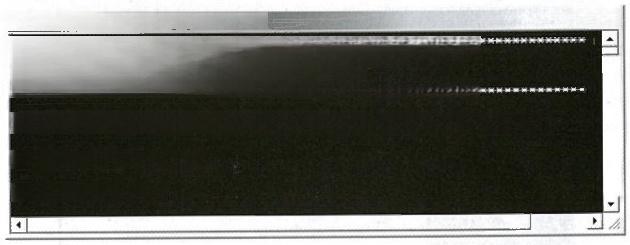
具体安装步骤如下：

(1)解压源码目录。把 openssl-1.1.1b.tar.gz复制到某个目录下，比如D:, 然后解压缩， 解压后的目录为D:\openssl-1.1.1b,进入D:\openssl-1.1.1b,就可以看到各个子文件夹了。

(2)配置 OpenSSL。打开VC 2017的开发者命令行提示窗口，单击“开始”→“Visual Studio 2017”→“Visual Studio Tools”→“VS 2017的开发人员命令提示符”,即可出现如图2-3所 示的窗口。



**Windows C/C++加密解密实战**



|  |  |  |
| --- | --- | --- |
| **四管理员：S 2017的开发人员命令提示符**  *wwWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW*  **w Uisual Studio 2017 Developer Connand Prompt v15.9.11 \*Copyright(c)2017 Microsoft Corporation**  WW**WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWHWWWWWWWWWWWWWWWWWWW** | -l | 口× |
| **c:Progran Piles (×86)Hicrosoft Uisual Studio2017 Enterpriae>d:**  **D:Sed d:openssl-1.1.1b d:\openss1-1.1.1b>** |  |  |

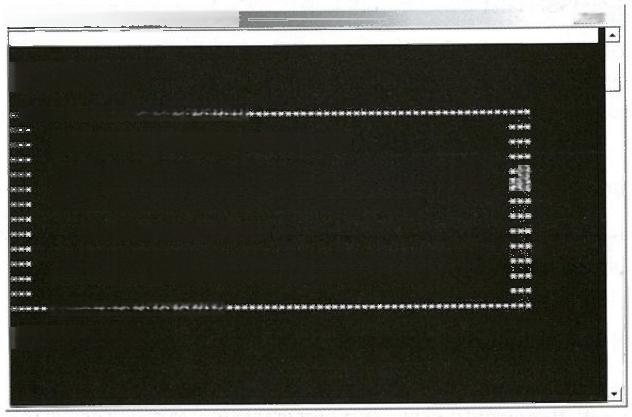
图2-3

然后在图2-3所示的命令行窗口中输入cd d:\openssl-1.1.1b,然后输入命令：



perl Configure debug-VC-WIN32 no-shared no-asm -prefix="d:/openssl-1.1.1b/win32-debug" --openssldir="d:/openssl-1.1.1b/win32-debug/ssl"

其中，debug-VC-WIN32 表示32位调试模式，no-asm 表示不用汇编。 接着按回车键，然后开始自动配置，如图2-4所示。



四管理员：vs 2017的开发入员命令提示符 ppn install dmake

Creating configdata.pn Creating nakefile

WWWHWWWWWWWW\*WWWWRWWWWWWWWWWWW

OpenSSL has heen suceasfully configured

If you encounter a problen vhile building.please open an issue on GitHub <https://github.con/openssl/openssl/issues> and include the output fron the folloving conmand:

perl configdata.pn --dunp

(If you are new toOpenSSL,you might uant to consult the 'Troublechooting'section in the IHSTALL file firat)

WWWRWWWWWWWWWWwWWWWWwWm

d:opensa1-1.1.1b>

-l回x

W

WW

图2-4

其中，VC-WIN32 表示我们要编译出32位的版本，若要编译出 Debug 版本，则使用 debug-VC-WIN32,若要使用Release 版本的OpenSSL库，则不要加debug-。no-shared表示我 们要编译出静态库，若要编译出动态库，则不要加 no, 用-shared即可，这个很好理解，静态 库是不共享的，动态库是用来共享的。参数-prefix是 OpenSSL 编译完后所生成的命令程序、 库、头文件等的存放路径。-openssldir是OpenSSL编译完后生成的配置文件的存放路径。

(3)编译OpenSSL。配置完成后，我们可以继续用nmake 命令开始编译，在命令行窗口 中 输 入nmake 后按回车键即可开始编译，nmake 程 序 是VC 自带的命令行编译工具。这 一 步时 间稍长，大家可以喝杯茶。编译成功后如图2-5所示。



**第** **2** **章** **搭** **建C** **和C++** **密码开发环境**



国管理员：S 2017的开发人员命令提示符 回 ×

"C:Per164hin\per1.exe"“-1."-Heonfigdata "uti1

o""apps\tsget.in">"apps \tsget.pl"

"C:VPer164\bin \perl.exe"“-I."-Mconfigdata "util

d:\openssl-1.1.1h>

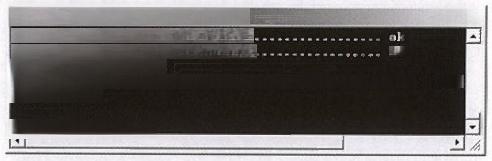
e""toolac\_rchash.in"》"toolsc rehash.pl"

图2-5

(4)测试编译。这一步不是必需的，但最好检查一下上一步的编译是否正确。在命令行 窗口中输入命令：

nmaketest

这一步时间也稍长，其实可以不用做，但笔者做了，测试全部成功，如图2-6所示。



回 管 理 员 ：Vs 2017的开发人员命令程示符

teat vrecipes\99-test\_ocstrean.t

test vrecipes\99-test\_fuzz.t .... ok

A11 tests successfu1.

Files-155,Tests-1163,495 vallclock secs(0.48 usr

Result:PASS

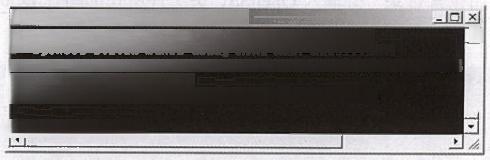
\opensrl-1.1.1h>

回 ×

0.33

图2-6

(5)安装。继续输入命令nmake install,执行成功后如图2-7所示。



园 管 理 员 ：Vs 2017的开发人员命令提示符

**D**::**p**--11..11..11hbin**32**-deia?72s.1h9tnhml 

pl1v-. .b?Xin25-e19.htnl hug htnlvnan?X448.htnl ->D:Nopen

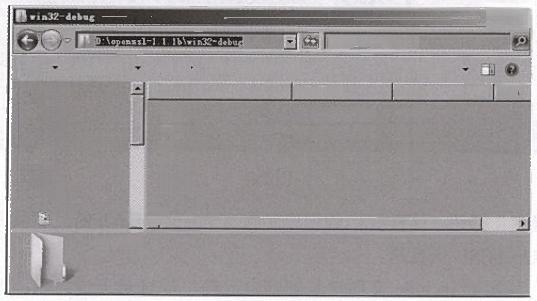
D=\openss1-1.1.1bwin32-debug\html nan?×509.htnl

Aopenss1-1.1.1h>

图2-7

(6)清理。主要是删除一些中间文件。继续输入命令nmake clean。

此时，进入 D:\openssl-1.1.1b\win32-debug,可以看到一些子文件夹，如图2-8所示。



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 回 × | | |
|  |  | 素 winQ-detu | | |
| 组织 也含到库中 共享▼ | 到录 | 新建文件夹 |  |  |
| ☆收骤夹  bn  四点面  2最近访间的位置 |  | 修改日期  2019/4/1713:10 2019/4/1713:09 2019/4/1713:07 | 类型 文件夹 文件夹 文件夹 | 大小 |
| 库 1ib |  | 2019/4/1713:10 | 文件夹 |  |
| 日视频 D wl  图片 |  | 2019/4/1713:07 | 文件夹 |  |
| 文档 |  |  |  |  |
| 5个对象 |  |  |  |  |

htal ]include

下载



名 称

图2-8



**Windows** **C/C++加密解密实战**

其 中 ，bin 目录下存放OpenSSL 的命令行程序，利用该程序我们可以在命令行下执行一些 加解密任务、证书操作任务。在 lib 目录下存放的就是我们编译出来的32位的静态库 libcrypto.lib 和 libsslL.lib, 一般的加解密程序使用libcrypto.lib 即 可 。include 目录就是我们开发 所需要的 OpenSSL 头文件。

现在，趁热打铁，马上来开发一个使用OpenSSL 静态库的程序，以此检验我们生成的静 态库是否正确。

【例2 . 1】使用32位OpenSSL 1.1.1的 Debug 静态库 ( 1 ) 打 开VC 2017,新建一个控制面板程序test。

( 2 ) 在 test.cpp 中输入代码如下：

#include "pch.h"

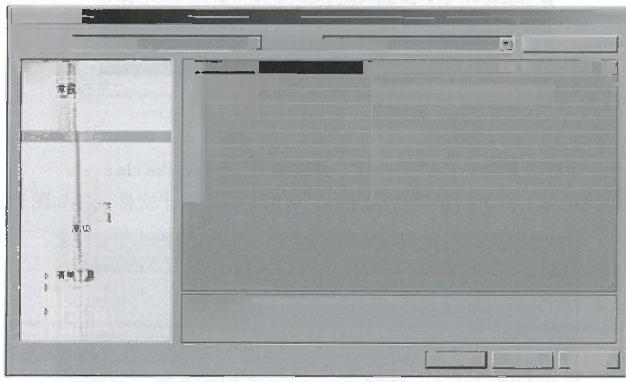
#include "openssl/evp.h"

int main() {

openssl\_add\_all\_algorithms();//使用openssl库前，必须调用该函数 printf("openssl ok\n");

return 0; }

(3)包含 include 目录。打开“test 属性页”对话框，在界面左边选择“C/C++”→ “ 常 规”,在右边的“附加包含目录”旁输入 OpenSSL 头文件所在的路径D:\openssl-1.1.1b\win32- debuglinclude, 如图2-9所示。



m 性 页

配置心):居动①la0

p 画用属性

配置属住

用于“偏描并继续”的程序数据库(zn

目 录

*4 C/C++*

常现 优 8

理

代码生成



预 译 头 出 Q

信

所有选项

合 行

链接器

档息生 成 器

生成事件

义生成步臂 p 代码分折

确 定 取 消 用

加 目 其他熟uring 指令 调试信息格式

公共语言远行时支持

使 用rindms 运行时扩展 雕消显示启动断权标志

器古等级

将器告视为描误 sDL 检壹

多处理器偏译

附加也含目录

富个或多个要添加饱含器径中的目录；当目录不止一个时，请用分号分隔。

口平台0):居动is2) 配置管理器0)

■:opasl-1.1.1\ia32-4\iaclal

是 (aoloco) 等 级 3 U3)

(-)a)

I 区



UI[

图2-9

然后单击“确定”按钮。这样我们的程序中包含openssl/evp.h 的时候就不会出错了，因为 include 目录下有OpenSSL 子目录。

顺便说一句，其实把include 文件夹复制到自己的工程目录下也可以，但考虑到很多程序 都要用到头文件，所以没必要每个工程都去复制一份 include 文件夹，建议放在一个公共路径

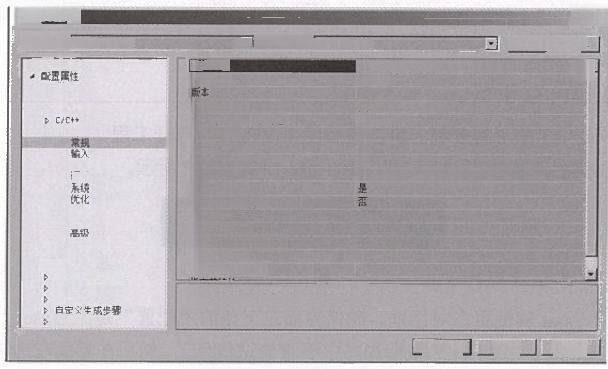


**第** **2** **章** **搭** **建C** **和C++** **密码开发环境**

下(比如D:\openssl-1.1.1b\win64-debuglinclude), 各个开发者自己包含这个公共路径即可。

(4)添加静态库。在“test属性页”对话框中，在界面左边选择“链接器”→“常规”, 在右边的“附加库目录”旁输入OpenSSL 静态库所在的路径D:openss-1.1.1b\win32-debuglib,

如图2-10所示。



oat页

配置C):居动Os)

国性

VC++目 录

4 键 接 器

青单文件 调试

副接库依赖项

使用库依精项输入

链接忧态

阻止m1 定

将醒接器器告视为误 强制文件输出

创健司热修补碘像

W信L文生成器

生成事件

代鸿分析

确定 取消 应用6)

出农件 显示进度

启用竞重链接

取消显示启动版权标志 忽略导入库

注册输出

避用户重定向 附加库目录

3.0utir)sCugaUua)5Cugtnt) 未设置

是 (THCRTAL) 是 OLD90)

☑平白更):居动0is2) 配透管理吾⑩

输出文件

/ou7 选项重写创接器创健的程序的认名称和位置。

嵌入的 mL

inds元数据

所有选项

D:\pentsl-1.1.1n\vin32-4eba\lib

令行 ▶清单工

2冈



中

否 否 否

图2-10

然后单击“应用”按钮。接着展开界面左边的“链接器”→“输入”,在右边第一行“附 加依赖项”右边的开头输入 ws2\_32.lib;Crypt32.lib;libcrypto.lib;,其中 ws2\_32.lib和Crypt32.lib 是 VC自带的库，分别实现网络功能和微软提供的加解密功能，加入这两个库的原因是 libcrypto.lib依赖于它们。最后单击“确定”按钮。

(5)保存工程并运行，运行结果如图2-11所示。

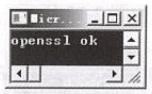


图2-11

至此，说明32位的Debug 版本的OpenSSL1.1.1b 静态库使用起来了。

**5** **.编译出32位的Release** **版本的静态库**

(1)解压源码目录(若已经存在源码目录 openssl-1.1.1b, 则可以不必再解压)。把 openssl-1.1.1b.tar.gz复制到某个目录下，比如D:,然后解压缩，解压后的目录为D:\openssl-1.1.1b, 进入D:\openssl-1.1.1b, 就可以看到各个子文件夹了。

(2)配置OpenSSL。打开VC2017 的开发者命令行提示窗口，单击“开始”→“Visual Studio 2017”→“Visual Studio Tools”→“VS 2017 的开发人员命令提示符”,输入命令 cd d:\openssl-1.1.1b, 然后输入Perl 命令如下：

perl Configure VC-WIN32 no-shared no-asm -prefix="d:/openssl-1.1.1b/win32-release"



**Windows C/C++加密解密实战**

-openssdir="d:/openssl-1.1.1b/win32-release/slI"

其实就是把no-shard改为 shared, 后面的步骤都一样，分别是：

nmake

nmake test

nmake install nmake clean

完毕后，我们到D:\openssl-1.1.1bwin32-release下去看，可以看到生成的各个子目录。

【例2.2】测试32位Release版本的库

(1)把例子2.1的工程复制一份，然后使用VC2017 打开工程。

(2)在工具栏选择解决方案配置为Release, 如图2-12所示。

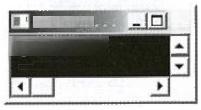
Release x86

图2-12

(3)打开“test 属性页”对话框，确保界面左上角的“配置”是Release, 意思是我们要 生成的程序是Release版本的程序，即程序中不带调试信息了。然后在界面左边选择“C/C++” →“常规”,在右边添加附加包含目录为D:\openssl-1.1.1bwin32-releaselinclude。

(4)添加静态库。在“test属性页”对话框中，在界面左边选择“链接器”→“常规”, 在右边的“附加库目录”旁输入OpenSSL 静态库所在路径D:\openssl-1.1.1blwin32-releaselib, 然后单击“应用”按钮。接着展开左边的“链接器”→“输入”,在右边第一行“附加依赖项” 右边的开头输入ws2\_32.ib;Crypt32.ib;libcryptolib;,其中 ws2\_32.1ib和 Crypt32.lib是VC 自带 的库，分别实现网络功能和微软提供的加解密功能，加入这两个库的原因是libcrypto.lib 依赖 于它们。最后单击“确定”按钮。

(5)保存工程并运行，运行结果如图2-13所示。



licroso

openssl ok



×

图2-13

至此，说明32位的Release版本的静态库使用起来了。

**6.编译出32位的Debug** **版本的动态库**

(1)解压源码目录(如果前面解压过了，就不必再解压)。把 openssl-1.1.1b.tar.gz 复制 到某个目录下，比如D:,然后解压缩，解压后的目录为D:\openssl-1.1.1b,进 入D:\openssl-1.1.1b, 就可以看到各个子文件夹了。

( 2 ) 配 置 OpenSSL 。打 开VC 2017的开发者命令行提示窗口，单击“开始” →“Visual Studio 2017”→“Visual Studio Tools”→“VS 2017 的开发人员命令提示符”,输入命令 cd



**第** **2** **章** **搭** **建C** **和C++** **密码开发环境**

d:\openssl-1.1.1b,然后输入Perl 命令如下：



perl Configure debug-VC-WIN32 shared no-asm -prefix="d:/openssl-1.1.1b/win32-shared-debug"

-openssldir="d:/openssl-1.1.1b/win32-shared-debug/sl"

后面的步骤都一样，分别是：



nmake

nmake test

nmake install nmake clean

完毕后，我们到D:\openssl-1.1.1b\win32-shared-debug 下查看，可以看到生成的各个子目录。 【例2 .3】使用32位的Debug 版本的动态库

(1)打开VC 2017,新建一个控制面板程序test。

(2)在test.cpp中输入代码如下：

#include "pch.h"

#include "openss/evp.h" int main()

{

openssl\_add\_all\_algorithms();

printf("openssl win32-debug-shard-lib okn"); return 0;

(3)包含 include 目录。打开 “test 属性页”对话框，在界面左边选择 “C/C++”→ “ 常 规”,在右边的“附加包含目录”旁输入OpenSSL 头文件所在的路径D:\openssl-1.1.1b\win32- shared-debug\include。

(4)添加链接符号库。在“test 属性页”对话框中，在界面左边选择“链接器”→“常 规”,在右边的“附加库目录”旁输入OpenSSL引用库(注意虽然名字和静态库一样，但动 态库中叫引用库，用于编译时的符号引用)所在的路径 D:\openssl-1.1.1bwin32-shared-debuglib 然后单击“应用”按钮。接着展开界面左边的“链接器”→“输入”,在右边第一行“附加依 赖项”右边的开头输入 ws2\_32.lib;Crypt32.lib;libcrypto.lib;, 其 中ws2\_32.lib 和 Crypt32.1ib 是 VC 自带的库，分别实现网络功能和微软提供的加解密功能，加入这两个库的原因是 libcrypto.lib 依赖于它们。最后单击“确定”按钮。

( 5 ) 把D:\openssl-1.1.1b\win32-shared-debug\bin 下 的libcrypto-1\_ 1.dll 复制到我们的解决

方案的debug 目录下，即和test.exe同 一个目录下。

(6)保存工程并运行，运行结果如图2- 14所示。

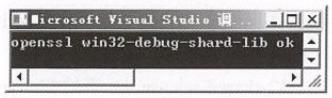


图2-14



**Windows C/C++加密解密实战**

**7.编译出32位的Release版本的动态库**

(1)解压源码目录。把 openssl-1.1.1b.tar.gz 复制到某个目录下，比如D:, 然后解压缩， 解压后的目录为D:\openssl-1.1.1b, 进 入D:\openssl-1.1.1b, 就可以看到各个子文件夹了。

(2)配置OpenSSL。打 开VC 2017的开发者命令行提示窗口，单击“开始”→“Visual Studio 2017”→“Visual Studio Tools”→“VS 2017 开发人员命令提示符”,输入命令cd d:\openssl-1.1.1b, 然后输入Perl 命令如下：



perl Configure VC-WIN32 shared no-asm --prefix="d:/openssl-1.1.1b/win32-shared-release" -openssldir="d:/openssl-1.1.1b/win32-shared-release/ssl"

其实就是把no-shard改为shared, 后面的步骤都一样，分别是：

nmake

nmake test

nmake install nmake clean

完毕后，我们到D:\openssl-1.1.1blwin32-shared-release下去看，可以看到生成的各个子目录。 【例2 .4】使用32位的release 版本的动态库

( 1 ) 打 开VC 2017, 新建一个控制面板程序 test。

(2)在 test.cpp 中输入代码如下：

#include "pch.h"

#include "openssl/evp.h" int main(

{

openssl\_add\_all\_algorithms()

printf("openssl win32-release-shard-lib ok\n") return 0;

}

(3)将界面左上角工具栏上的选择解决方案配置为Release, 意思是我们要生成的程序是 Release 版本的程序，即程序中不带调试信息了。在界面左边选择“C/C++”→ “常规”,在 右边的“附加包含目录”旁输入OpenSSL 头文件所在的路径D:\openssl-1.1.1b\win32-shared- release\include。

(4)添加链接符号库。在“test 属性页”对话框中，在界面左边选择“链接器”→“常 规”,在右边的“附加库目录”旁输入OpenSSL 引用库所在的路径D:\openssl-1.1.1b\win32-shared- release\lib,然后单击“应用”按钮。接着展开左边的“链接器”→“输入”,在右边第一行 “附加依赖项”右边的开头输入 ws2\_32.lib;Crypt32.lib;libcrypto.lib;,其 中 ws2\_32.lib 和

Crypt32.lib 是 VC 自带的库，分别实现网络功能和微软提供的加解密功能，加入这两个库的原 因 是libcrypto.lib 依赖于它们。最后单击“确定”按钮。

( 5 ) 把D:\openssl-1.1.1b\win32-shared-release\bin 下 的libcrypto-1\_ 1.dll 复制到解决方案的

release目录下，即和test.exe同一个目录下。

**第** **2** **章** **搭** **建C** **和** **C++** **密码开发环境**

(6)在工具栏切换解决方案配置为Release,保存工程并运行，运行结果如图2-15所示。



Tierosoft Visaal Stadi。词试控制台口 区 openssl vin32-releao-ahard-1i动 ok



图2-15

**8.编译出64位的Debug** **版本的静态库**

(1)解压源码目录。把openssl-1.1.1b.tar.gz 复制到某个目录下，比如D:, 然后解压缩， 解压后的目录为D:\openssl-1.1.1b,进入D:\openssl-1.1.1b,就可以看到各个子文件夹了。

(2)配置OpenSSL。打开VC 2017的开发者命令行提示窗口，单击“开始”→“Visual Studio 2017”→“Visual Studio Tools”→“VS 2017 的开发人员命令提示符”,输入命令 cd d:\openssl-1.1.1b,然后输入 Perl 命令如下：



perl Configure debug-VC-WIN64A no-shared no-asm-prefix="d:/openssl-1.1.1b/win64-debug"

- -openssldir="d:/openssl-1.1.1b/win64-debug/ssl"



debug-VC-WIN64A 表示64位调试模式。 后面的步骤都一样，分别是：

nmake

nmake test

nmake install nmake clean

完毕后，我们到D:\openssl-1.1.1bwin64-debug下去看，可以看到生成的各个子目录，如 果我们进入 lib 子目录下去看，可以发现libcrypto.lib的文件尺寸比32位的版本大了2MB多， 如图2-16所示。

8liberypto.lib 2019/8/2216.49 Obiect File Li 15,631距

图2-16

【例2.5】验证64位Debug版本的OpenSSL静态库

(1)打开VC 2017,新建一个控制面板程序test。

(2)在 test.cpp中输入代码如下：

#include "pch.h"

#include "openssl/evp.h" int main()

{

openssl\_add\_all\_algorithms0);

printf("openssl win64-debug-staticlib okln") return 0;

}

(3)在工具栏上的解决方案平台选择“x64”, 意思是我们要生成的程序是64位的程序， 如图2-17所示。



**Windows C/C++加密解密实战**

Debug x64

**解决方案平台**

图2-17

打开“test 属性页”对话框，在左边选择“C/C++”→“常规”,在右边的“附加包含目 录”旁输入OpenSSL头文件所在的路径D:\openssl-1.1.1b\win64-debuglinclude。

(4)添加链接符号库。在“test 属性页”对话框中，在左边选择“链接器”→“常规”, 在右边的“附加库目录”旁输入 OpenSSL 静态库所在的路径D:\openssl-1.1.1blwin64-debuglib, 然后单击“应用”按钮。接着展开左边的“链接器”→“输入”,在右边第一行“附加依赖项” 右边的开头输入 ws2\_32.lib;Crypt32.lib;libcrypto.lib;,其中ws2\_32.1ib和Crypt32.lib是VC自带 的库，分别实现网络功能和微软提供的加解密功能，加入这两个库的原因是libcrypto.lib依赖 于它们。最后单击“确定”按钮。

(5)保存工程并运行，运行结果如图2-18所示。



lierosoft Visual Stadio谓 回 × openss1 win64-dehug-staticlih ok 



图2-18

**9.编译出64位的** **Release版本的静态库**

(1)解压源码目录。把openssl-1.1.1b.tar.gz 复制到某个目录下，比如 D:, 然后解压缩， 解压后的目录为D:lopenssl-1.1.1b,进 入D:lopenssl-1.1.1b,就可以看到各个子文件夹了。

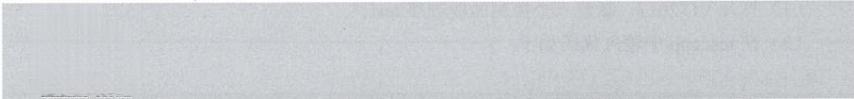
(2)配置OpenSSL。打开VC 2017的开发者命令行提示窗口，单击“开始”→“Visual Studio 2017”→“Visual Studio Tools”→“VS 2017 的开发人员命令提示符”,输入命令 cd d:\openssl-1.1.1b,然后输入Perl 命令如下：



perl Configure VC-WIN64A no-shared no-asm--prefix="d:/openssl-1.1.1b/win64-release"

--openssldir="d:/openssl-1.1.1b/win64-release/ssl"

后面的步骤都一样，分别是：



nmake

nmake test

nmake install nmake clean

完毕后，我们到D:\openssl-1.1.1blwin64-release下去看，可以看到生成的各个子目录。 【例2.6】验证64位Release版本的OpenSSL静态库

(1)打开VC 2017,新建一个控制面板程序 test。

(2)在 test.cpp中输入代码如下：

"stdafx.h"

#include

#include

"openssl/evp.h"



**第** **2** **章** **搭** **建C** **和C++** **密码开发环境**

#pragma comment(lib,"libcrypto.lib") #pragma comment(lib,"ws2\_32.lib") #pragma comment(lib,"crypt32.lib")

int main() {

openssl\_add\_all\_algorithms();

printf("openssl win64-release-staticlib ok\n"); return 0;

#pragma comment表示以代码方式引用库。这样就不用在工程属性中设置了。

( 3 ) 打 开“test 属性页”对话框，新建一个x64 平台，在工程属性中切换到Release 模 式 ， 然后添加头文件包含路径： D:\openssl-1.1.1b\win64-releaselinclude,以及静态库路径：

D:\openssl-1.1.1b\win64-release\lib。这里讲的简略了，路径具体在哪个位置添加，前面已经介 绍过了。

(4)保存工程，然后在工具栏上选择解决方案平台为x64, 解决方案配置为Release, 然 后运行工程，运行结果如图2- 19所示。



国C:\Tindorslsysteo32\cod.ex □[× openga1 win64-release-staticlih ok

请按任意键维续..



图2-19

**10** **.** **编译出64位的** **Debug 版本的动态库**

(1)解压源码目录。把openssl-1.1.1b.tar.gz 复制到某个目录下，比如D:, 然后解压缩， 解压后的目录为D:\openssl-1.1.1b,进 入D:\openssl-1.1.1b,就可以看到各个子文件夹了。

(2)配置OpenSSL 。打 开VC 2017的开发者命令行提示窗口，单击“开始”→“Visual Studio 2017”→“Visual Studio Tools”→“VS 2017的开发人员命令提示符”(注意是x64 本 机 工 具命令提示，不要选择其他的),输入命令cd d:\openssl-1.1.1b,然 后 输 入Perl 命令如下：



perl Configure debug-VC-WIN64A shared no-asm --prefix="d:/openssl-1.1.1b/win64-shared-debug" --openssldir="d:/openssl-1.1.1b/win64-shared-debug/ssl"

后面的步骤都一样，分别是：

nmake

nmake test

nmake install nmake clean

完毕后，我们到D:\openssl-1.1.1b\win64-shared-debug 下去看，可以看到生成的各个子目录。



**Windows C/C++加密解密实战**

【例2.7】验证64位 Debug版本的OpenSSL 动态库

(1)打开VC 2017,新建一个控制面板程序test。

(2)在test.cpp中输入代码如下：

#include "stdafx.h"

#include "openssl/evp.h"

#pragma comment(lib,"libcrypto.lib") #pragma comment(lib,"ws2\_32.lib") #pragma comment(lib,"crypt32.lib")

int main( {

openssl\_add\_all\_algorithms()

printf("openssl win64-debug-shared-lib ok\n"); return 0;

#pragma comment表示以代码方式引用符号库。这样就不用在工程属性中设置了。

(3)打开 “test 属性页”对话框，新建一个 x64 平台，然后添加头文件包含路径：

D:\openssl-1.1.1bwin64-shared-debuglinclude,以及符号库路径：D:\openssl-1.1.1b\win64-shared- debuglib。这里讲的简略了，路径具体在哪个位置添加，前面已经介绍过了。

把D:\openssl-1.1.1b\win64-shared-debug\bin下的libcrypto-1\_ 1-x64.d1l复制到解决方案路径 下的x64 文件夹下的 Debug子目录下。

(4)保存工程，然后在工具栏上选择解决方案平台为x64, 然后运行工程，运行结果如 图2-20所示。

应CA\Tindos\systeo32\cd ox

openanlwin64-debug-ahared-1ib ok 请按任意键继续..

图2-20

口 区

D

**11.编译出64位的Release版本的动态库**

(1)解压源码目录。把 openss-1.1.1b.tar.gz 复制到某个目录下，比如D:, 然后解压缩， 解压后的目录为D:\openssl-1.1.1b,进 入D:\openssl-1.1.1b,就可以看到各个子文件夹了。

(2)配置OpenSSL。打开VC2017 的开发者命令行提示窗口，单击“开始”→“Visual Studio 2017”→“Visual Studio Tools”→“VS 2017的开发人员命令提示符”(注意是x64 本机工 具命令提示，不要选择其他的),输入命令cdd:\openssl-1.1.1b, 然后输入Perl 命令如下：

perl Configure VC-WIN64A shared no-asm--prefix="d:/openssl-1.1.1b/win64-shared-release" --openssldir="d:/openssl-1.1.1b/win64-shared-release/ssl"

后面的步骤都一样，分别是：



**第** **2** **章** **搭** **建C** **和C++** **密码开发环境**

nmake

nmake test

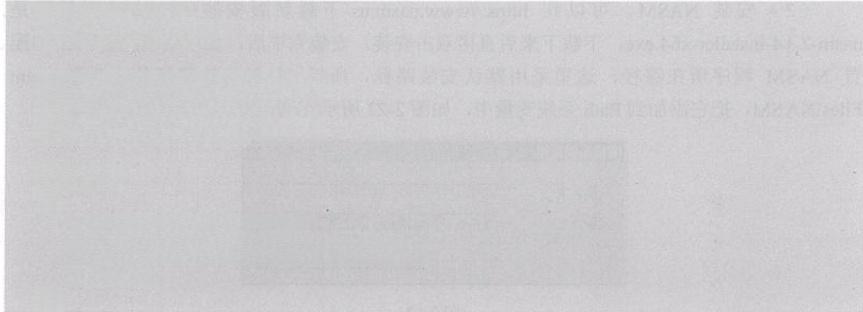
nmake install nmake clean

完毕后，我们到D:\openssl-1.1.1b\win64-shared-release下去看，可以看到生成的各个子目 录。

【例2.8】验证64位Release 版本的OpenSSL动态库

(1)打开VC 2017,新建一个控制面板程序test。

(2)在 test.cpp中输入代码如下：



#include "stdafx.h"

#include "openssl/evp.h"

#pragma comment(lib,"libcrypto.lib") #pragma comment(lib,"ws2\_32.lib") #pragma comment(lib,"crypt32.lib")

int main( {

openssl\_add\_all\_algorithms();

printf("openssl win64-release-shared-lib ok\n"); return 0;

}

#pragma comment表示以代码方式引用符号库。这样就不用在工程属性中设置了。

(3)打开“test 属性页”对话框，新建一个x64 平台，在工程属性中切换到Release 模式， 然后添加头文件包含路径：D:\openssl-1.1.1blwin64-shared-release\include,以及静态库路径： D:\openssl-1.1.1b\win64-shared-releaselib。这里讲的简略了，路径具体在哪个位置添加，前面 已经介绍过了。

(4)保存工程，然后在工具栏上选择解决方案平台为x64, 解决方案配置为Release, 并 把 D:\openssl-1.1.1b\win64-shared-release\bin下 的libcrypto-1\_ 1-x64.dll 复制到解决方案的x64 文件夹下的Release 文件夹下，然后运行工程，运行结果如图2-21所示。



C:ATindovs\systen32\ead.exe

openggl win64-release-shared-1ih

请按任意键维续..





口

ok

×

图2-21

以上我们对新版本的OpenSSL的各种库都进行了编译和测试，虽然略显烦琐，但也是必 要的，尤其是在实际项目中使用之前，建议大家都测试一下，库好才用。



**Windows** **C/C++加密解密实战**

**2.2.12 在Windows下编译OpenSSL 1.0.2m**

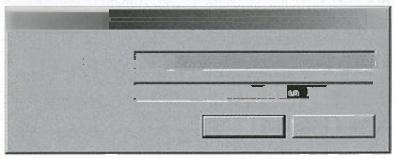
这个1.0.2版本属于当前主流使用的版本，无论是维护老项目，还是开发新项目，这个版 本都用得比较多，因为其成熟、稳定。尤其对于信息安全相关的项目，建议大家不要直接使用

很新的算法库，因为可能有潜在的Bug 没有被发现。该版本下载地址：<https://www.openssl.org/> source/old/。

这里我们下载 openssl-1.0.2m.tar.gz,把它复制到C: (也可以是其他目录),然后按照下 面的步骤开始编译和安装。这里我们编译32位的Debug版本的动态库。

(1)安装ActivePerl。这个软件我们前面已经介绍过了，这里不再赘述。

( 2 ) 安 装 NASM。可 以 到 <https://www.nasm.us/>下载新版安装包，这里下载的是 nasm-2.14-installer-x64.exe,下载下来后直接双击安装。安装完毕后，要在系统变量Path 中配 置 NASM 程序所在路径，这里采用默认安装路径，所以 NASM 的路径是：C:\Program Files\NASM, 把它添加到Path 系统变量中，如图2-22所示。



Path

Fo\Tools\Bimn\**CAFro**

确定

×

**FErWS**

取消

描系统变里

变量名0): 变量值V):

图2-22

单击“确定”按钮。然后打开一个命令行窗口，输入命令nasm, 此时界面显示如图2-23 所示。



应管理员：C:\Yindorslsyste32\cud.er 回

Hicrosoft Vindowe[版本6.1.

版权所有(c)2009 Hicrosoft

a:e:tMalA:ittorflneaecified

type 'nasn -h'for help

C:Wsers Administrator>



26011

Corporation。保留所



日

图2-23

这说明NASM安装并配置成功了。好了，准备工作完成，可以正式开始编译OpenSSL了。 为了让大家知道不指定目录OpenSSL 会把生成的文件放在哪里，我们先在不指定路径的情况 下进行编译。

**1.不指定生成目录的32位Release版本动态库的编译**

(1)解压OpenSSL源码目录。把 openssl-1.0.2m.tar.gz复制到某个目录下，比如C:, 然 后解压缩，解压后的目录为C:\openssl-1.0.2m, 进 入C:\openssl-1.0.2m, 就可以看到各个子文 件夹了。



**第** **2** **章** **搭** **建C** **和C++** **密码开发环境**

(2)配置OpenSSL。打 开VC 2017的 “VS 2017的开发人员命令提示符”提示窗口，单 击“开始”→“所有程序”→“Visual Studio 2017”→“Visual Studio Tools”→“VS 2017的 开发人员命令提示符”,输入命令如下：

cd C:\openssl-1.0.2m

perl Configure VC-WIN32

ms\do nasm

nmake -fmsintdll.mak

VC-WIN32 表示生成 Release 版本的32位的库，如果需要 Debug 版本，就使用 debug-VC-WIN32。ntdll.mak 表示即将生成动态链接库。执行完毕后，我们可以在 C:\openssl-1.0.2m\out32dll\下看到生成的动态链接库，比如libeay32.dll,如图2-24所示。



out32a

▼计算机▼本地磁盘(C:)▼ openssl-1.0.2mv out32dl1

组织▼ 吃 View Dependencies▼ 刻录 新建文件夹

曰☆收藏来 下载 桌面

归最近访问的位置

曰后库

padlock.dll

nur on.dll

libeay32.dl1

gost.dll

名称

图2-24

该文件夹除了包括动态库外，相关的导入库文件(比如libeay32.lib和 ssleay32.lib)和一 些可执行的工具(.exe) 程序也在该目录下。导入库文件在开发中需要引用，所以我们需要知 道它的路径。

头文件文件夹所在的路径是 C:\openssl-1.0.2m\inc32\,开发的时候我们可以把inc32 下的 OpenSSL 文件夹复制到工程目录，再在VC 工程设置中添加引用，就可以使用头文件了。当 然，不复制到工程目录也可以，只要在VC中引用到这里的路径即可。稍后我们会通过实例来 演示如何使用这里编译出来的动态库。如果大家觉得在OpenSSL 目录下去找这个子目录很麻 烦，也可以执行安装命令：nmake -f ms\ntdll.mak install,执行该命令后，将会把 include 文件 夹、lib文件夹、和bin文件夹复制到在C:\usrlocal\ssl下，有兴趣的读者可以试试。

重要提示：如果编译过程中出错，建议把 C:\openssl-1.0.2m这个文件夹删除，然后重新解 压，再按上面的步骤执行。

好了，下面我们进行指定生成目录下的编译。一般这种方式用得多，这样可以和OpenSSL 源码目录分离开来。

**2.指定生成目录的32位** **Release版本动态库的编译**

( 1 ) 如 果 C 盘已经有 openssl-1.0.2m 文件夹，就解压OpenSSL 源码目录。把 openssl-1.0.2m.tar.gz 复制到 C:, 然后解压缩，解压后的目录为 C:\openssl-1.0.2m,进入 C:\openssl-1.0.2m,就可以看到各个子文件夹了。如果C 盘已经有openssl-1.0.2m文件夹，可



**Windows C/C++加密解密实战**

以不用再解压。

(2)配置OpenSSL。打 开VC 2017的 “VS 2017的开发人员命令提示符”提示窗口，单 击“开始”→“所有程序”→“Visual Studio 2017”→“Visual Studio Tools”→“VS 2017的 开发人员命令提示符”,输入命令如下：

cd C:\openssl-1.0.2m

perl Configure VC-WIN32--prefix=c:/myOpensllout

ms\do\_nasm

nmake -fms\ntdll.mak

--prefix 用于指定安装目录，就是生成的文件存放的目录；VC-WIN32 表示生成 Release 版本的32位的库，如果需要 Debug 版本，就使用debug-VC-WIN32。稍等片刻，编译完成， 如图2-25所示。



vS201 的开发人员命令提示符 - 回×

IF EKIST out32d¹1Uhad\_dt1s\_test.exe.nanifest nt -mologo -n l1 had\_dtls\_test.exe.nanifest -outputresourceiout32d11 had\_dtls\_tes

link /nologo /cuhsystensconsole /opt:ref /debug /out :out32d e EC:Wsers ADHIHI~1AppData Local\Tenp Min?D35.tnp

Great ing library tnp32d11Njunk.1ih and ohject tnp32dl1junk.oxp

IF EXIST out32dl1\igetest.exo.nanifest nt -nologo -nanifest test.exe.nanifest -outputresource iout32dii\igetest.exei1

link /nologo /uhs ysten:console /opt :ref /debug /out:out32d 。PC:Wsera ADHIMI~1AppData Local\Tenpvn7FBn.tnp

Greating library tnp32dlI\junk.1ih and object tnp32dll\junk.exp

IF EXIST out32d1l\openssl.exe.nanifest nt -nologo-nanifest

nssl.exe.nanifest -outputresource:out32dl1\openss1.exe;1

oensal-1.8.2m>-

图2-25

此时可以看到C 盘下并没有 myOpensslout,因为我们还没有执行安装命令，但可以在 C:\openss-1.0.2m\out32dI下看到生成的动态链接库，比如 libeay32.dll。头文件文件夹OpenSSL 所在的路径为C:\openssl-1.0.2mlinc32\。下面执行安装命令：

nmake -f msintdll.mak install

执行完毕后，我们看到C 盘下有myOpensslout 了，如图2-26所示。



oyOpensslout

计算机▼本地磁盘(C:)▼myOpensslout▼

组织▼ 包含到库中▼

☆收藏夹 下载 桌面

9最近访问的位置

库

日视频

图2-26

如果喜欢干净，可以用nmake -fms\ntdll.mak clean命令清理一下。

共享▼刻录 名称

.bin

include lib

ssl

新建文件夹

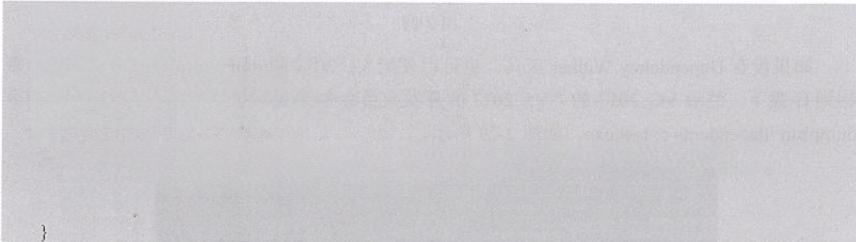


**第** **2** **章** **搭** **建C** **和C++** **密码开发环境**

至此，32位动态库编译安装完成。下面进入验证环节。 【例2.9】验证32位动态库

(1)新建一个控制面板工程test。

(2)打开 test.cpp, 输入代码如下：



#include "stdafx.h"

#include "openssl/evp.h"

#pragma comment(lib,"libeay32.lib") int\_tmain(int argc,\_TCHAR\*argv[]) {

openssl\_add\_all\_algorithms();// 载入所有SSL 算法，这个函数是OpenSSL 库中的函数 printf("win32 openssl1.0.2m-shared-lib ok\n");

return 0;

(3)打开工程属性对话框，然后添加头文件包含路径：C:\myopensslout\include,以及导 入库路径：C:\myopensslout\lib。如果此时运行程序，系统干净的朋友是无法运行的，会提示 缺少动态库，如图2-27所示。

**test.cre-系续错误** 凶

×

无 法 肩 动 些 程 序 ， 因 为 计 算 机 中 丢 失LIBEAY32.d1。 尝 试 重

新安装该程序以解决此问题 。

确定

图2-27

但有些朋友发现可以直接运行，难道上面生成的lib 文件是静态库，而不是导入库。其实 是导入库，我们可以验证一下。打开VC 2017的“VS 2017的开发人员命令提示符”提示窗口， 即单击“开始”→“所有程序”→“Visual Studio 2017”→“Visual Studio Tools”→“VS 2017 的开发人员命令提示符”,输入命令如下：

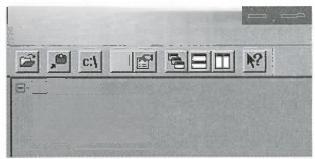
lib/list C:\myopensslout\liblibeay32.lib

如果输出的是LIBEAY32.dll,就说明libeay32.1ib是一个导入库，如果输出的是.obj, 就说 明是静态库。既然不是静态库，为何能运行起来呢?说明系统路径肯定存在libeay32.dll。大家 可以去C:\Windows\SysWOW64或C:\Windows\System32 等常见系统路径下搜索，如果删掉或 重命名后还能运行test.exe,就说明安装了某些软件导致 test.exe依然能找到libeay32.dll, 比如 安装了 ice3.7.2 这个通信库。或许有朋友到这里有点怀疑 test.exe是否真的依赖libeay32.d,

大家可以验证一下，如果有Dependency Walker 工具就查看一下依赖项，如图2-28所示。



Windows C/C++加密解密实战

pondngy Tolker est.e] File Edit Vier Windov Halp



□ 」TEST.EXE

田□MSVCR120D.DEL 田□LIBEAY32.DL

田 □KERNEL32.DL

图2-28

如果没有Dependency Walker 工具，也可以使用VC 2017的 dumpbin 程序，把 test.exe 复 制到C 盘下，然后VC 2017的“VS 2017的开发人员命令提示符”提示窗口，然后输入命令 dumpbin/dependents c:\test.exe,如图2-29所示。



四 管 理 员 ：VS 2017的开发人员命令提示符 l 回×

cEProgran Files(x86)Hicrosoft Uisual Studio 2817\Enterprise>

c VProgran Piles (×86)Microsoft Uisual studio 2017 Enterprise>dunpbin /dopenden ts c:\test.exe

Hicrosoft(R)COFF/PEDunperUersion 14.16.27030.1

copyright (C>Hicrosoft Corporation.Al1 righta reserved.

Dunp of file c:test.exe

File Iype:BXECUTABLE IaGE

Inagehan the following dependencies:

MSUCR128D.dl1 LI BEAY32.dl1 KERNEL32.dl1

**Sunnary**

1808 .data 1808 .idata

3800 .rdata

1000 .reloc BPB.rsre

4090 .text

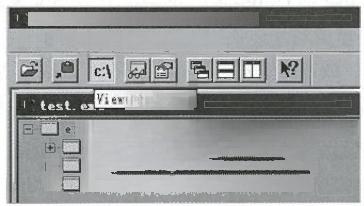
10000 .texthas

EProTan Fil (x6)Hicronoft Uiual Studio9017Entorprino>

图2-29

可以看到的确依赖libeay32.dl 。另外，可以发现把 test.exe 放到一个干净的操作系统上， 就运行不起来了。如果一定要知道 libeay32.dll 在哪里，也不是没有可能。大招就是使用 Dependency Walker,这个依赖项查看工具自VC6 开始就自带了，后来高版本的VC 虽然不带 它了，但可以从官网(www.dependencywalker.com)上下载。这里还是使用VC 6自带的1.0 版。我们把 test.exe 拖进Dependency walker工具，然后单击工具栏上的c:l 按钮，它用于显示

全路径，如图2-30所示。



**Dependency Tnlker -test.exe**

File Edit View Window Help

Full Path

Aebook\m密码\mnote\分章\2搭建c和c++密码

e:\windows\system32\MSVCR120D.DLL

田 **卧WindowNsyzten32MIBEA32.页** 田 **c:\windows\system32\KERNEL32.DLL**

图2-30



**第** **2** **章** **搭** **建C** **和** **C++** **密码开发环境**

可以看到，test.exe 依赖的libeay32.dll位于system32 下，终于找到元凶了，把它删除再运 行 test.exe 会发现无法运行。

绕了一大圈，让系统干净的朋友久等了，继续把C:\myopensslout\bin 下的libeay32.dll 复 制到解决方案路径下的Debug子目录下，即和test.exe同一文件夹下。

(4)保存工程并运行，运行结果如图2-31所示。



园 C Yiados\systs32\cd ae

vin32 openssl1.8.2n-shared-lih ok

请按任意键维续 . . 



D

图2-31

这里测试工程 test 用了Debug 模式，而库libeay32.dll是Release 版本的，这是没问题的。

**3** **.编译64位Release版本的动态库**

首先把C 盘下的openssl-1.0.2m文件夹删除(如果有的话),然后按照下面的步骤进行：

(1)解压OpenSSL 源码目录。把 openssl-1.0.2m.tar.gz复制到某个目录下，比如C:, 然 后解压缩，解压后的目录为C:\openssl-1.0.2m, 进 入C:\openssl-1.0.2m,就可以看到各个子文 件夹了。

(2)配置OpenSSL。单击“开始”→“Visual Studio 2017”→“Visual Studio Tools”→ “VS 2017的开发人员命令提示符”,打开VC 2017的 “VS 2017的开发人员命令提示符”窗 口，输入命令如下：

cd C:\openssl-1.0.2m

perl Configure VC-WIN64A no-asm --prefix=c:/myopensslstout64 ms\do\_win64A

nmake -fms\ntdll.mak

--prefix用于指定安装目录，就是生成的文件存放的目录。VC-WIN64A 表示生成Release 版本的64位的库，如果需要Debug版本，就使用debug-VC-WIN64A。稍等片刻，编译完成， 如图2-32所示。



bpenssl.c

EG:Ws\AoINgI1/atlp: . bug /out:out32dl1wopens

正在创建库 tnp32d11 junk.1ib 和对象 tnp32d11\junk.exp

IF EXIST out32d11\openss1.exe.nanifest nt -nologo nanifest out32a1

hss1.exe .nanifest -outputresource:out32d11\openss1.exe;1

**Nopenss1-1.0.2n>**

图2-32

此时我们看到C 盘下并没有文件夹myopensslout64, 这是因为还没有执行安装命令，但可 以在 C:\openssl-1.0.2m\out32d1I\下看到生成的动态链接库，比如 libeay32.dll。头文件文件夹 OpenSSL所在的路径为C:\openssl-1.0.2m\inc32\,有些多疑的读者可能会疑惑，为何64位的.dll

文 件 会 生 成 在 名 字 是out32dll 的 文 件 夹 下 ， 看 名 字out32dll 像 是 存 放 3 2 位 的 库 。 笔 者 认 为 这

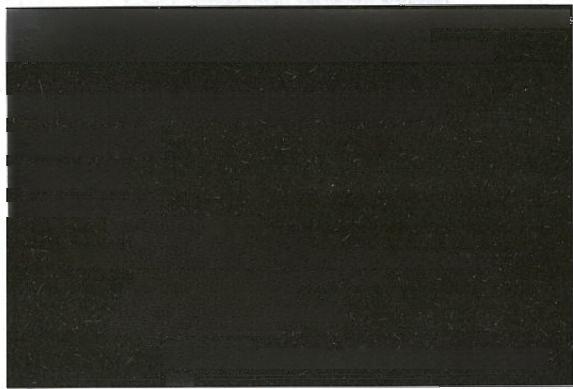


**Windows** **C/C++加密解密实战**

是 OpenSSL 官方偷懒的地方，这样的文件夹名字的确容易引起歧义，为了消除读者的疑惑， 我们可以验证一下生成的 libeay32.dll 到底是32位还是64位的，方法有多种：

(1)在“VS 2017的开发人员命令提示符”窗口的提示符下输入命令： dumpbin/headers c:\openssl-1.0.2m\out32dlNlibeay32.dl

如果出现 machine(x64) 字样，就说明该库是64位库，如图2-33所示。



正openss1-1.0.2m>dunphin /headers C=openss1-1.0.2n\out32dl1\lihe Hicrosoft(R>COFF/PEDunperUersion 12.00.30723.0

sopyright (C>Hicrosoft Corporation.Al1 rights reserved.

**bunp of file C:\openss1-1.8.2n\out32dl1libe ay32.dl1**

PE signature found

File Iype:DLL

FILE HEADER UALWES

8664 machine (x64)

6 nunber of sections

5CCOFC4E time date stamp Thu Apr 2588:16:142019

B file pointer to synbol table I number of synbols

FO size of optional header

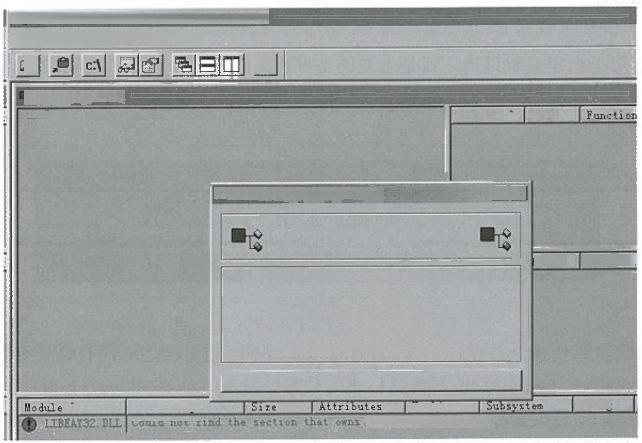
2022 characteristics

Executable

Application can handle large (>2GB)addresses DLL

图2-33

(2)如果安装的是VC6, 可以用VC6 自带的Dependency Walker工具来查看，因为VC 6自带的该工具(版本是1.0)只能查看32位的动态库，所以64位的库拖进去是看不到信息 的，如图2-34所示。



Dependeney Tolker=iboy32.m1 File Edit View Window Help

 K?

libeay32.ai

① LIBEAY32.DIL 0rdinal Hint

About Dependeney Yalker ? 区

Dependency Yalker

tint Funetion

Dependency Walker for Yin32 Intel x86) Version 1.0

Steve P.Miller (stavemil0microsoft.com) Copyright ?1996 Microsoft Corporation

onDet 251996 at 15:42:48

0X

TMachine

the Ioport Directory

Tiaa Stamp

Built

nahu

图2-34



**第** **2** **章** **搭** **建C** **和** **C++** **密码开发环境**

当然，现在高版本的 Dependency Walker 工具已经可以同时查看32位和64位的库了。不 过 ，VC 2017不自带这个小工具，如果读者需要的话可以去官网(http://www.dependencywalker. com/)下载。

只有执行了安装命令才会把生成的库、头文件等放到我们指定的目录 myOpenSSLout64 下。下面执行安装命令：

nmake -f ms\ntdll.mak install

执行完毕后，如图2-35所示。



Copying:out32a11/libeay32.1ib to c:/nyopenss lout64/lib/liheay32.1ih

perl util/mkdir-p.pl 'c:vnyOpensslout64\lib \engines"

created directory 'c:/myOpensslout64/lib/engines'

perl util/copy.pl out32d11\4758cca.dll out32d1l aep.dll out32d1l ats

dll out32dll\cswift.dll out32dll\gnp.dll out32dl1vchil.dll out32dllvuron.

out32dllsureware.dll out32dl1ubsec.dll out32dll1\padlock.dl1 out32dl1ycap out32d11\gost.dl1 'c:nyOpensslout64\lib\engines"

|  |  |
| --- | --- |
| Copying:out32dl1/4758cca.dll | to c:/nyOpensslout64/lib/engines/4758cca.dl1 |
| Copying:out32dl1/aep.dll | to c:/my0pensslout64/lib/engines/aep.dl1 |
| Copying:out32dl1/atalla.dll | to c:/nyOpensslout64/lib/engines/atalla.dll |
| Copying:out32dl1/cswift.dll | to ci/myOpensslout64/lib/engines/cswift.dl1 |
| **ConvingE out32dll∠mn.dll** | **to ci/mvOpensslout64/lib/enginessmp.dll** |

图2-35

仔细看图2-35,我们发现其实就是把out32dll下的内容复制到C:/myopensslout64下。此 时我们看到C 盘下有myopensslout64了。至此，64位的动态库编译安装完成。下面进入验证 阶段。

【例2.10】验证64位动态库

(1)新建一个控制面板工程test。

(2)打开test.cpp, 输入代码如下：

#include "stdafx.h"

#include "openssl/evp.h"

#pragma comment(lib,"libeay32.lib") int\_tmain(int argc,\_TCHAR\*argy[]) (

openssl\_add\_all\_algorithms();// 载入所有SSL 算法，这个函数是OpenSSL库中的函数 printf("win64 openssl1.0.2m-release-shared-lib okln")

return 0;

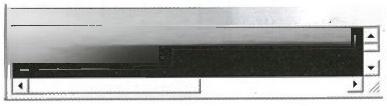
礼 j

(3)打开工程属性对话框，新建一个x64 平台，在工程属性中切换到Release 模式，然 后添加头文件包含路径：C:\myopensslout64linclude,以及导入库路径：C:\myopensslout64\lib。

(4)保存工程，然后在工具栏上选择解决方案平台为x64, 解决方案配置为Release, 并 把 C:\myopensslout64\bin 下 的libeay32.dll 复制到解决方案的x64 文件夹下的Release 文件夹下， 然后运行工程，运行结果如图2-36所示。



**Windows C/C++加密解密实战**



四C:ATindovs\systen32\end.ex -|回×

**vin64 openss11.0.2m-release-shared-1ih ok**

请按任意键继续..

图2-36

**4. 编译64位** **Debug 版本的静态库**

本来编译完动态库想结束本节的讲解，但考虑有朋友喜欢静态库，所以笔者再演示一下静 态库的编译过程。首先把C 盘下的openssl-1.0.2m文件夹删除(如果有的话),然后按下面的 步骤进行：

(1)解压 OpenSSL 源码目录。把 openssl-1.0.2m.tar.gz 复制到某个目录下，比如C:, 然 后解压缩，解压后的目录为C:\openssl-1.0.2m,进 入 C:\openss-1.0.2m,就可以看到各个子文 件夹了。

(2)配置OpenSSL。单击“开始”→“所有程序”→“Visual Studio 2017”→“Visual Studio Tools”→“VS 2017的开发人员命令提示符”,打开VC 2017的“VS 2017的开发人员命令提 示符”窗口，输入命令如下：



cd C:\openssl-1.0.2m

perl Configure debug-VC-WIN64A--prefix=c:/myopensslout64 ms\do\_win64A

nmake -fmslntdll.mak

--prefix用于指定安装目录，就是生成的文件存放的目录。VC-WIN64A 表示生成Release 版本的64位的库，如果需要Debug版本，就使用debug-VC-WIN64A。稍等片刻，编译完成。

**2.2.13** **在** **Linux** **下编译安装OpenSSL** **1.0.2**

打开官网下载源码。 OpenSSL 的官网地址是 <htps://www.openssl.org> 。 这里使用的版本是 1.0.2m,不求最新，但求稳定，这是一线开发者的原则。另外要注意的是，OpenSSL 官方现在 已停止对0.9.8 和1.0.0两个版本的升级维护。这里下载下来的是一个压缩文件： openssl-1.0.2m.tar。

**1.卸载当前已有的版本**

刚下载下来不能马上安装，先要看看现在的操作系统是否已经安装OpenSSL 了，可以用 以下命令进行查看：

[root@localhost ~]#rpm -ql openssl 或者直接查询OpenSSL 版本：

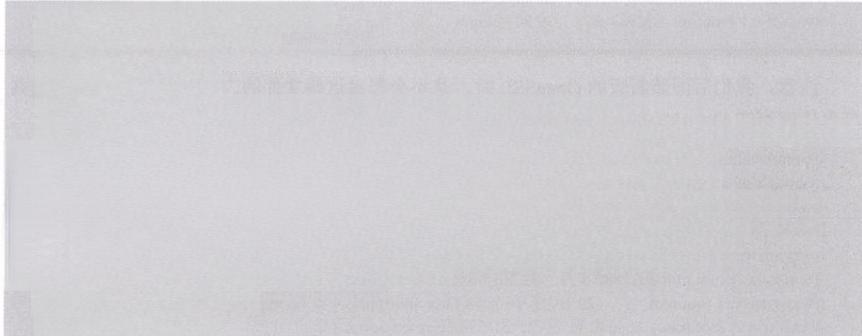
[root@localhost ~]#openssl version opensst 1.0.1e-fips 1l feb 2013

可以看出，在笔者的CentOS 7上已经预先安装了OpenSSL 1.0.1e版本，如果要查看这个



**第** **2** **章** **搭** **建C** **和C++** **密码开发环境**

版本更为详细的信息，可以输入命令：



[root@localhost ~]#openssl version -a openssl 1.0.1e-fips 11 Feb 2013

built on:Mon Jun 2912:45:07 UTC 2015 platform:linux-x86\_64

options:bn(64,64)md2(int)rc4(16x,int)des(idx,cisc,16,int)idea(int)blowfish(idx)

compiler:gcc -fPIC-DOPENSSL\_PIC-DZLIB-DOPENSSL\_THREADS-D\_REENTRANT -DDSO\_DLFCN -DHAVE\_DLFCN\_H-DKRB5\_MIT-m64 -DL\_ENDIAN-DTERMIO-Wall -02 -g-pipe -Wall

-Wp,-D\_FORTIFY\_SOURCE=2-fexceptions -fstack-protector-strong --param=ssp-buffer-size=4

-grecord-gcc-switches -m64 -mtune=generic -Wa,-noexecstack -DPURIFY-DOPENSSL\_IA32\_SSE2 -DOPENSSL\_BN\_ASM\_MONT-DOPENSSL\_BN\_ASM\_MONT5 -DOPENSSL\_BN\_ASM\_GF2m

-DSHA1\_ASM-DSHA256\_ASM-DSHA512\_ASM-DMD5\_ASM-DAES\_ASM-DVPAES\_ASM -DBSAES\_ASM-DWHIRLPOOL\_ASM-DGHASH\_ASM

OPENSSLDIR:"/etc/pki/tls" engines:rdrand dynamic

其实，也就是加了-a 选项。如果要查看OpenSSL所在的路径，可以使用whereis openssl 命令，比如：



[root@localhost bin]#whereis openssl

openssl:/usr/bin/openssl/usr/lib64/openssl /usr/include/openssl /usr/share/man/manl/openssl.lssl.gz

其中，/usr/bin/ 下的openssl 是一个程序；/usr/lib64/openssl 是一个目录；/usr/include/openssl 也是一个目录，里面存放的是开发所用的头文件。

因为我们要用OpenSSL 1.0.2m,所以要先卸载这个自带的旧版本，卸载命令如下：

[root@localhostsof]#rpm -e --nodeps openssl

然后再次查看：

[root@localhost sofl]#rpm-qa openssl [root@localhost soft]#

或者再次查看其版本：

[root@localhost 桌面]#openssl version

bash:/usr/bin/openssl:没有那个文件或目录

可以看到/usr/bin 下 的 程 序OpenSSL 没有了，说明卸载成功了。但要注意，有些目录并没 有删除，我们可以用whereis查看 一 下：

[root@localhost openssl-1.0.2m]#whereis openssl

openssl:/usr/lib64/openssl/usr/include/openssl

我们可以进入/usr/include/openssl/ 下查看，头文件依旧存在，当我们用11命令查看时，可 以发现是2015年生成的：

[root@localhost openssl]#cd /usr/include/openssl [root@localhost openssl]#l

总用量1580

-rW-r--I--.1 root root 55076月292015 aes.h -rW-r--I--.I root root 522526月292015 asnl.h

**C/C++加密解密实战**

**Windows**

-rW-r--I--.1 root root 191436月 292015 asnl mac.h -rW-r--r-.1 root root 300926月 292015 asnlt.h

-rW-r--r--.1 root root 329876月292015 bio.h

注意，我们后面装新版的 OpenSSL 时，是不会覆盖这些文件的。另外，/usr/lib64下的共

享库依旧存在libcrypto.so.1.0.1e:

[root@localhost lib64]#cd /usr/lib64/ [root@localhostlib64]#l libcry\*

-rWXr-Xr-x.I root root 4081611月202015 libcrypt-2.17.so

lrwxrwxrwx.1 root root 1910月162018 libcrypto.so->libcrypto.so.1.0.1e

lrwxrwxrwx.1 root root 1910月162018 libcrypto.so.10->libcrypto.so.1.0.1e

-rWxr-Xr-x.I root root 20128806月292015 libcrypto.so.1.0.1e

Irwxrwxrwx.I root root 2210月162018 libcryptsetup.so.4->libcryptsetup.so.4.7.0

-rWXr-Xr-x.I root root 16664011月212015 libcryptsetup.so.4.7.0

Irwxrwxrwx.I root root 2510月162018 libcrypt.so->..1./lib64/libcrypt.so.1

Irwxrwxrwx.1 root root 1610月162018 libcrypt.so.1->libcrypt-2.17.so

这里，我们可以直接把目录/usr/include/openssl、动态库文件/usr/lib64/libcrypto.so.1.0.1e和 符号链接文件/usr/lib64/libcrypto.so删除。当然，以后这3样都要在安装新版本OpenSSL时手 工恢复成新版本OpenSSL对应的内容。为了怕大家遗忘，这里先不删，在下一节安装后再删 除也可以。

当然，用到1.0.2m的例子，其实用1.0.1e 也是可以的。这里主要是为了让大家学会卸载 和重新安装。

**2** **.不指定安装目录安装OpenSSL**

假设旧版本已经卸载。把下载下来的压缩文件放到Linux 中，这里存放的路径是/root/soft, 大家可以自定义路径，然后进入这个路径后解压缩：

[root@localhost~]#cd/root/soft

[root@localhost soft]#tar zxf openssl-1.0.2m.tar.gz

进入解压后的文件夹，开始配置、编译和安装：

[root@localhost soft]#cd openssl-1.0.2m/

[root@localhost openssl-1.0.2m]#./config shared zlib

shared 表示除了生成静态库外，还要生成共享库，如果仅仅想生成静态库，可以不用这个 选项，或者使用no-shared;zlib 表示编译时使用zlib 这个压缩库。更多配置选项可以参考源 码目录下的configure 文件。

下面开始编译：

[root@localhost openssl-1.0.2m]#make

稍等片刻，编译结束。编译完成并不会复制新的文件到默认目录，我们可以使用whereis 看一下：

[root@localhost openssl-1.0.2m]#whereis openssl



**第2章** **搭** **建C** **和** **C++** **密码开发环境**

openssl:/usr/lib64/openssl /usr/include/openssl

依旧是这两个目录，我们进入/usr/include/openssl/看看里面的文件有没有被更新：



[root@localhost openssl-1.0.2m]#cd /usr/include/openssl/ [root@localhost openssl]#Ⅱ

总用量1580

-rW-r--r--.1 root root 55076月292015 aes.h -rW-r--r--.1 root root 522526月292015 asnl.h

-rW-r--r--.1 root root 191436月292015 asnl\_mac.h

可以看出，没有被更新。而且我们用make install 安装新版本的OpenSSL 后，也不会被更 新。这一点要注意，开发时不要去引用这个目录下的头文件。

[root@localhost openssl-1.0.2m]#make install

稍等片刻，安装完成。通过查看make install的过程可以发现新建了几个目录，如图2-37 所示。



**make[1]: 对** **all” 无需做任何事。**

**e***t***[****:***i**t* *l/*o*s*p*l*nssl-1.0.2m/tools”

created directory /usr/local/ssl/man

created directory'/usr/local/ssl/man/manl

created directory /usr/Local/ssl/man/man3'

created directory '/usr/local/ssl/man/man⁵'

ciigdlnrl/ssl/man/man7'

openssl-asnl parse.1 →asnl parse.1

**installing manl./CA.pl.1**

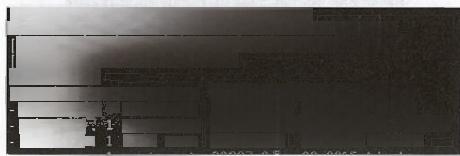
图2-37

从图2-37可以看出，安装程序创建了目录/usr/local/ssl,这个目录就是不指定安装目录时 安装程序所采用的默认安装目录。我们可以进入这个目录下查看：

[root@localhost openssl]#cd/usr/local/ssl [root@localhost ssl]#ls

bin certs include lib man misc openssl.cnf private

其中，子目录bin 存放OpenSSL程序，该程序可以在命令行下使用OpenSSL 功能；include 子目录存放开发所需的头文件；lib 子目录存放开发所需的静态库和共享库。值得注意的是， /usr/include/openssl/下的头文件依然是旧的，如图2-38所示。



[rootatocathost ssI]#whereis openssL

5507-6月292015 aes.h

4532**6月**29**2**90210515asansln\_lm.hac.h

300926月 292015 asnt t.h

openssl:/usr/lib64/openssl /usr/include/openssl

root@localhost ssl]#cd /usr/include/openssl/

root@localhost 总用量1580

-rw-r--r--.1 root

rw r r . 1 root

root root root root

openssl]#ll

W-r--r--

rw-r--r--

root root

图2-38

但要注意的是，/usr/lib64/下依然有libcrypto.so.1.0.1e: [root@localhost openssl-1.0.2m]#find/-name libcrypto.so.1.0.1e

**C/C++加密解密实战**

**Windows**

/usr/lib64/libcrypto.so.1.0.1e

我们开发时不需要引用这个目录下的头文件，而要引用/usr/local/sslinclude 下的头文件。为了 防止以后误用，我们可以直接删除旧的头文件，包含目录/usr/include/openssl/:

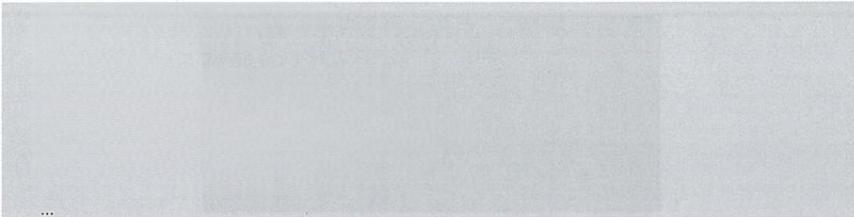
[root@localhost include]#rm -rf /usr/include/openssl 再创建新的头文件，包含目录的软链接：

In-s/usr/local/ssl/include/openssl /usr/include/openssl 下面再创建可执行文件的软链接，这样就可以在命令行下使用 openssl 命令了：



In-s/usr/local/ssl/bin/openssl/usr/bin/openssl

这样执行/usr/bin下 的OpenSSL 实际就是执行/usr/local/ssl/bin下 的OpenSSL 程序。引用 /usr/include/openssl下的头文件就是引用/usr/local/sslinclude/openssl下的头文件。不放心的话， 我们可以到/usr/include/openssl 下查看 一 下：



[root@localhost include]#cdopenssl/ [root@localhost openssl]#Ⅱ

总用量1856

-rW-r-r--.1 root root 61465月1510:14 aes.h -rW-r--I-.1 root root 631425月1510:14 asnl.h

-rW-r--r-.I root root 244355月1510:14 asnl\_mac.h

-rW-r--r--.1 root root 344755月1510:14 asnlt.h -rW-r-r--.1 root root 387425月1510:14 bio.h

终于不是2015年的了。最后添加动态库路径到动态库配置文件并更新：

echo "/usr/local/ssl/lib">>/etc/ld.so.conf ldconfig -v

至此，升级安装工作完成了。我们可以看一下现在OpenSSL的版本号：

[root@localhost bin]#openssl version openssl 1.0.2m 2 nov 2017

版本升级成功了。如果要以命令方式使用OpenSSL, 可以在终端下输入 openssl, 然 后 就

会 出 现OpenSSL 提示，如图2-39所示。



图2-39

具 体 的OpenSSL 命令我们会在后面的章节讲述，这里暂且不表。

值得注意的是，/usr/local/ssl/bin/ 下的程序OpenSSL 依赖于共享库libcrypto.so.1.0.0 。 例 如

把/usr/local/ssl/lib目录改个名字，再运行OpenSSL, 可以发现出错了：

[root@localhost libbk]#openssl

openssl:error while loading shared libraries:libssl.so.1.0.0:cannot open sharedobject file:No such file or directory



**第** **2** **章** **搭** **建C** **和C++** **密码开发环境**

这也说明，openssl 程序和/usr/lib64/libcrypto.so.1.0.1e 没什么关系。但我们依旧需要删除 /usr/lib64/libcrypto.so.1.0.1e,因为编译自己写的C/C++ 程序的时候，需要到/usr/lib64下 找libcrypto.so, 而/usr/lib64/ 下有一个libcrypto.so 是一个软链接，它测试指向的是/usr/lib64/libcrypto.so.1.0.1e 这 个 共享库，如果此时编译我们的程序，那么使用的共享库是/usr/lib64/libcrypto.so.1.0.1e, 而 不 是 新版的OpenSSL 的共享库。

为了让自己的C/C++ 程序能链接到新版OpenSSL 的共享库libcrypto.so.1.0.0,我们需要重 新做一个软链接。先删除旧的共享库/usr/lib64/libcrypto.so.1.0.1e:



rm -f/usr/lib64/libcrypto.so.1.0.1e

如果此时我们写一个C++ 程序，比如例2.11,然后在命令行下编译： g++test.cpp -o test -lcrypto

就会发现报错了：

[root@localhost ex]#g++test.cpp -o test -lcrypto /usr/bin/ld:cannot find -lcrypto

collect2:错误：ld 返回1

这说明我们把旧版共享库删除后，虽然软链接依旧存在，但还是无法编译成功。下面我们

需要把/usr/lib64/下的软链接libcrypto.so 指向新版OpenSSL 的共享库/usr/local/ssl/lib/libcrypto. so.1.0.0 。因为原来已经有软链接，需要先删除才能再创建：

[root@localhost lib64]#In -s/usr/local/ssl/lib/libcrypto.so.1.0.0/usr/lib64/libcrypto.so In: 无法创建符号链接"/usr/lib64/libcrypto.so":文件已存在

[root@localhost lib64]#rm /usr/lib64/libcrypto.so

rm: 是否删除符号链接"/usr/lib64/libcrypto.so"?y

[root@localhost lib64]#In -s/usr/local/ssl/lib/libcrypto.so.1.0.0/usr/lib64/libcrypto.so

**此时如果编译我们的程序，会发现可以编译，但运行报错：**

[root@localhost ex]#g++test.cpp -o test -Icrypto

[root@localhost ex]#./test

./test:error while loading shared libraries:libcrypto.so.1.0.0:cannot open shared object file:No such file or directory

我们需要把libcrypto.so.1.0.0 复制一份到/usr/lib64/。

[root@localhost ex]#cp/usr/local/ssl/lib/libcrypto.so.1.0.0/usr/lib64 此时如果运行 test, 会发现可以运行了：

[root@localhost ex]#./test Hello,OpenSSL!

有朋友说了，既然/usr/lib64 下 有libcrypto.so.1.0.0 了，那么是否可以让符号链接libcrypto.so 指向同目录下的 libcrypto.so.1.0.0? 这样完全可以，而且做法和原来旧版本的情况是一样的， 旧版本时的libcrypto.so 就是指向同目录下的libcrypto.so.1.0.1e 。下面先删除符号链接，再新建：

[root@localhost ex]#cd /usr/lib64



**Windows C/C++加密解密实战**

[root@localhost lib64]#rm -f libcrypto.so

[root@localhost lib64]#In-s libcrypto.so.1.0.0 libcrypto.so

此时，我们编译t**est.cpp,** 然后运行：

[root@localhost ex]#g++test.cpp -o test -lcrypto

[root@localhostex]#./test Hello,OpenSSL!

一气呵成!而且此时链接的动态库是新的OpenSSL 的动态库，不信可以用ldd命令查看 一下：

[root@localhost ex]#lddtest

linux-vdso.so.1=>(0x00007ffdaa7b8000)

libcrypto.so.1.0.0=>/lib64/libcrypto.so.1.0.0(0x00007f3862796000) libstdc++.so.6=>/lib64/libstdc++.so.6(0x00007f386248e000)

libm.so.6=>/lib64/libm.so.6(0x00007f386218b000)

libgcc\_s.so.1=>/lib64/libgcc\_s.so.1(0x00007f3861f75000) libc.so.6=>/lib64/libc.so.6(0x000073861bb4000)

libdl.so.2=>/lib64/libdl.so.2(0x00007f38619af000) libz.so.1=>/lib64/libz.so.1 (0x00007f3861799000) /lib64/ld-linux-x86-64.so.2(0x00007f3862c0d000)

我们可以看到粗体部分就是新的共享库。

是不是感觉有点麻烦?升级就是这样的，不彻底把旧的删除，那以后用了许久，说不定使 用的还是旧版的共享库。

顺便说一句，如果不想复制共享库也可以，只要在/usr/lib64下做一个符号链接，指向/usr/ local/ssl/lib/ibcrypto.so.1.0.0, 比如：

In -s/usr/local/ssl/lib/libcrypto.so.1.0.0/usr/lib64/

这样也可以运行test。反正一句话，/usr/lib64 下要有libcrypto.so和 libcrypto.so.1.0.0, 无 论是符号链接还是真正的共享库。

之所以讲这些，就是为了让大家知道运行下面的例子时背后的故事，别编译运行了半天， 链接的还是旧版的共享库。下面我们详细说明自己的OpenSSL程序的建立过程。

【例2.11】第一个OpenSSL的C++程序

( 1 ) 在Windows下打开UltraEdit或其他编辑软件，输入代码如下：

#include<iostream> using namespace std

#include "opensslevp.h"//包含相关Openssl 头文件，实际位于/usr/local/ssl/include/openssl/evp.h int main(int argc,char \*argy[])

{

char sz[]="Hello,openssl!"; cout<<sz<<endl;

openssl\_add\_all algorithms();//载入所有SSL 算法，这个函数是OpenSSL 库中的函数

return 0;

代码很简单，就调用了 一 个OpenSSL 的 库 函 数openssl\_add\_all\_algorithms, 该 函 数 的 作 用



**第** **2** **章** **搭** **建C** **和C++** **密码开发环境**

是载入所有SSL 算法，我们这里调用就是看看能否调用得起来。

evp.h的路径是/usr/localsslinclude/openssl/evp.h,它包含常用密码算法的声明。

(2)保存为 test.cpp, 上传到Linux, 在命令行下编译运行：



[root@localhost test]#g++test.cpp -o test -lcrypto

[root@localhost test]#/test Hello,OpenSSL!

运行成功了。编译的时候要注意链接 OpenSSL 的动态库 crypto,这个库文件位于 /usr/lib64/libcrypto.so,是一个符号链接，我们前面让它指向了/usr/local/ssl/lib 下的共享库 /usr/local/ssl/lib/libcrypto.so.1.0.0。

有读者或许会问，evp.h 的存放路径是/usr/localssl/include/openss/evp.h, 编译的时候为何 不用-I 包含头文件的路径呢?答案是双引号包含头文件时，如果当前工作目录没有找到所需的 头文件，就到-I所包含的路径下去找；如果编译时没有用-I指定包含目录，就去/usr/local/include 下找；如果/usr/local/include 下也没有，再到/usr/include 下去找，再找不到就报错了。而 /usr/include 下是有 OpenSSL 的，因为前面我们做了软链接，软链接指向的实际目录是 /usr/localssl/include/openss/, 因此我们使用的evp.h 就是/usr/local/sslinclude/openssl/evp.h。

**3.在指定安装目录安装OpenSSL**

前面因为要讲不少原理，所以比较啰唆，这里将进行简化，直接用步骤阐述。

(1)卸载旧版OpenSSL

这一步前面的章节已经讲过，这里不再赘述。

(2)解压和编译

把下载下来的压缩文件放到Linux 中，这里存放的路径是/root/soft, 大家可以自定义路径， 然后进入这个路径后解压缩：

[root@localhost~]#cd /root/soft

[root@localhost soft]#tar zxf openssl-1.0.2m.tar.gz

进入解压后的文件夹，开始配置、编译和安装：

[root@localhost soft]#cd openssl-1.0.2m/

[root@localhost openssl-1.0.2m]#./config --prefix=/usr/local/opensslshared

其中，--prefix 表示安装到指定的目录中，这里的指定目录是/usr/local/openssl, 这个目录 不必手工预先建立，安装 (make install) 的过程会自动新建；shared 表示除了生成静态库外， 还要生成共享库，如果仅仅想生成静态库，可以不用这个选项，或者用no-shared。

下面开始编译：

[root@localhost openssl-1.0.2m]#make

此时，如果到/usr/local 下查看，发现并没有 openssl 文件夹，这说明还没建立。而且 /usr/include/openssl下的头文件依旧是老版本OpenSSL 遗留下来的。



**Windows** **C/C++加密解密实战**

( 3 ) 安 装OpenSSL

[root@localhost openssl-1.0.2m]#make install

细心的朋友可以看到，安装过程中有如图2-40所示的这几步。



make1]: **对al”无需做任何事。**

make[i: **高开目录7root/soft/**openssl-1.0.2m/tools"

created directory /usr/local/openssl'

created directory /usr/Local/openssl/ssl

created directory /usr/local/openssl/ssl/man

ddit /lel3'

created directory \*/usr/local/openssl/ssl/man/man⁵'

created directory /usr/local/openssl/ssl/man/man7'

installing manl/asnl parse.1

openssl-asni parse.1 →asnt parse.1

lnggl// .1pl.1

图2-40

created directory 表示目录创建完成，所以/usr/local/openssl建立了。

稍等片刻，安装完成。此时如果到/usr/local下查看，发现有 openssl文件夹了，而且在该 目录下可以看到其子文件夹，如图2-41所示。

root@Localhost openssl-1.0.2同#cd /usr/local/opensst

root@localhost openssl]#ls in include lib ssl

图2-41

其 中 ，bin 里面存放OpenSSL 命令程序，include 存放开发所需要的头文件，lib 存放静态

库文件，ssl 存放配置文件等。

(4)更新头文件包含的目录和命令程序

删除旧的头文件包含的目录/usr/include/openssl/:

[root@localhost include]#rm -rf/usr/include/openssl/openssl 再创建新的头文件包含目录的软链接：

In -s/usr/local/openssl/include/openssl /usr/include/openssl

下面再创建可执行文件的软链接，这样就可以在命令行下使用openssl 命令： In -s /usr/local/openssl/bin/openssl /usr/bin/openssl

这样执行/usr/bin下 的openssl 实际就是执行/usr/local/opensslbin 下 的openssl 程序。引用 /usr/include/openss1下的头文件就是引用/usr/local/openss/includelopenss下的头文件。此时，我 们可以在任意目录下运行openssl命令。我们可以看一下现在openssl的版本号：



[root@localhost bin]#opensslversion openssl 1.0.2m 2 nov 2017

如果要以命令方式使用OpenSSL, 可以在终端下输入 opensl, 然后就会出现OpenSSL 提 示，如图2-42所示。



第 2 章 搭 建C 和 C++ 密码开发环境



**root@localhost openssl]#openssl** D**penSSL>**

图2-42

(5)更新共享库

删除旧的共享库/usr/lib64/libcrypto.so.1.0.1e: m -f/usr/lib64/libcrypto.so.1.0.1e

我们需要把libcrypto.so.1.0.0 复制一份到/usr/lib64/:

[root@localhost ex]#cp/usr/local/openssl/lib/libcrypto.so.1.0.0/usr/lib64

(6)更新符号链接

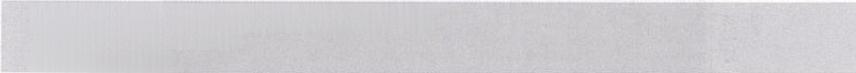
删除旧的符号链接才能再创建新的：

[root@localhost lib64]#rm -f/usr/lib64/libcrypto.so

[root@localhost lib64]#In-s /usr/local/openssl/lib/libcrypto.so.1.0.0/usr/lib64/libcrypto.so

(7)验证

我们对上例的test.cpp 进行编译，然后运行：



[root@localhost ex]#g++test.cpp -o test -lcrypto

[root@localhost ex]#./test Hello,OpenSSL!



一气呵成!而且此时连接的动态库是新的 OpenSSL的动态库，不信可以用ldd 命令查看 一下：

[root@localhost ex]#ldd test

linux-vdso.so.1=>(0x00007ffdaa7b8000)

**libcrypto.s<0.1.0.0>=>/lib64/libcrypto.so.1.0.0(0x00007f3862796000)**

libstdc++.so.6=>/lib64/libstdc++.so.6(0x00007f386248e000) libm.so.6=>/lib64/libm.so.6(0x00007f386218b000)

libgcc\_s.so.1=>/lib64/libgcc\_s.so.1(0x000073861f75000) libc.so.6=>/lib64/libc.so.6(0x000073861bb4000)

libdl.so.2=>/lib64/libdl.so.2(0x00007f38619af000) libz.so.1 =>/lib64/libz.so.1(0x00007f3861799000) /lib64/d-linux-x86-64.so.2(0x00007f3862c0d000)

我们可以看到粗体部分就是新的共享库。

**2.2.14 测试使用openssl命令**

OpenSSL 的命令行程序为 openssl.exe 。 本节的命令用32位的1. 1. 1b 版 本 的openssl.exe 来 阐述。其他版本的 openssl.exe 的用法类似。openssl 命令程序位于apps 目录下，编译这些源码 最终会生成 一个可执行程序，在 Linux 下 为 opessl, 在 Windows 下 为 openssl.exe, 生 成 的

openssl.exe 位 于D:\openssl-1.1.1bwin32-debug\bin 。 用户可运行 openssl 命令来进行各种操作。

打开操作系统的命令行窗口，然后进入D:\openssl-1.1.1blwin32-debuglbin\, 输 入openssl.exe,



**Windows C/C++加密解密实战**

按回车键运行。虽然也可以在Windows 资源管理器中双击openssl.exe,但此时出现的OpenSSL

命令行窗口中居然不能粘贴，这对于懒惰的“码农”来说是不可接受的。但很幸运，可以从操 作系统的命令行窗口中启动openssl.exe。

**1.查看版本号**

在 OpenSSL 命令行提示符后输入version 可以查看版本号，如图2-43所示。



D:\openss1-1.1.1b\rin32-debnz

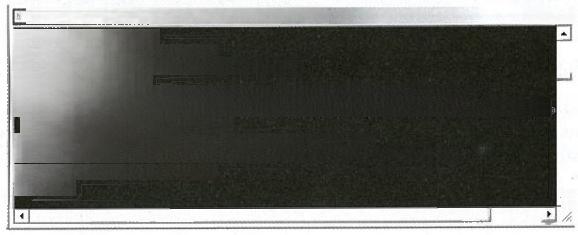
OpenSSL>version

openSSL 1.1.1b 26 Feh 2019 OpenSSL>

×

图2-43

这是我们学到的OpenSSL 的第一个命令。如果要查看详细的版本信息，可以加-a, 如 图2-44所示。



D\openss1-1.1.1\vin32-dcbog\inlopens.ex - 口×

OpenSSL>version -a

opensSL 1.1.1h 26 Feb 2619

huilt on:Wed Apr 1704:56:132819 UIC

platforn:UC-WIH32

optiong:bm(64.32>re4(int)des(long)idea(int)blovf ish(ptr)

onpiler:c1 Zi Fdosa1\_otatic.pah HT z1 GsO GF Cy M3 Aad4098 mo

Ax -DLENDIAN -DOPENSSsL PIC

OPEHSSLDIR:"D=\openss1-1.1.1bwin32-dobug :1" ENGIHESDIR:"D:\openssl-1.1.1bwin32-dabug\libunginen-1\_ 1"

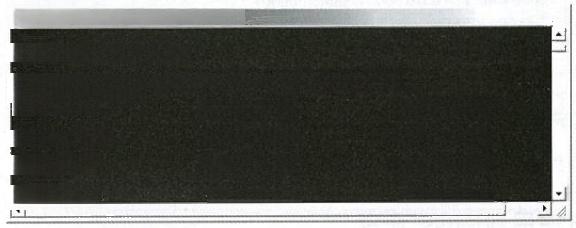
Seeding source:o8-apecific OpensSSL>

v

图2-44

**2.查看支持的加解密算法**

定 位 到bin 文件夹路径，然后输入命令：openssl enc-ciphers,如图2- 45所示。



回管理员Cadr\yta32\a

e Wsere Adninistrator>d:

D:\openss1-1.1.1bwin32-debug sbin>openssl

Supported ciphera:

Faes-128-che -aes-128-cfh

-aes- 128-cfh⁸ -aes-128-ctr

Faes-128-ofh -aes-192-cbe

aea-192-cfh -aes-192-cfh8

×

ene -ciphers

-aes-128-cfh1 -aes-128-ech

-aes-192-cfb

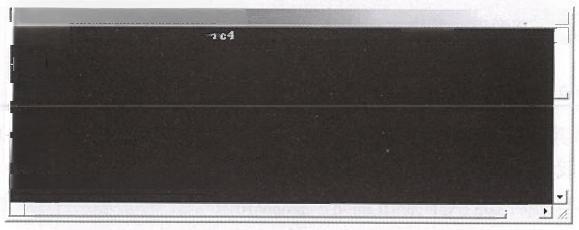
-aes-192-ctr

图2-45

支持好多算法，最激动的是支持我们国产算法了，比如SM4 。我们可以往下拖曳滚动条， 可以看到SM4 了，如图2-46所示。



**第** **2** **章** **搭** **建** **C** **和** **C++** **密码开发环境**



应管理员 CWindors\systs32\cd re2-ofh

-seed

-eeed-ech

en4-cbe

an⁴-ucb

D=o penas1-1.1.1bwin32-de bug bin>

4

-rc4-40

-seed-cfh

sn4

-sn4-ctr

-seed-che -seed-ofb -an4-cfh

-an4-ofh

- 口×



o

图2-46

**3.查看某个命令的帮助信息**

查看某个命令的帮助信息使用命令-help。比如我们要查看 version命令的帮助信息，如 图2-47所示。

车 尸 也 子 口 户 1



D:\openssl-1.1.1b\vin32-dabat\b

OpenSSL>version -help

Usage:version [options]

Ualid options are:

-help Display this sunnary

Show all data

Show build date

Show configuration directory Show engines directory

Show conpiler flags used

Show sone internal datatype options

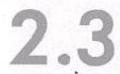
Show target build platforn Shouvandon saeding options Show library version

OpenSSL>

口 ×

图2-47

通过几个简单命令的使用，我们知道安装成功了。



**纯C++ 密码开发Crypto++ 库**

每种强大的语言都有相应的密码安全方面的库，比如Java 自带了加解密库。那么C++ 有 没有这样的库呢?答案是肯定的，那就是Crypto++。

Crypto++是一个C++ 编写的密码学类库。读过《过河卒》的朋友还记得作者的那个不愿意 去微软工作的儿子吗?就是Crypto++ 的作者WeiDai 。Crypto++是一个非常强大的密码学库， 在密码学界也很受欢迎。虽然网络上有很多密码学相关的代码和库，但是Crypto++ 有其明显 的优点。主要是功能全、统一性好，例如椭圆曲线加密算法和 AES 在 OpenSSL 的 Crypto 库 中就还没最终完成，而在Crypto++中就支持得比较好。

基本上密码学中需要的主要功能都可以在里面找得到。Crypto++ 是由标准的C++ 写成的， 学 习C++、密码学、网络安全都可以通过阅读Crypto++ 的源代码得到启发和提高。



**Windows C/C++加密解密实战**

Crypto++是一个开源库，其官方网站是 www.cryptopp.com。

**2.3.1 Crypto++的编译**

我们可以从其官网上下载最新源码，这里下载下来的文件名是 cryptopp610.zip,是一个 ZIP 压缩文件，我们可以把它放到Linux 下解压缩：

[root@localhost soft]#unzip cryptopp610.zip -d cryptopp610

加-d是解压到目录cryptopp610下，这个目录会自动建立。

解压完毕后，进入目录cryptopp610,然 后 用make 进行编译：

[root@localhost sofl]#cd cryptocpp610/ [root@localhost cryptocpp610]#make

稍等片刻，编译完成，此时会在文件夹cryptocpp610下生成一个静态库libcryptopp.a。有 了这个静态库，我们就可以在应用程序中使用Crypto++提供的加解密函数了。

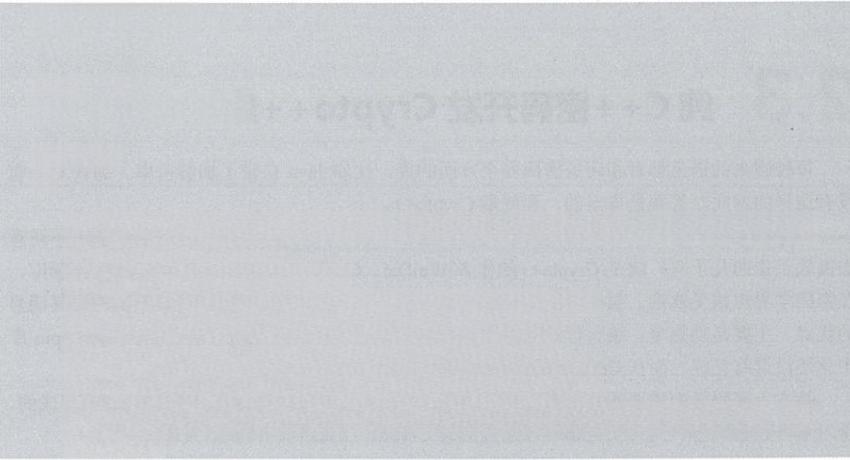
**2.3.2使用Cypto++进行AES加解密**

前面我们通过Crypto++源码编译出来了一个静态库libcryptopp.a, 现在开始使用它。

首先看 一 个例子，这个例子是直接用AES 加密 一 个块，AES 的 数 据 块 ( 分 组 ) 大 小 为 1 2 8 位，密钥长度可选择128位、192位或256位。直接用AES 加密一个块很少用，因为我们平 时都是加密任意长度的数据，需要选择CFB 等加密模式。但是直接的块加密是对称加密的基 础 。

【例2 . 12】 一个使用Crypto++ 库的例子

( 1 ) 在Windows 下打开UE (或其他编辑器),然后输入代码如下：



#include <iostream> using namespace std;

#include <aes.h>

using namespace CryptoPP;

int main() {

1/AES中使用的固定参数是以类AES 中定义的Enum 数据类型出现的，而不是成员函数或变量 //因此需要用：:符号来索引

cout<<"AES Parameters:"<<endl;

cout<<"Algorithmname:"<<AES::StaticAlgorithmName()<<endl;

//Crypto++ 库中一般用字节数来表示长度，而不是常用的字节数

cout<<"Block size :"<<AES::BLOCKSIZE\*8<<endl;

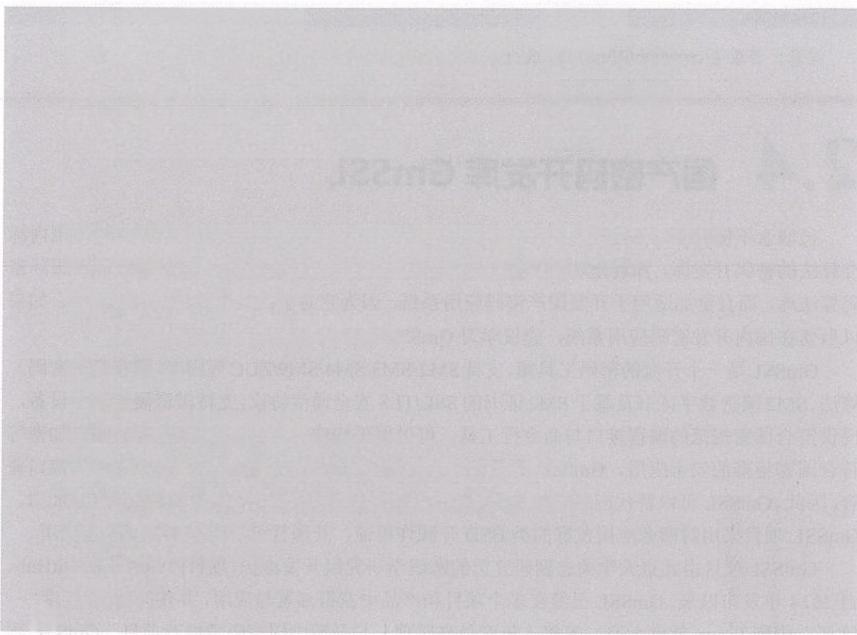
cout <<"Min key length:"<<AES:MIN\_KEYLENGTH\*8<<endl;

cout<<"Max key length:"<<AES:MAX\_KEYLENGTH\*8<<endl;

//AES 中只包含一些固定的数据，而加密解密功能由 AESEncryption 和 AESDecryption 来完成 //加密过程



**第** **2** **章** **搭** **建C** **和** **C++** **密码开发环境**



AESEncryption aesEncryptor;

unsigned char aesKey[AES:DEFAULT\_KEYLENGTH];

unsigned char inBlock[AES:BLOCKSIZE]="123456789";

unsigned char outBlock[AES::BLOCKSIZE]; unsigned char xorBlock[AES::BLOCKSIZE]; memset(xorBlock,0,AES::BLOCKSIZE);

aesEncryptor.SetKey(aesKey,AES::DEFAULT\_KEYLENGTH); aesEncryptor.ProcessAndXorBlock(inBlock,xorBlock,outBlock) //以16进制显示加密后的数据

for(int i=0;i<16;i++){

cout<<hex<<(int)outBlock[i]<<"";

}

cout<<endl; //解密

AESDecryption aesDecryptor

unsigned char plainText[AES::BLOCKSIZE]

aesDecryptor.SetKey(aesKey,AES::DEFAULT\_KEYLENGTH); aesDecryptor.ProcessAndXorBlock(outBlock,xorBlock,plainText);

for(int i=0;i<16;i++) cout<<plainText[i];

cout<<endl; return 0;



//加密器 //密钥

//要加密的数据块 //加密后的密文块 //必须设定为全零 //置零

//设定加密密钥 //加密

代码中有几个地方需要注意一下：

AES 并不是一个类，而是类Rijndael的一个typedef。

Rijndael虽然是一个类，但是其用法和Namespace很像，本身没有什么成员函数和成员变 量，只是在类体里面定义了一系列的类和数据类型，真正能够进行加密解密的AESEncryption 和 AESDecryption都是定义在这个类内部的类。

AESEncryption和 AESDecryption除了可以用SetKey()这个函数设置密钥外，在构造函数 中也能设置密钥，参数和SetKey()是一样的。

ProcessAndXorBlock()可能会让人比较疑惑，函数名的意思是ProcessBlock 和 XorBlock, ProcessBlock 就是对块进行加密或解密，XorBlock 在各种加密模式中使用，这里我们不需要使 用加密模式，因此把用来Xor 操作的XorBlock 设置为0,那么 Xor 操作就不起作用了。

(2)保存代码为test.cpp,上传到Linux,在命令行下编译并运行：

[root@localhost test]#g++test.cpp -o test -I/root/soft/cryptopp610-Lroot/soft/cryptopp610-leryptopp [root@localhost test]#./test

AES Parameters

Algorithm name:AES Block size :128 Min key length:128 Max key length:256

776e 2c a52177a 5b19e4286526 f37e 14



Windows C/C++加密解密实战

123456789

注意：目录名cryptopp610不要写成 cryptoapp610。

**2.4 国** **产** **密** **码** **开** **发** **库GmSSL**

长城永不倒，国货当自强。随着我国科技的发展，现在我们自己也拥有了包含多种国内标 准算法的密码开发库，那就是功能强大的GmSSL。作为后起之秀，GmSSL 丝毫不逊于国际密 码算法库，而且更加适用于开发国产密码应用系统，因为它对于国密算法的支持更完善。如果 以后要在国内开发密码应用系统，建议学习GmSSL。

GmSSL 是一个开源的密码工具箱，支持SM2/SM3/SM4/SM9/ZUC 等国密(国家商用密码) 算法、SM2国密数字证书及基于SM2证书的SSL/TLS安全通信协议，支持国密硬件密码设备， 提供符合国密规范的编程接口与命令行工具，可以用于构建PKI/CA、安全通信、数据加密等 符合国密标准的安全应用。GmSSL 项目是OpenSSL项目的分支，并与OpenSSL 保持接口兼 容。因此，GmSSL可以替代应用中的OpenSSL 组件，并使应用自动具备基于国密的安全能力。 GmSSL项目采用对商业应用友好的类BSD 开源许可证，开源且可以用于闭源的商业应用。

GmSSL项目由北京大学关志副研究员的密码学研究组开发维护，项目源码托管于GitHub。 自2014年发布以来，GmSSL已经在多个项目和产品中获得部署与应用，并获得2015年度“一 铭杯”中国Linux 软件大赛二等奖(年度最高奖项)与开源中国密码类推荐项目。GmSSL 项 目的核心目标是通过开源的密码技术推动国内网络空间的安全建设。

**2.4.1 GmSSL的特点**

GmSSL作为我国自主研发的密码算法库，在功能和性能上有着自己的特点：

(1)支持SM2/SM3/SM4/SM9/ZUC等已公开的国密算法。

(2)支持国密SM2 双证书SSL 套件和国密SM9 标识密码套件。

(3)高效实现，在主流处理器上可完成4.5万次 SM2签名。

(4)支持动态接入具备 SKF/SDF 接口的硬件密码模块(SKF 是 USBKEY/TF 卡的应用接 口规范，SDF 是密码设备的应用接口规范)。

(5)支持门限签名、秘密共享和白盒密码等高级安全特性。

(6)支持Java 、Go 、PHP等多语言接口绑定和REST 服务接口。

**2.4.2 GmSSL的一些历史**

2018年12月18日，GmSSL 已部署Travis和 AppVeyor持续集成工具，用以测试Linux 和 Windows环境下的编译和安装。

2018年10月13日，GmSSL-2.4.0 发布，支持国密256位 Barreto-Naehrig 曲线参数



**第** **2** **章** **搭** **建C** **和C++** **密码开发环境**

(sm9bn256vl) 上的SM9算法。

2018年6月27日，密码行业标准化技术委员会公布了所有密码行业的标准文本。

2018年5月27日，GmSSL 增加SM4 算法的Bitslice实现。

2018年3月21日，IESG工作组批准TLS1.3 协议作为建议标准。

2018年3月13日，增加 GmSSL PHP语言API。

2017年11月11日，中国可信云计算社区暨中国开源云联盟安全论坛在北京大学举办。 2017年5月15日，发布GmSSL-1.3.0二进制包下载(5.4MB)。

2017年4月30日，增加GmSSL Go语言API。

2017年3月2日，GmSSL 项目注册了OID {iso(1)identified-organization(3)dod(6)internet(1) private(4)enterprise(1)GmSSL(49549)}。

2017年2月12日，支持完整的密码库Java语言封装 GmSSL-Java-Wrapper。 2017年1月18日，更新了项目主页。

**2.4.3** **什么是国密算法**

GmSSL 最大的特点是对国密算法的强大支持，可以说，它就是为国密算法而生的。那什 么是国密算法呢?国密算法是国家商用密码算法的简称。自2012年以来，国家密码管理局以 《中华人民共和国密码行业标准》的方式陆续公布了SM2/SM3/SM4等密码算法标准及其应用 规范。其中，SM 代表“商密”,即商用的、不涉及国家秘密的密码技术。SM2 为基于椭圆曲 线密码的公钥密码算法标准，包含数字签名、密钥交换和公钥加密，用于替换 RSA/Diffie-Hellman/ECDSA/ECDH 等 国 际 算 法 ；SM3 为密码哈希算法，用于替代 MD5/SHA-1/SHA-256等国际算法；SM4 为分组密码算法，用于替代DES/AES等国际算法； SM9为基于身份的密码算法，可以替代基于数字证书的PKI/CA体系。通过部署国密算法，可 以降低由弱密码和错误实现带来的安全风险和部署PKI/CA带来的开销。

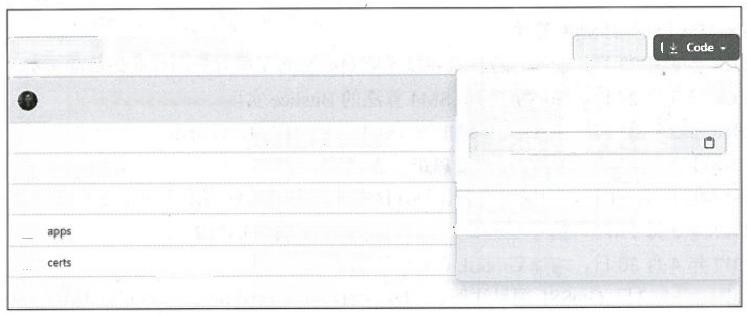
由于密码在国民经济中的敏感性，因此涉密的系统采用国密算法是大势所趋。学习和使用 国密算法也是每个密码行业开发者的基本功。

**2.4.4** **GmSSL的** **下** **载**

我们可以到GitHub网站上下载源码，网址是<https://github.com/guanzhi/GmSSL>。打开网页 后，单击右方的Code 下拉按钮，然后在下拉框中单击 Download ZIP按钮，就可以下载了， 如图2-48所示。



**Windows C/C++加密解密实战**



P master P 2 branches 1 tag **Gotofle**

davidkhala Updatestale.yml(P1020)

**HTTPS** GtHub Cu

github

Confgurations

VMS

**Open withGitHub Desktop**

Add GM double cert supportins\_cient Revert"quantum init

crypto

Update stale.yml(\*1020) Fx Windowsbuild error Revert "quantum init"

https://github,com/guanzhl/G=SSL.git Use Gt or checkout with SVN using the web URL

ZUC256 bug fix

5 months ago

D Download ZIP



①

**Clone**

图2-48

下载下来后是一个ZIP 文件，文件名是GmSSL-master.zip。当前下载的新版本是2.5.4。

**2.4.5** **在** **Windows** **下编译安装GmSSL**

我们把下载下来的GmSSL-master.zip放 到D 盘(也可以放到其他盘)并解压。

在Windows 下编译GmSSL需要先安装ActivePerl 和 Visual Studio 2017,相信读者编译 OpenSSL 的时候已经安装过这两个工具了，这里不再赘述。然后以管理员身份打开Visual Studio Tools下的“VS 2017的开发人员命令提示符”,并定位到D:GmSSL-master, 接着运行： perl Configure VC-WIN32,运行后如图2-49所示。

然后输入编译命令：nmake, 稍等片刻，编译完毕，如图2-50所示。



丽管理员：VS 2017的开发人员命令 回 ×

.nanifest -outputresource:test zuctest. 

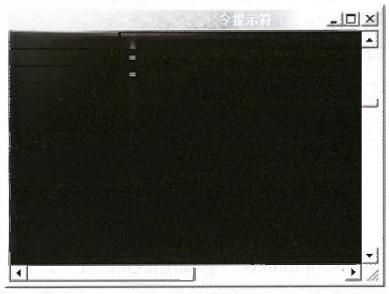
"G:NPer164bin \perl.exe""-I." e""apps \CA.pl.in">"apps CA.pl"

"C:\Per164\bin \perl.exe""-1." e""apps tsget.in">"appe \tsget.pl"

"C:\Per164 whin \perl.exe""-1."

e""toolsN\_rehash.in">"tools c\_rehash

**D:\GnSSL-naster>**



回管理员：VS 2017的开发人员命

POLV1305\_0BJ -poly1385-x86.0 BLAKE2\_0BJ

PROCESSOR RANLIB

-/ologo

-C:\Per164

mode

Configured for UC-WIH32.

D:VGnSSL-master>。

ARFLAGS

PERL

THIRIY\_IWO\_BII

BN\_LLOHG mode

bin\perl.exe

-true

图2-49 图2-50

接着输入安装命令： nmake install,再次稍等片刻，安装完毕，如图2 - 51所示。

安装成功，可在C:\Program Files(x86)\GmSSL下找到bin 、html 、include 、lib等文件夹， 如图2-52所示。



国管理员：Vs 2017的开发人员命令提示 -I 口 ×

C:VProgram Piles(x86)\GnSSLNhtnIVhan3ZUC\_

C:Program Files(x86)\GnSSL\html vman3ZUC\_g Files(x86>\GnSSL\htmlvnan3ZUC\_set\_key.htn

C:VProgram Files (x86)GmsSLNhtn1wan3 zUC\_g les(x86)\GnSSLShtml man32UC\_set\_key.htnl

**D=\GnSSL-master>**



图2-51

**第** **2** **章** **搭** **建C** **和** **C++** **密码开发环境**



**·本地磁盘**(C:)-Program Piles (k86)▼GmSSL▼

**共享▼** 刻录新建文件夹

名称 ·

bin

html

include

1ib

修改日期

2019/12/19 2019/12/19 2019/12/19 2019/12/19

图2-52

其中，目录 bin 存放GmSSL 命令行工具程序，目录html存放一些帮助文件，目录 include 存放开发所需要的头文件，目录lib存放开发所需的库文件，这些都和OpenSSL 类似。

下面是一些常见的编译错误及原因。

(1)安装的Visual C++版本较低，比如使用Visual C++6 、Visual Studio 2008。

(2)源码不干净或并非最新，建议从一份干净的(没有已经编译出来的对象文件或汇编 文件)最新的Master分支源代码开始编译。

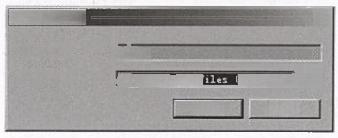
(3)编译系统没有找到nmake。实际上nmake是 Visual Studio自带的工具，不需要单独 安装。编译系统无法找到nmake 的原因是没有在Visual Studio的命令行环境下执行编译指令。

(4)无法执行 nmake install。这个命令需要以管理员身份执行。

(5)对象文件(.obj) 和目标平台不一致，通常是由于在Visual Studio的32位控制台下 执行Perl Configure VC-WIN64A,或者在Visual Studio的64位控制台下执行Perl Configure VC-WIN32 导致的。

**1.验证命令行工具**

为了方便在命令行下使用gmssl 命令，可以把命令程序所在的路径加入系统的Path 变量 中，在桌面上对“计算机”右击，选择“属性”→“高级系统设置”→“高级”→“环境变量”, 选中系统变量下的 Path, 然后单击“编辑”按钮，并在其末尾加入路径“C:\Program Files (x86)\GmSSL\bin”, 注意前面要用分号隔开，如图2-53所示。



冈

**GNCSSL**

取消

Fath

**4\CAFrorm**

确定

**编辑系统变里**

变里名0: 变里值V:

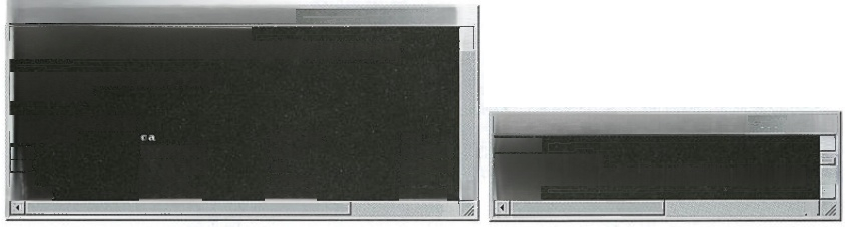
图2-53

单击“确定”按钮关闭对话框。然后重新打开一个新的命令行窗口，并输入命令 gmssl, 可以发现出现提示符>了，此时可以输入gmssl的子命令，比如help, 如图2-54所示。

也可以输入 version 查看版本号，如图2-55所示。



**Windows C/C++加密解密实战**

国管理员：CNhdssst32\ad

回 凶 互

**Hicrosoft** **Windous** **r版本6.1.76811**

**版权所有(c>2H⁰9** **Microaoft** **Corporation。保留所有权利**。

**C:Wsers MAdninistrator>gnss1 GnSSL>help**

凶

回

回管理员：C Tidos\sy:ten32Acd.exe

GnSSL>

GnSSL>version

GnS8L 2.5.4-OpenS⁸L 1.1.8d 3 Sep

GnSSL>

**Standardconnands**

asniparse

crl daa

enc

send

crl2pkcs? dsaparan engine

genpkay

ciphers dgst

cns

dhparan ecparan exit

qansn9

ec

2019

erratr genria



图2-54 图2-55

如果要退出GmSSL 命令行，可以输入 quit。

下面我们用SM3 命令来计算一个Hash 值。首先在D 盘下新建一个文本文件，文件名是 my.txt, 并 输 入 3 个 字 符abc, 然后保存。接着在命令行下输入命令：sm3 D:\my.txt,运算结果 如下：

GmSSL>sm3 d:\my.txt

SM3(d:\my.txt)=66c7f0f462eeedd9d1f2d46bdc10e4e24167c4875cf2f7a2297da02b8f4ba8e0

其 实 ，GmSSL 的命令行操作和OpenSSL 大同小异。至此，命令行验证正确。下面开始程 序验证。

**2** **.程序验证GmSSL**

GmSSL 提供了 EVP (Envelop的简称)系列API 供开发者使用。EVP API是 GmSSL 的 密码服务接口，它屏蔽了具体算法的细节，为上层应用提供统一、抽象的接口。该接口的头文 件 为openssl/evp.h, 所对应的函数库为libcrypto。

【例2.13】用代码验证GmSSL

( 1 ) 打 开VC 2017,新建一个控制面板工程，工程名是test。 ( 2 ) 在test.cpp 中输入代码如下：

#include "pch.h"

#include <openss/conf.h>

#include <openssl/evp.h>

#include <openssl/err.h>

int main(int arc,char \*argv[])

{

/\*Load the human readable error strings for libcrypto \*/

ERR\_load\_crypto\_strings()

/\*Load all digest and cipher algorithms \*/

OpenSSL\_add\_all\_algorithms();

/\*Load config file,and other important initialisation\*/

OPENSSL\_config(NULL);



**第2章** **搭建C** **和C++** **密码开发环境**

/\*...Do some cryptostuff here...\* printf("call GmSSL lib ok\n");

/\*Cleanup\*/

/\*Removes all digests and ciphers \*/ EVP\_cleanup()

/\*if you omit the next,a small leak maybe left when youmake use of the BIO(low level API)for e.g. base64 transformations\*/

CRYPTO\_cleanup\_all\_ex\_data();

/\*Remove error strings\*/ ERR\_free\_strings)

return 0;

代码很简单，我们只是加载了所有密码函数并释放空间，并没有做实际的运算，但用来测 试GmSSL 库已经够用了。

然后，打开工程属性，在左边选中“C/C++”, 然后在右边的“附加包含目录”旁输入 “D:\gmssl-masterlinclude”。接着，在右边选中“连接器” → “常规”,在右边的“附加库目 录”旁输入“D:\gmssl-master”,在右边选中“输入”,在左边的“附加依赖性”旁边输入

“libcrypto.lib;”,最后单击“确定”按钮。这样头文件路径、库路径和库名称都设置好了。 D:\gmssl-master 下 的libcrypto.lib 是我们编译生成的静态库。

(3)保存工程并按Ctrl+F5 键运行，运行结果如图2-56所示。



licrosoft Vis 回冈

cal1 GnSSL lib ok



图2-56

**2.4.6** **在** **Linux** **下编译安装GmSSL**

这里使用的是CentOS 7(也可以使用其他版本的Linux) 。 以 root 账户登录Linux, 把 下 载下来的GmSSL 压缩包GmSSL-master.zip 放 到Linux 下，然后解压：

unzipGmSSL-master.zip

接着进入文件夹GmSSL-master, 开始配置：

[root@localhost soft]#cd GmSSL-master

root@localhost GmSSL-master]#./config --prefix=/usr/local/mygmssl

其中，--prefix 用来指定安装目录。也可以不用--prefix, 那么将采用默认路径，即命令程 序会安装在/usr/local/bin 下，头文件会安装到/usr/local/include 下，库文件会存放到/usr/local/lib 下。这里为了安装后简洁，我们采用--prefix,这样安装后的可执行程序、头文件和库文件分



**Windows C/C++加密解密实战**

别会放到mygmssl 目录下的bin 、include 和 lib 中。目录mygmssl 会自动建立，不需要预先手 工建好。

配置完毕后，开始漫长的编译： [root@localhost GmSSL-master]#make

这个过程有点长，读者可以去泡壶茶。编译完毕，开始安装： [root@localhost GmSSL-master]#makeinstall

这个过程也稍长，可以喝会茶。安装完毕后，我们可以进入/usr/local/mygmssl, 使 用 1s 查 看可以发现所有东西都在：

[root@localhost local]#cd mygmssl/ [root@localhost mygmsl]#ls

bin include lib share ssl root@localhost mygmssl]#

其 中 ，bin 存放命令工具程序，include 存放头文件，lib 存放库文件，这都是开发所需要 的。而/usr/local/bin、/usr/local/include和/usr/local/lib依旧为空：



[root@localhost/]#cd /usr/local/lib [root@localhost lib]#ls

[root@localhost lib]#cd../include [root@localhost include]#ls

[root@localhost include]#cd../bin [root@localhost bin]#ls

root@localhost bin]#

如果我们不指定安装目录，采用默认安装目录，这3个文件夹下都会有东西，多疑的读者 可以尝试一下。

**1.验证命令行工具**

安装完毕后，验证是否能正常工作。前面我们通过指定安装目录的方式来安装，其实这样 也有不便的地方，就是运行命令程序gmssl 的时候，要到/usr/local/mygmssl/bin 下执行。如果 默认安装，存放到/usr/locl/bin 下，那么可以在任意目录运行 gmssl 。怎么办呢?可以做 一 个 链接：

[root@localhost bin]#In -s/usr/local/mygmssl/bin/gmssl /usr/local/bin/gmssl

这样，/usr/local/bin/ 下有一个软链接gmssl 指向/usr/local/mygmssl/bin 下的程序gmssl 。此 时，可以在任意目录下执行gmssl, 操作后却失败了：

[root@localhost~]#gmssl

gmssl:error while loading shared libraries:libssl.so.1.1:cannot open shared object file:No such file or directory

gmssl 运 行 需 要 动 态 库 libssl.so.1.1, 但 是 没 找 到 。 通 过 搜 索 发 现 ， 该 库 位 于

/usr/local/mygmss/lib/ 下：

[root@localhost GmSSL-master]#find/-name libssl.so.1.1



**第** **2** **章** **搭** **建C** **和C++** **密码开发环境**

 /usr/local/mygmssl/lib/libssl.so.1.1 再查看其大小：



[root@localhost GmSSL-master]#du /usr/local/mygmssl/lib/libssl.so.1.1 548 /usr/local/mygmssl/lib/libssl.so.1

可以看到有548字节，确实是一个文件，而不是链接。du 用于查看文件大小的命令。

如何让gmssl找到libssl.so.1.1呢?我们知道，在CentOS7 下，可执行程序会自动到系统 路径(比如/usr/lib64/下)去搜索所需的库。那libssl.so.1.1放到/usr/lib64下，不就可以了。其 实不需要实际复制库文件过去，只需要做一个软链接，即在/usr/lib64/ 下新建 一个软链接，使 其指向/usr/local/mygmssl/lib/libssl.so.1.1 即可，这样可以节省磁盘空间。在命令行下输入：

[root@localhost GmSSL-master]#In -s/usr/local/mygmssl/lib/libssl.so.1.1 /usr/lib64/libssl.so.1.1 再次运行 gmssl:

[root@localhost GmSSL-master]#gmssl

gmssl:error while loading shared libraries:libcrypto.so.1.1:cannot open shared object file:No such file or directory

可以发现错误提示变了，找不到另一个共享库libcrypto.so.1.1 了，这说明 libssl.so.1.1 找

到了。我们继续搜索 libcrypto.so.1.1, 凭经验应该也在/usr/local/mygmssl/lib/ 下。果然：

[root@localhost GmSSL-master]#cd /usr/local/mygmssl/lib/ [root@localhost lib]#ls

engines-1.1 libcrypto.a libcrypto.so libcrypto.so.1.1 libssl.a libssl.so libssl.so.1.1 pkgconfig

找到就好办了，继续在/usr/lib64/ 下 建立软链接指向/usr/local/mygmssl/lib/libcrypto.so.1.1:

[root@localhost lib]#In -s/usr/local/mygmssl/lib/libcrypto.so.1.1 /usr/lib64/libcrypto.so.1.1

再次运行 gmssl, 发现出现提示符了，说明终于成功了：

[root@localhost lib]#gmssl GmSSL>

我们可以输入 version 和 help 命令来测试一 下：

GmSSL>version

GmSSL 2.5.4-OpenSSL 1.1.0d 3 Sep 2019 GmSSL>help

|  |  |  |
| --- | --- | --- |
| Standard commands |  |  |
| asnlparse ca  crl crl2pkcs7 dsa dsaparam | ciphers dgst  ec | cms  dhparam ecparam |
|  |  |  |

发现都成功了。下面准备运算abc 的 SM3 哈希值，我们在/root 下 用vi 新建 一 个文件my.txt, 输 入abc 三个字符，然后保存。再回到GmSSL> 下，再输入 一个SM3 运算命令：

GmSSL>sm3 /root/my.txt



**Windows C/C++加密解密实战**

SM3(/root/my.txt)=12d4e804elfcfdc181ed383aa07ba76cc69d8aedcbb7742d6e28ff4fb7776c34

可以发现，正确输出SM3 哈希值了。细心的读者可能会发现，怎么同样是内容为abc的 文本文件，结果却和Windows 下的不同?这是因为，所有的linux 会自动加上一个文件结束符 0a ( 即LF), 这样导致my.txt的实际内容(16进制)是6162630a, 我们可以用xxd命令查看 一下：

[root@localhost ~]#xxd my.txt

0000000:6162630a abc

xxd 命令以16进制显示文件内容。因此，这里的SM3 其实是对4个字符进行SM3 运 算 ， 而Windows下的SM3 是对3个字符进行运算，结果自然不同了。

此外，也可以不在GmSSL 提示符下测试 SM3, 可以在普通Linux 命令行下直接测试sm3:

[root@localhost test]#echo -n "abc"|gmssl sm3

(stdin)=66c7f0f462eeedd9d1f2d46bdc¹0e4e24167c4875cf2f7a2297da02b8f4ba8e0

至 此 ，SM3 命令测试成功。下面进入代码程序测试验证阶段。

**2.程序验证GmSSL**

GmSSL提供了EVP(Envelop 的简称)系列API 供开发者使用。EVP API是GmSSL密 码服务接口，它屏蔽了具体算法的细节，为上层应用提供统一、抽象的接口。该接口的头文件 为openssl/evp.h。所对应的函数库为libcrypto。

【例2.14】在CentOS 7下用代码验证GmSSL

( 1 ) 在Windows 下 打 开UE (或其他编辑器),然后输入代码如下：

#include <openssl/conf.h> #include <openssl/evp.h> #include <openssl/err.h>

int main(int arc,char \*argv[])



/\*Load the human readable error strings for libcrypto \*/ ERR\_load\_crypto\_strings);

/\*Load all digest and cipher algorithms\*/ OpenSSL\_add\_all\_algorithms();

/\*Load config file,and other important initialisation \* OPENSSL\_config(NULL)

/\*...Do some crypto stuff here ...\*/

printf("Under Linux,call GmSSL lib ok\n") /\*Clean up\*

/\*Removes alldigests and ciphers \*/ EVP\_cleanup(



**第** **2** **章** **搭** **建C** **和C++** **密码开发环境**

/\*if you omit the next,a small leak may be left when you make use of the BIO(low level APD)for e.g.

base64 transformations\*/

CRYPTO\_cleanup\_all\_ex\_data();

/\*Remove error strings\*/ ERR\_free\_strings()

return 0; }

(2)保存代码为test.cpp, 上传到CentOS 7下，在命令行下编译并运行：

[root@localhost test]#g++test.cpp -o test -I/usr/local/mygmssl/include -Lusr/local/mygmssllib -lcrypto [root@localhost test]#./test

Under Linux,call GmSSL lib ok

至此，在Linux下用代码验证GmSSL成功。

**2.4.7** **默认编译安装** **GmSSL**

考虑到不少朋友或许更喜欢在默认路径下安装GmSSL, 因此我们进行默认配置、编译和 安装。这里使用的是CentOS 7(也可以使用其他版本的Linux) 。以 root账户登录Linux, 把 下载下来的GmSSL压缩包GmSSL-master.zip放到Linux下，然后解压：

unzip GmSSL-master.zip

接着进入文件夹GmSSL-master, 开始配置：

[root@localhost soft]#cd GmSSL-master/ [root@localhost GmSSL-master]#./config Operating system:x86\_64-whatever-linux2 Configuring for linux-x86\_64

Configuring GmSSL version 2.5.4(0x1010004fL)

no-asan [default]OPENSSL\_NO\_ASAN

no-crypto-mdebug [default]OPENSSL\_NO\_CRYPTO\_MDEBUG

no-crypto-mdebug-backtrace [default]OPENSSL\_NO\_CRYPTO\_MDEBUG\_BACKTRACE no-ec\_nistp\_64\_gcc\_ 128 [default]OPENSSL\_NO\_EC\_NISTP\_64\_GCC\_ 128

no-egd [default]OPENSSL\_NO\_EGD

no-fuzz-afl [default]OPENSSL\_NO\_FUZZ\_AFL

no-fuzz-libfuzzer [default]OPENSSL\_NO\_FUZZ\_LIBFUZZER no-gmieng [default]OPENSSL\_NO\_GMIENG

no-heartbeats [default]OPENSSL\_NO\_HEARTBEATS

no-md2 [default]OPENSSL\_NO\_MD2(skip dir)

no-msan [default]OPENSSL\_NO\_MSAN

no-rc5 [default]OPENSSL\_NO\_RC5(skip dir)

no-sctp [default]OPENSSL\_NO\_SCTP

no-sdfeng [default]OPENSSL\_NO\_SDFENG

no-skfeng [default]OPENSSL\_NO\_SKFENG

no-ssl-trace [default]OPENSSL\_NO\_SSL\_TRACE

no-ssl3 [default]OPENSSL\_NO\_SSL3

no-ssl3-method [default]OPENSSL\_NO\_SSL3\_METHOD



**Windows C/C++加密解密实战**

no-ubsan [default]OPENSSL\_NO\_UBSAN

no-unit-test [default]OPENSSL\_NO\_UNIT\_TEST

no-weak-ssl-ciphers [default]OPENSSL\_NO\_WEAK\_SSL\_CIPHERS no-zlib [default

no-zlib-dynamic [default] Configuring for linux-x86\_64 CC =gcc

CFLAG =-Wall -O3-pthread -m64 -DL\_ENDIAN -Wa,--noexecstack

SHARED\_CFLAG =-fPIC -DOPENSSL\_USE\_NODELETE

DEFINES =DSO DLFCN HAVE\_DLFCN\_H NDEBUG OPENSSL\_THREADS

OPENSSL\_NO\_STATIC\_ENGINE OPENSSL\_PIC OPENSSL\_IA32\_SSE2 OPENSSL\_BN\_ASM\_MONT OPENSSL\_BN\_ASM\_MONT5 OPENSSL\_BN\_ASM\_GF2m SHA1\_ASM SHA256\_ASM SHA512\_ASM RC4\_ASM MD5\_ASM AES\_ASM VPAES\_ASM BSAES\_ASM GHASH\_ASM ECP\_NISTZ256\_ASM

PADLOCK\_ASM GMI\_ASM POLY1305\_ASM

LFLAG =

PLIB\_LFLAG =

EX\_LIBS =-ldl

APPS OBJ =

CPUID OBJ =x86\_64cpuid.o

UPLINK OBJ 二

BN\_ASM =asm/x86\_64-gcc.o x86\_64-mont.o x86\_64-mont5.0 x86\_64-gf2m.o rsaz\_exp.0 rsaz-x86\_64.o rsaz-avx2.0

EC\_ASM =ecp\_nistz256.0 ecp\_nistz256-x86\_64.0 ecp\_sm2z256.0 ecp\_sm2z256-x86\_64.0

DES\_ENC =des\_enc.o fcrypt\_b.o

AES\_ENC =aes-x86\_64.0 vpaes-x86\_64.0 bsaes-x86\_64.o aesni-x86\_64.o aesni-shal-x86\_64.0 aesni-sha256-x86\_64.o aesni-mb-x86\_64.0

BF\_ENC =bf enc.0

CAST ENC =c\_enc.o

RC4\_ENC =rc4-x86\_64.o rc4-md5-x86\_64.0

RC5\_ENC =rc5\_enc.o

MD5\_OBJ\_ASM =md5-x86\_64.0

SHA1\_OBJ\_ASM =shal-x86\_64.o sha256-x86\_64.o sha512-x86\_64.o shal-mb-x86\_64.0 sha256-mb-x86\_64.0

RMD160\_OBJ\_ASM=

CMLL ENC =cmll-x86\_64.0 cmll\_misc.o

MODES\_OBJ =ghash-x86\_64.o aesni-gcm-x86\_64.0

PADLOCK\_OBJ =e\_padlock-x86\_64.0 GMI\_OBJ =e\_gmi-x86\_64.0

CHACHA\_ENC =chacha-x86\_64.0 POLY1305\_OBJ =polyl305-x8664.0

BLAKE2\_OBJ 

PROCESSOR =

RANLIB =ranlib

ARFLAGS =

PERL =/usr/bin/perl

SIXTY\_FOUR\_BIT\_LONG mode

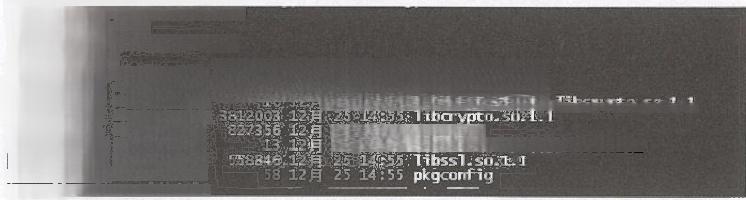
配置完毕后，开始漫长的编译： root@localhost GmSSL-master]#make



**第** **2** **章** **搭** **建C** **和C++** **密码开发环境**

这个过程有点长，读者可以去泡壶茶。编译完毕，开始安装： [root@localhost GmSSL-master]#make install

这个过程也稍长，可以喝会茶。安装完毕后，我们可以进入/usr/local/include,使 用 1s 查 看可以发现多了openssl目录。在/usr/locallib64/下也多了静态库(libcrypto.a和libssl.a) 和共 享库(libcrypto.soh和libssl.so), 如图2-57所示。



t4 1]]d

rwxr-xr-x.2 root root rw-r--r-- 1 root root

rF 1 root root

-rwxr-Xr-x. 1 root root rw-r--r--. 1 root root Irwxrwxrwx. 1 root root rwxr-xr-x. 1 root root drwxr-xr-x 2 root root

lrootalocalhost lib6414

1 0 8 张

4 yg0S oayu0.S0.1.

::1ibslss1->..s0.1.1

2

图2-57

下面做两个软链接，让gmssl 程序可以找到libcrypto.so.1.1. 和 libssl.so.1.1:



[root@localhost local]#In -s usr/locallib64libcrypto.so.1.1 /usr/lib64libcrypto.so.1.1

[root@localhost local]#In-s/usr/local/lib64/libssl.so.1.1 /usr/lib64/libssl.so.1

In 是建立软链接的命令，用法为“In -s 源文件目标文件”。源：实际存放文件的位置 再执行gmssl, 发现可以运行了：

[root@localhost local]#gmssl GmSSL>version

GmSSL2.5.4-OpenSSL1.1.0d 3 Sep 2019 GmSSL>quit

[root@localhost local]#

下面在普通Linux 命令行下直接测试SM3:



[root@localhost test]#echo -n "abc"|gmssl sm3

(stdin)=66c7f0f462eeedd9d1f2d46bdc10e4e24167c4875cf2f7a2297da02b8f4ba8e0

命令行验证结束，下面再用程序来验证。

【例2.15】在CentOS 7下用代码验证默认安装的GmSSL

(1)在Windows下打开UE (或其他编辑器),然后输入代码如下：

#include <openssl/conf.h> #include <openssl/evp.h> #include<openssl/err.h>

int main(intarc,char \*argv[]) {

/\*Load the human readable error strings for libcrypto\*/ ERR\_load\_crypto\_strings0;

/\*Load all digest and cipher algorithms \*



**Windows C/C++加密解密实战**

OpenSSL\_add\_all\_algorithms();

/\*Load config file,and other important initialisation \*/ OPENSSL\_config(NULL);

/\*...Do some crypto stuff here...\*/

printf("Under Linux,call GmSSL lib ok\n"); /\*Clean up\*/

/\*Removes all digests and ciphers \*/ EVP\_cleanup(;

/\*if you omit the next,a small leak may be left when you make use of the BIO(low level API)for e.g base64 transformations\*/

CRYPTO\_cleanup\_all\_ex\_data();

/\*Remove error strings \*/

ERR free\_strings( return 0;

(2)保存代码为 test.cpp, 上 传 到CentOS7 下，在命令行下编译并运行：

[root@localhost test]#g++test.cpp -o test -I/usr/local/include -L/usr/local/lib64/-lcrypto [root@localhost test]#./test

Under Linux,call GmSSL lib ok

至此，在Linux 下用代码验证默认安装的GmSSL 成 功 。

**2.4.8在老版本的Linux下编译安装** **GmSSL**

考虑到一些老项目所在的平台是老内核版本的 Linux, 但 也 想 用 一 下 GmSSL, 因 此 我 们

在 Linux 内核2.6的一些操作系统下编译、安装和测试GmSSL。

**1.编译、安装Perl**

这里采用内核是2 .6的老版本Linux, 通 常 会 提 示 少 了Perl:



[root@localhost GmSSL-master]#./config Operating system:x86\_64-whatever-linux2

Perl v5.10.0 required--this is only v5.8.8,stopped at ./Configure line 13.

Perl v5.10.0 required--this is only v5.8.8,stopped at ./Configure line 13.

This system (linux-x86\_64)is not supported.See file INSTALL for details

提示现在系统中只有版本为5.8.8的Perl, 而 GmSSL 需要5.10.0版本。所以需要先装5.10.0 版 本 的Perl。

我们也可以通过perl-v 来查看现有版本：

[root@localhost perl-5.10.0]#perl -v



**第** **2** **章** **搭** **建C 和C++密码开发环境**

This is perl,v5.8.8 built for x86\_64-linux-thread-multi Copyright 1987-2006,Larry Wall

Perl maybe copied only under the terms of either the Artistic License or the GNU General Public License,which may be found in the Perl 5 source kit.

Complete documentation for Perl,including FAQ lists,should be found on this system using"man perl"or "perldoc perl".If you have access to the Internet,point your browser at htp:/www.perl.org/,the Perl Home Page

并且系统自带的Perl 程序位于/usr/bin/ 下，我们可以用命令11查看：



[root@localhost perl-5.10.0]#1l/usr/bin/perl

-rWxr-xr-x 2 root root 193602007-10-1901:35/usr/bin/perl

下面开始编译安装Perl 5.10,并将老版本替换掉。首先是配置： [root@localhost perl-5.10.0]#/Configure -des -Dprefix=/usr/local/perl

参数-Dprefix指定安装目录为/usr/local/perl。 然后就是make 和 make install:

[root@localhost perl-5.10.0]#make

稍等片刻，编译完毕，开始安装： [root@localhost perl-5.10.0]#make install

如果这个过程没有错误的话，那么恭喜你安装完成了，是不是很简单?接下来替换系统原 有 的Perl, 有新的就使用新的。

#mv /usr/bin/perl /usr/bin/perl.bak

#In -s/usr/local/perl/bin/perl /usr/bin/perl

此时，如果使用 perl -v 查看版本，对于有些老系统会提示没有这个文件，必须重启操作 系统，然后就可以看到新版本提示了：

[root@localhost ~]#perl-v

This is perl,v5.10.0 built for x86\_64-linux Copyright 1987-2007,Larry Wall

Perl maybe copied only under the terms of either the Artistic License or the GNU General Public License,which may be found in the Perl 5 source kit

Complete documentation for Perl,including FAQlists,should be found on this system using"man perl"or "perldoc perl".If you have access to the Internet,point your browser at <http://www.perl.org/>,the Perl Home Page.

**2.编译安装GmSSL**

Perl 5.10.0安装升级成功后，就开始安装GmSSL 。安 装GmSSL 的步骤和前面一样。最好



**Windows** **C/C++加密解密实战**

先设置目录下所有的文件为最高权限： chmod -R 777 GmSSL-master

其中，-R 表示级联应用到目录里的所有子目录和文件，777表示所有用户都拥有最高权限 (可自定权限码)。

然后进入GmSSL-master,开始三部曲：config、make和make install:

[root@localhost GmSSL-master]#./config [root@localhost GmSSL-master]#./make

[root@localhostGmSSL-master]#./make install

稍等片刻，安装完毕后，将在/usr/local/lib64/下生成库文件：

[root@localhost lib64]#ls

engines-1.1 libcrypto.a libcrypto.so libcrypto.so.1.1 libssl.a libssl.so libssl.so.1.1 pkgconfig [root@localhost lib64]#pwd

/usr/local/lib64

并且，将在/usr/local/include/下生成头文件所在的目录 openssl:

[root@localhost include]#ls

ansidecl.h bfd.h bfdlink.h dis-asm.h gdb openssl plugin-api.h symcat.h [root@localhost include]#pwd

/usr/local/include

此时若执行gmssl 程序，则会发现是执行不了的，提示少了库。下面做两个软链接，让 gmssl 程序可以找到libcrypto.so.1.1. 和 libssl.so.1.1:

[root@localhost local]#In-s /usr/locallib64/libcrypto.so.1.1 /usr/lib64/liberypto.so.1.1

[root@localhost local]#In-s /usr/local/lib64/libssl.so.1.1 /usr/lib64/libssl.so.1.1

In 是建立软链接的命令，用法为“In -s 源文件目标文件”。源：实际存放文件的位置。 再执行 gmssl, 发现可以运行了：

[root@localhost include]#gmssl GmSSL>version

GmSSL2.5.4-OpenSSL 1.1.0d 3 Sep2019 GmSSL>

最后用代码验证一下。

【例2.16】在CentOS 7下用代码验证默认安装的GmSSL

(1)在Windows下打开UE(或其他编辑器),然后输入代码如下：

#include <openssl/conf.h> #include <opensslevp.h> #include <openssl/err.h>

int main(intarc,char \*argv[]) {

/\*Load the human readable error strings for libcrypto\*/



**第** **2** **章** **搭建C** **和C++** **密码开发环境**

ERR\_load\_crypto\_strings()

/\*Load all digest and cipher algorithms\*/ OpenSSL\_add\_all\_algorithms();

/\*Load config file,and other important initialisation \*/ OPENSSL\_config(NULL);

/\*...Do some crypto stuff here..\*/

printf("Under Linux,call GmSSL lib ok\n"); /\*Clean up\*/

/\*Removes all digests and ciphers \*/ EVP cleanupO

/\*if you omit the next,a small leak may be left when you make use of the BIO(low level API)for e.g. base64 transformations\*/

CRYPTO\_cleanup\_all\_ex\_data(); /\*Remove error strings \*/

ERR\_free\_strings() return 0;

(2)保存代码为test.cpp, 上传到CentOS 7下，在命令行下编译并运行：



[root@localhost test]#g++test.cpp -o test -Vusr/local/include -L/usr/local/lib64/-lcrypto [root@localhost test]#/test

Under Linux,call GmSSL lib ok

至此，在老版本的Linux下用代码验证默认安装的GmSSL成功。