

安全多方计算在基于区块链技术的信用评级平台中的应用

课程名称： 信息安全
院系： 计算机学院
专业： 软件工程
年级： 2016 级
姓名： 刘瑞康
学号： 2016302580242

摘要

若对银行领域应用安全多方计算进行优化, 不仅能解决信息封闭的问题, 还能加快整个查询流程的效率。同时, 由于其独到的优势, 将会吸引更多银行等企业参与到该平台中来, 随着参与多方计算的节点和数据量的增加, 将进一步扩大信用评级范围, 提升计算结果的准确性, 从而帮助该领域中各个参与成员实现高效的反信用欺诈。

Abstract

If we apply the secure multi-party computing in the banking field, we can not only solve the problem of the closeness of the information, but also improve the efficiency of the whole query process. At the same time, because of its unique advantages, it will attract more banks and other enterprises to participate in the platform. With the increase of the number of nodes and data, the scope of credit rating will be further expanded and the accuracy of the calculation results will be enhanced, thus helping the members of the field to achieve high efficiency and anti credit fraud.

关键字

安全多方计算、银行、信用评级、反欺诈、平台

引言

目前，信用风险是银行业面临的主要风险，因此反欺诈仍是银行业面临的重中之重。相关研究报告显示，全球每年因欺诈损失的总额已超过了 500 亿美元。然而目前，各银行间的信息系统相对封闭，尽管处在整个金融界的大数据环境中，却不能做到有效的互联互通，这进一步制约了大数据风控的发展。我所在的小组参加了今年的“花旗杯”金融创新应用大赛，设计了一款在保护银行客户信息不被泄露的前提下实现一定程度信息共享的信用评级平台。当平台的某一成员需要获得某用户在整个平台上的信用评级信息时，平台通过安全多方计算，在确保平台各成员输入的独立性、计算的正确性以及各输入值不泄露给参与计算的其他任何成员的前提下，将计算结果储存在区块链上保证录入的信息无法被篡改。目前，该项目还在开发过程中，本人参与了其中安全多方计算模块的开发，以此为基础进行本次课程考核的论文撰写。

国内外研究现状

安全多方计算(Secure Multi-Party Computation，即 SMC)作为密码学应用研究的一个范例，从该问题的提出开始(由百万富翁问题提出)，就一直受到国内外密码界的关注，并成为研究的热点。随着互联网的普及、移动互联应用的推广，在很大程度上推动了安全多方计算的实际应用，如匿名投票、网络竞拍等。

但在金融信用领域，如果想要在授信前对客户信用状况进行查询，国内目前大多采用的是银行工作人员通过央行系统，支付查询手续费后等待信息反馈的形式；而国外则由于大多数银行为私有资本企业，缺少中央统一管理平台，因而多采用仅参考客户在自家私有数据库中信用状况的方式。而实际情况表明，国内的方式虽然一定程度上确保了信息的权威和统一，但通过该方式能查询到的信息大多是可公开的房屋抵押等数据，不能保证信息的完整性，另外还增大借贷流程的繁琐程度；而另一方面，国外的方式则难以避免客户利用信息封闭的漏洞，在多家银行同时进行信用欺诈等不良信用行为。

在这种情形下，向该领域引入安全多方计算将带来巨大的提升。安全多方计算的研究正是针对无可信第三方的情况下，如何安全地计算一个约定函数、解决一组互不信任的参与方之间保护隐私的协同计算的问题。在衡量放贷风险的时候，银行想要通过了解某个客户在其他银行的借贷还贷等各种数据，来避免因向不良信用客户放贷而造成的损失。但客户借贷等数据作为银行的高度隐私，肯定没有银行愿意透露自身客户的相关隐私，此时引入信用评级和安全多方计算的模式。平台中每一个拥有相应客户数据的企业或银行都是计算中的一个节点，当某个节点发送查询某个客户的请求的时候，就会触发安全多方计算，每一个节点独立地提供相关数据，通过安全多方计算，得到根据信用模型和各方输入计算出的客户的整体信用评级，在整个过程中并不会泄露任何银行或客户的隐私信息。

由此分析，若对银行领域应用安全多方计算进行优化，不仅能解决信息封闭的问题，还能加快整个查询流程的效率，同时，由于以上独到的优势，将会吸引更多银行等企业参与到该平台中来，随着参与多方计算的节点和数据量的增加，将进一步扩大信用评级范围，提升计算结果的准确性，从而帮助该领域中各个参与成员实现高效的反信用欺诈。

个人研究工作

实际 SMC 模型由以下四个方面组成：

- ① 参与方。
- ② 安全性定义。
- ③ 通信网模型。
- ④ 信息论安全与密码学安全。

为了便于实现和描述，本次仅实现了简化后的 SMC 模型：

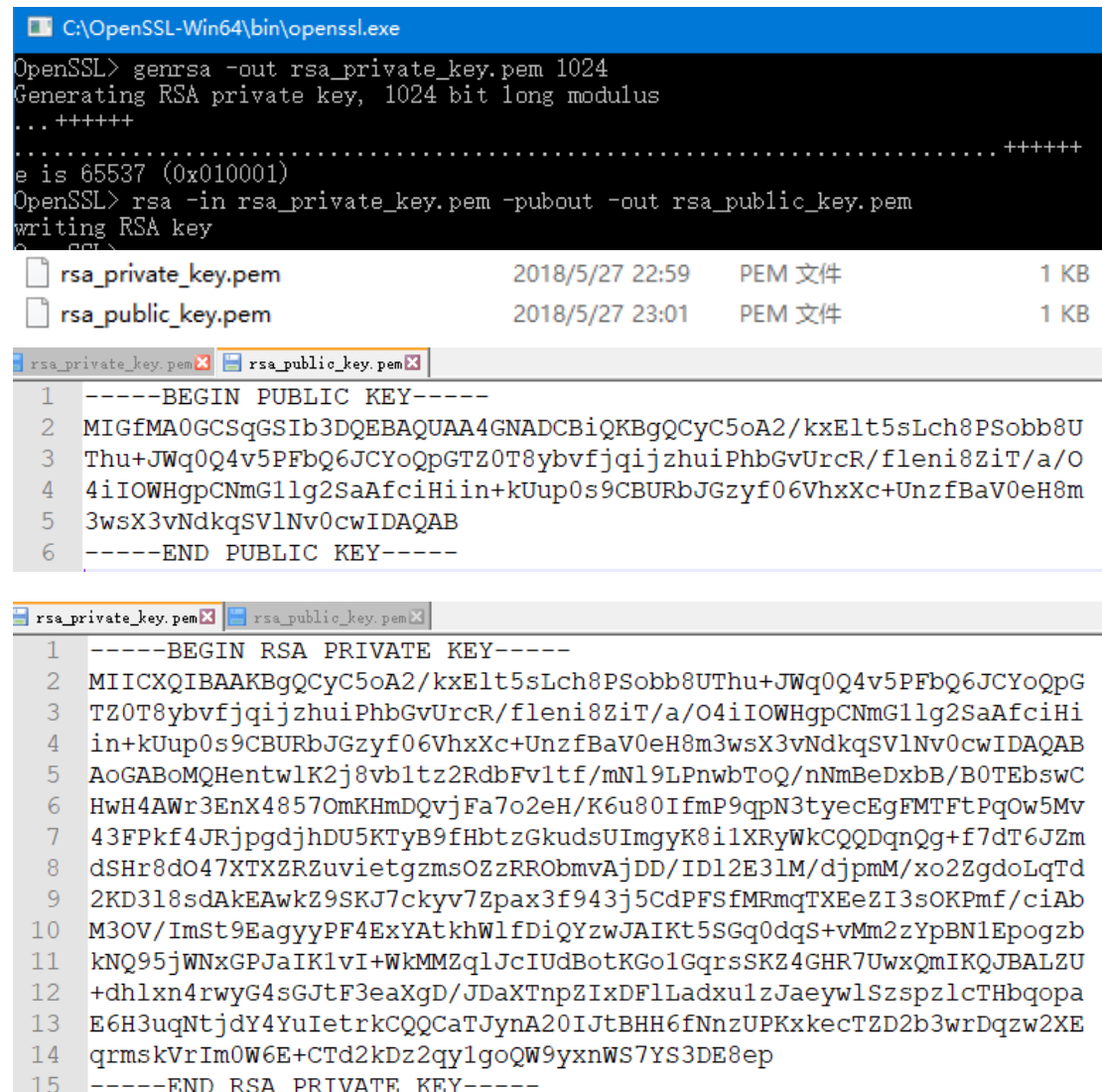
- ① 由于参与 SMC 计算的参与者互不信任，参与者也许是恶意参与者、攻击者或半诚实参与者。为了解决在一个不可信的环境下，不可信参与者参与计算的条件下，各方保证不泄漏自己的输入的情况下，实现 SMC 计算，会极大程度的要求整个 SMC 模型的可靠度和安全性，导致模型复杂度增加。在基于区块链技术的信用评级平台中，假设计算的各参与方都是取得了国家权威机构认证的银行或企业，因而排除恶意参与者参与计算的可能。
- ② 假定存在安全可靠的可信第三方，即运行该平台核心计算部分的机构（如央行），在该前提下，运行在平台服务器上的数据不会被泄密同时不会被非法获取。平台负责对实际数据计算处理后根据模型生成客户的信用等级评价报告，将报告发送回查询发起者即请求方，同时将报告内容储存到区块链上。
- ③ 由于本人专业所限，对金融领域知识不甚了解，具体模型仍待经管专业的同学进行最后的调整和完善，客户数据和生成的报告所包含的内容均为简化后的象征性内容。
- ④ 由于没有多服务器和节点的支持，在本地模拟整个安全多方计算的过程。因为本人不参与此项目区块链储存的部分，因此相应函数使用空函数体的实现。

流程

- ① 运行服务器平台
- ② 参与平台的节点进行注册
- ③ 平台等待请求
- ④ 某节点向平台发送查询请求
- ⑤ 平台确认请求合法后将请求转发给全部节点
- ⑥ 各节点分别在自己的数据库信息系统中查询请求中客户的信息，使用从平台获得的公钥对信息加密后发送到平台
- ⑦ 平台确认已获得全部节点的加密信息后，对每个节点的信息使用私钥进行解密。
- ⑧ 将全部内容进行汇总后，根据模型（简化后，只根据在全部节点的预期贷款总量来判断）产生该用户的信用评级信息，并将报告返还给最初发送查询请求的节点，同时将报告信息储存在区块链上（空函数体，未实现）。

实现步骤

- ① 下载并安装 OpenSSL 库，使用 OpenSSL 结构生成 RSA 对应的公钥密钥对。



The screenshot shows the OpenSSL command line interface and a file explorer. The command line shows the generation of a 1024-bit RSA private key and its corresponding public key. The file explorer shows the generated files: rsa_private_key.pem and rsa_public_key.pem. The public key content is displayed in a text editor, showing the BEGIN PUBLIC KEY and END PUBLIC KEY markers, and the private key content is also displayed, showing the BEGIN RSA PRIVATE KEY and END RSA PRIVATE KEY markers.

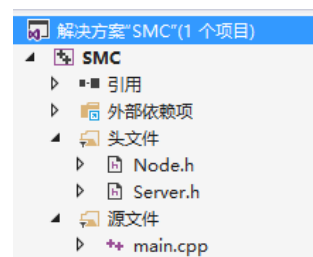
```
C:\OpenSSL-Win64\bin\openssl.exe
OpenSSL> genrsa -out rsa_private_key.pem 1024
Generating RSA private key, 1024 bit long modulus
...+++++
e is 65537 (0x010001)
OpenSSL> rsa -in rsa_private_key.pem -pubout -out rsa_public_key.pem
writing RSA key

rsa_private_key.pem      2018/5/27 22:59    PEM 文件      1 KB
rsa_public_key.pem      2018/5/27 23:01    PEM 文件      1 KB

rsa_private_key.pem x  rsa_public_key.pem x
1  -----BEGIN PUBLIC KEY-----
2  MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC5oA2/kxElt5sLch8PSobb8U
3  Thu+JWq0Q4v5PFbQ6JCYoQpGTZ0T8ybvffqijzhuiPhbGvUrcR/fleni8ZiT/a/O
4  4iIOWHgpCNmG1lg2SaAfcHiin+kUup0s9CBURbJGzyf06VhxXc+UnzfBaV0eH8m
5  3wsX3vNdkqSV1Nv0cwIDAQAB
6  -----END PUBLIC KEY-----

rsa_private_key.pem x  rsa_public_key.pem x
1  -----BEGIN RSA PRIVATE KEY-----
2  MIICXQIBAAKBgQC5oA2/kxElt5sLch8PSobb8UThu+JWq0Q4v5PFbQ6JCYoQpG
3  TZ0T8ybvffqijzhuiPhbGvUrcR/fleni8ZiT/a/O4iIOWHgpCNmG1lg2SaAfcHi
4  in+kUup0s9CBURbJGzyf06VhxXc+UnzfBaV0eH8m3wsX3vNdkqSV1Nv0cwIDAQAB
5  AoGABoMQHentwK2j8vb1tz2RdbFvltf/mNl9LPnwbToQ/nNmBeDxbB/B0TEbswC
6  HwH4AWr3EnX4857OmKHMDQvjFa7o2eH/K6u80IfmP9qpN3tyecEgFMTFtPqOw5Mv
7  43FPkf4JRjpgdjhdU5KTyB9fHbtzGkudsUImgyK8ilXRyWkCQQDqnQg+f7dT6JZm
8  dSHr8d047XTXZRZuvietgzmsOZZRRObmVajDD/IDl2E3lM/djpmM/xo2ZgdoLqTd
9  2KD3l8sdAkeAwkZ9SKJ7ckyv7Zpax3f943j5CdPFSfMRmqTXEeZI3sOKPmf/ciAb
10 M3OV/ImSt9EagyyPF4ExYAtkhWlfdiQYzwJAikt5SGq0dqS+vMm2zYpBN1Epogzb
11 kNq95jWNxGPJaIK1vI+WkMMZqlJciUdBotKGolGqrsSKZ4GHR7UwxQmIKQJBALZU
12 +dh1xn4rwyG4sGJtF3eaXgD/JDaXTnpZIXDFlLadXulzJaeyw1SzspzlcTHbqopa
13 E6H3uqNtjdY4YuIetrkCQQCaTJynA20IJtBHH6fNnzUPKxkeCTZD2b3wrDqzw2XE
14 qrmskVrIm0W6E+CTd2kDz2qylgoQW9yxNWS7YS3DE8ep
15 -----END RSA PRIVATE KEY-----
```

- ② 实现 Node 类和 Server 类，分别模拟参与平台的节点和平台本身的相应行为。此处只提供头文件内容。



```

#pragma once
#define _CRT_NONSTDC_NO_WARNINGS
#pragma warning(disable:4996)

#include <openssl/rsa.h>
#include <openssl/pem.h>
#include <io.h>
#include <fstream>
#ifdef __cplusplus
extern "C" {
#endif
#include <openssl/applink.c>
#ifdef __cplusplus
}
#endif

using namespace std;

#pragma comment(lib, "libssl.lib")
#pragma comment(lib, "libcrypto.lib")

class Node
{
public:
    char* name;
    Node(char* _name) { name = _name; }
    bool sendRequest( char* requestContent);
    unsigned char* requestProcessing();

private:
    unsigned char* EncryptUsingPublicKey(const unsigned char* plaintext, unsigned char*
publicKey);
    RSA* createRSA(unsigned char* publicKey);
};

#pragma once
#define _CRT_NONSTDC_NO_WARNINGS
#pragma warning(disable:4996)

#include <openssl/rsa.h>
#include <openssl/pem.h>
#include <io.h>

```

```

#include <fstream>
#include <list>
#include "Node.h"

using namespace std;

#pragma comment(lib, "libssl.lib")
#pragma comment(lib, "libcrypto.lib")

class Server
{
public:
    Node* registerNewNode(char* name);
    bool sendReport(Node* node, unsigned char* reportContent);
    bool sendQueryRequestToAll();

private:
    RSA * createRSA(unsigned char* privateKey);
    bool sendQueryRequest(Node* node, unsigned char* requestContent);
    bool processInfo();
    unsigned char* DecryptUsingPrivateKey(unsigned char* ciphertext, unsigned char*
privateKey);
    unsigned char* getPrivateKey();
    unsigned char* getPublicKey();
    bool storeToBlockChain();

    list<Node*> nodeList;
    Node* requestingNode;
};

```

- ③ 部分调用 OpenSSL 库进行封装的函数实现。包括利用公钥私钥构造 RSA 结构体，以及利用 RSA 结构体分别实现明文加密和密文解密。

```

RSA* Node::createRSA(unsigned char* privateKey)
{
    RSA *rsa = NULL;
    BIO *keybio;
    keybio = BIO_new_mem_buf(privateKey, -1);
    if (keybio == NULL)
    {
        printf("Failed to create key BIO");
        return NULL;
    }
}

```

```

        rsa = PEM_read_bio_RSAPrivateKey(keybio, &rsa, NULL, NULL);
        cout << "Use private key create RSA successfully." << endl;
        return rsa;
    }

RSA * Server::createRSA(unsigned char* privateKey)
{
    RSA *rsa = NULL;
    BIO *keybio;
    keybio = BIO_new_mem_buf(privateKey, -1);
    if (keybio == NULL)
    {
        printf("Failed to create key BIO");
        return NULL;
    }
    rsa = PEM_read_bio_RSAPrivateKey(keybio, &rsa, NULL, NULL);

    return rsa;
}

unsigned char* Node::EncryptUsingPublicKey(const unsigned char* plaintext, unsigned
char* publicKey)
{
    int length = strlen((const char *)plaintext)+1;
    unsigned char ciphertext[1024] = {};

    int encLength = RSA_public_encrypt(length, plaintext, ciphertext,
createRSA(publicKey), RSA_PKCS1_PADDING);

    if (encLength == -1)
        return NULL;
    return ciphertext;
}

unsigned char* Server::DecryptUsingPrivateKey(unsigned char* ciphertext, unsigned char*
privateKey)
{
    int length = strlen((const char *)ciphertext)+1;
    unsigned char plaintext[1024] = {};

    int decLength = RSA_private_decrypt(length, ciphertext, ciphertext,
createRSA(privateKey), RSA_PKCS1_PADDING);

    if (decLength == -1)

```

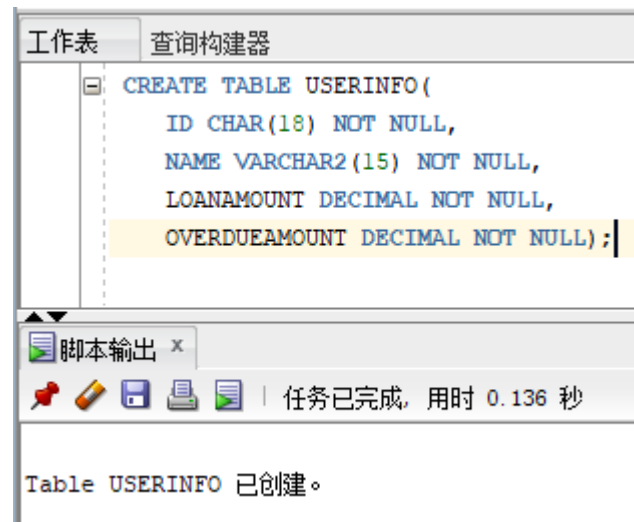


```

        return NULL;
    return plaintext;
}

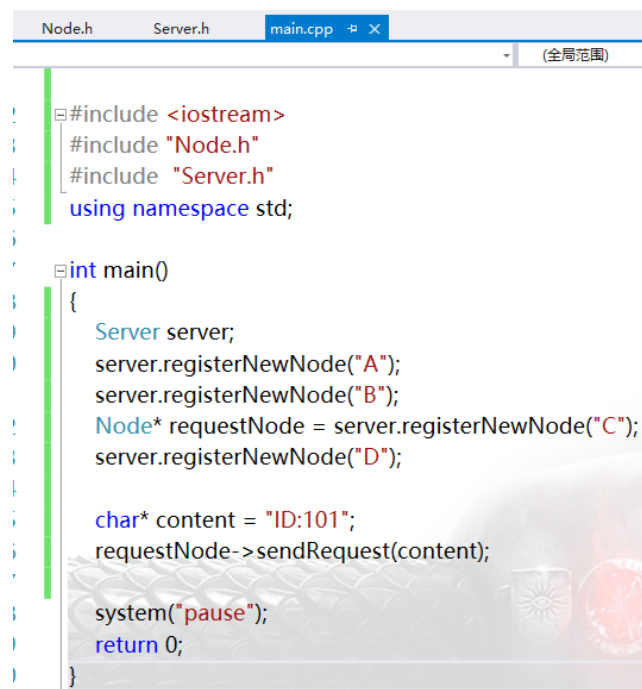
```

④ 模拟数据库储存用户信息，以下为简化后的模拟用户信息表。



测试分析

数据库中共模拟了 3 个人每人 4 份的信息，ID 分别为 101,102,103. 首先尝试使用 101 请求，通过 C 节点发送请求。



以下为控制台 Log 输出。

```
C:\Users\TR\Documents\Visual Studio 2015\Projects\SMC\x64\
Node A has registered on the server
Node B has registered on the server
Node C has registered on the server
Node D has registered on the server
Node C is sending request
Query request sends to node A
Query request sends to node B
Query request sends to node C
Query request sends to node D
Node A processing request
Node B processing request
Node C processing request
Node D processing request
Start to process retrived data
Process finished
Report sends to node C
请按任意键继续. . .
```

以下为生成的报告内容。

```
C:\Users\TR\Documents\Visual Studio 2015\Projects\SMC\x64\Debug\Report_ID101.txt -
文件(F) 编辑(E) 搜索(S) 视图(V) 格式(M) 语言(L) 设置(I) 宏(O) 运行(R) 插件(P) 窗
Report_ID101. txt
1 ID:101
2 Credit rating: good
```

然后使用 ID 为 103 进行尝试，此次使用 B 节点发送查询请求。

```
int main()
{
    Server server;
    server.registerNewNode("A");
    Node* requestNode = server.registerNewNode("B");
    server.registerNewNode("C");
    server.registerNewNode("D");

    char* content = "ID:103";
    requestNode->sendRequest(content);

    system("pause");
    return 0;
}
```

以下为控制台 Log 输出。

```
C:\Users\TR\Documents\Visual Studio 2015\Projects\SMC\x64\Debug
Node A has registered on the server
Node B has registered on the server
Node C has registered on the server
Node D has registered on the server
Node B is sending request
Query request sends to node A
Query request sends to node B
Query request sends to node C
Query request sends to node D
Node A processing request
Node B processing request
Node C processing request
Node D processing request
Start to process retrived data
Process finished
Report sends to node B
请按任意键继续. . .
```

以下为生成的报告内容。

```
C:\Users\TR\Documents\Visual Studio 2015\Projects\SMC\x64\Debug\Report_ID103.txt - Notepad
文件(F) 编辑(E) 搜索(S) 视图(V) 格式(M) 语言(L) 设置(T) 宏(O) 运行(R) 插件(P) 窗口(W)
Report_ID103.txt
1 ID:103
2 Credit rating: poor
```

总结

安全多方计算(Secure Multi-Party Computation , SMC)是针对无可信第三方的情况下,如何安全地计算一个约定函数、解决一组互不信任的参与方之间保护隐私的协同计算的问题。对银行领域应用安全多方计算进行优化,在保证客户和银行隐私不被泄露的前提下,不仅能解决信息封闭的问题,还能加快整个查询流程的效率,同时,由于以上独到的优势,将会吸引更多银行等企业参与到该平台中来,随着参与多方计算的节点和数据量的增加,将进一步扩大信用评级范围,提升计算结果的准确性,从而帮助该领域中各个参与成员实现高效的反信用欺诈。

参考文献列表

- [1] 范红, 冯登国. 安全协议理论与方法 [M]. 北京: 科学出版社, 2003.
- [2] [美]Christopher Swenson. 现代密码分析学——破译高级密码的技术[M]. 黄月江, 祝世雄, 张文政, 等译校. 北京: 国防工业出版社, 2012.
- [3] [美]Bruce Schneier. 应用密码学——协议、算法与 C 源程序[M]. 吴世忠, 祝世雄, 张文政, 等译校. 北京: 机械工业出版社, 2010.
- [4] [加]Alfred J Menezes, Paul C van Oorschot, Scott A Vanstone. 应用密码学手册[M]. 胡磊, 王鹏译校. 北京: 电子工业出版社, 2005
- [5] 张文科, 杨勇, 杨宇. 安全多方计算研究. [中图分类号] TN918. [文章编号] 1009-8054(2014)01-097-03