基于群签名和群密钥的安全通信系统

1 总体设计

本章介绍了本系统的总体设计，包括系统需求，系统总体方案，功能设计和关键技术。在系统方案中，介绍了系统结构和总体模块划分。在功能设计中，根据需求确定了系统的各项功能。在关键技术中，介绍了本系统所使用的网络、群签名、群密钥交换和加密算法。

1.1 系统方案

1.1.1 系统架构

整个系统包括基础架构和软件系统两个部分。基础架构包括硬件，系统，网络。本系统分为 GM 和 Member 两个角色，其中 GM 可以独立于 Member 存在。

系统结构如图 2.1 所示：

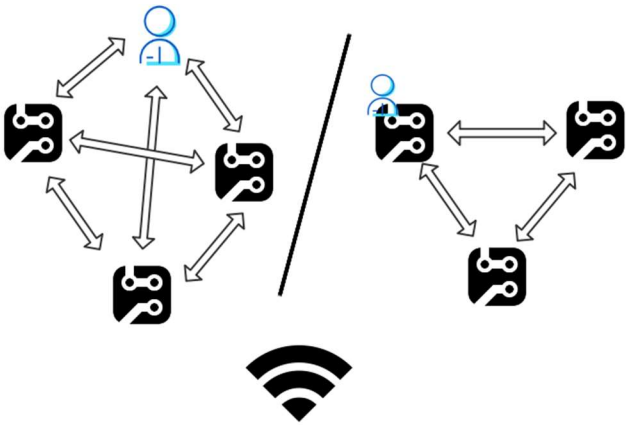

图 2.1 系统结构示意图

1.1.2 模块设计

软件部分分为顶层、网络层和内部逻辑层。

顶层模块控制调用内部层级，将内部网络与逻辑封装，控制密钥交换与群成员从入群到网络传输的整体流程和顺序，并及时将进程信息控制输出；

网络层作为系统的中间层，承接顶层控制层和内部逻辑层；网络层通过 Linux socket 编程实现节点与节点之间的网络连接与信息传递：通过制定专用的能够协同异构网络通信需要的网络架构和协议模型，整合顶层的数据；通过向应用层屏蔽通信网络传输信息的类型，为应用层提供透明的信息传递服务，充分利用现有网络资源，为顶层调用提供支撑，为逻辑层提供数据；

内部逻辑层通过实现 CamSta97 群签名方案进行群成员的消息签名，该群签名方案以基于离散对数的知识签名以及 RSA 算法为基础，群公钥和签名长度不依赖于群成员的个数，

也保证了群签名的基本安全性要求；群加密使用 AES 算法，保证安全性和高效率性；通过实现基于 GDH.2 的群密钥交换方案进行群成员之间的密钥交换，并使用生成的群密钥进行消息加密和解密。

这三个模块互相协调工作，共同构建了一个群组通讯安全通信系统。系统软件模块三层示意图如图 2.2 所示：

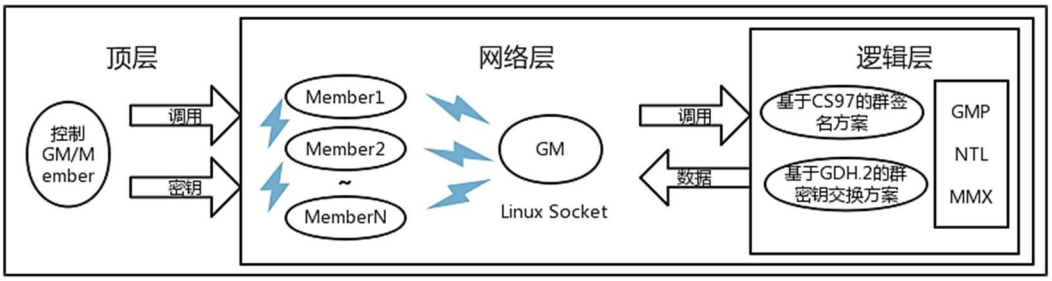

图 2.2 系统软件模块三层示意图

1.2 功能设计

本系统设计的功能在于在群组通信环境中，通过群签名和群密钥实现有具有身份隐私保护性、身份可追溯性、抵抗合谋攻击的能力的系统，同时具有较少管理开销，通信效率高等特点，能够对目前通信环境中存在的问题进行针对性的改进与创造性地提升。

（1）群组通信功能：通过网络传输，实现群组成员间彼此通信

（2）保密性功能：通过加密函数进行传输数据的保密，防止网络侦听；通过群密钥交换方式，保证秘钥不泄露，保证群组安全

（3）身份隐私保护功能：通过群签名方案的实现，保护群成员的身份匿名性

（4）身份可追溯功能：通过群签名和也能追查成员身份，防止虚假消息与欺诈

（5）抵抗攻击功能：能够抵抗群外成员假冒攻击、群内成员假冒攻击等多种攻击

1.3 关键技术

本章节将将整个系统分为不同的部分，分别介绍相应的关键技术：介绍网络部分实现的方法以及自定义的网络协议；介绍群签名部分使用的 CamSta97 群签名方案的实现原理，包括哈希函数、知识签名以及群签名的实现；介绍群密钥交换部分使用的 GDH.2 密钥交换协议；介绍加密算法。

1.3.1 C/S 网络模型

（1） 客户/服务器模式

在 TCP/IP 网络应用中，通信的两个进程间相互作用的主要模式是客户/服务器模式（Client/Server model），即客户向服务器发出服务请求，服务器接收到请求后，提供相应的服务。客户/服务器模式的建立基于以下两点：首先，建立网络的起因是网络中软硬件资源、运算能力和信息不均等，需要共享，从而造就拥有众多资源的主机提供服务，资源较少的客户请求服务这一非对等作用。其次，网间进程通信完全是异步的，相互通信的进程间既不存

在父子关系，又不共享内存缓冲区，因此需要一种机制为希望通信的进程间建立联系，为二者的数据交换提供同步。客户/服务器模式在工作过程中采取的是主动请求方式：

- 服务器方：

首先服务器方要先启动，并根据请求提供相应服务：

A. 打开一通信通道并告知本地主机，它愿意在某一 IP 地址上接收客户请求；
B. 等待客户请求到达该端口；
C. 接收到不同客户方服务请求，对不同请求进行分别的处理。服务完成后，关闭此新进程与客户的通信链路，并终止。
D. 关闭服务器

- 客户方：

A. 打开一通信通道，并连接到服务器所在主机的特定端口；
B. 向服务器发服务请求报文，等待并接收应答；或可继续提出请求；
C. 请求结束后关闭通信通道并终止。

（2） 网络实现与分析

网络通信使用 Linux socket 编程实现，分为 GM 与 Member 两个类别，各自有 TCP（Transmission Control Protocol，传输控制协议）连接和 UDP（User Datagram Protocol，用户数据报协议）连接两种类型，分别封装在头文件和源文件中。

TCP 是一种面向连接的、可靠的、基于字节流的传输层通信协议，用于除广播信息外的网络信息交换；在 TCP 连接过程中，其中 GM 作为服务端保持长时间连接准备，同时以非阻塞非 fork 的方式保持与客户端的网络收发包连接，可同时处理与多个客户端的连接；Member 作为客户端向 GM 发包发出连接请求，经历三次握手建立连接后可进行互相的信息收发，直到该客户端完成群组签名或验证其他操作，不再需要网络发包时，该连接终止。

UDP 是一种无连接的传输层协议，提供面向事务的简单不可靠信息传送服务，用于广播信息的网络信息交换；在 UDP 连接过程中，其中 GM 作为广播端通过建立 UDP 连接并设定套接字描述符以支持广播，并向指定的广播 IP 地址进行广播；Member 作为被广播端，确认接收到广播内容后 GM 端广播停止，连接终止。

网络传输中一律使用网络序传输。

1.3.2 CamSta97 群签名方案

CamSta97 方案（以下简称 CS97）是 Jan Camenisch 和 Markus Stadler 在“Efficient Group Signature Schemes For Large Groups”这篇论文中提出的方案，该方案以“离散对数”和 RSA 算法为理论基础，通过资料对比 CM98、ACJT2000 等群签名方案，该方案具有安全性较高、工作效率高等明显优点；在群签名的公共优点外，首先实现了群公钥和签名长度不依赖于群成员的个数，也首次在群签名方案中用到了知识签名[12]。下面具体介绍实现的 CS97 群签名方案的相关技术：

Murmur3 哈希函数，MurmurHash 是一种非加密型哈希函数，适用于一般的哈希检索操作。由 Austin Appleby 在 2008 年发明，并出现了多个变种，都已经发布到了公有领域(public domain)。与其它流行的哈希函数相比，对于规律性较强的 key，MurmurHash 的随机分布特

征表现更良好。在 Linux 系统中，MurmurHash 是默认的哈希实现方式。

知识签名，知识签名是指签名者利用数学知识和公共信息（本设计中是基于 Schnorr 签名方案的），在不泄露某个秘密的情况下，向别人证明自己知道这个秘密。

常见的知识签名有三种：离散对数的知识签名；双离散对数的知识签名；离散对数 e 次方根的知识签名。

定义：令$G =< g >$是一个阶为 n 的循环群，a 是Z_n^*中的一个元素。G 中元素 y 对于底 g 的离散对数值是满足$g^x = y$的最小正整数 x。类似地，y 对于底 g 和 a 的双离散对数值是满足$g^{(a^x)} = y$的最小正整数 x，如果 x 存在的话。其中，参数 n, G, g 和 a 的选择要使得在 G 上计算对于底 g 的离散对数和在Z_n^*上计算对于底 a 的离散对数是难解的。

此外，G 中元素 y 对底数 g 的离散对数值的 e 次方根是满足$g^{(x^e)} = y$的整数 x. 如果 n 的因式分解是未知的，则在Z_n^*上计算 e 次方根应为难解的。

设$H:\{0,1\}^* \to \{0,1\}^k$是一个能避免冲突的哈希函数。

(1) **离散对数的知识签名**

离散对数的知识签名针对的是$x = log_g y$提出的。只有当签名者知道群中元素 y 对底数 g 的离散对数 x 的值时才能进行签名。签名者需要给出一个(c, s)对，满足：

$$c = H(m\|y\|g\|g^s\|y^c) \tag{2.1}$$

其中 m, y, g 均为该签名算法的参数。

签名的构造方法如下：

① 随机选取 1 个数r ∈ Z_n^*，计算

$$c = H(m\|y\|g\|g^r) \tag{2.2}$$

② 计算

$$s = r - cx\ (mod\ n) \tag{2.3}$$

可以证明，按照上述步骤构造出来的(c, s)对满足式（2.1）。由于(c, s)对的构造过程中需要使用 x，故能够证明签名者知道 x.

(2) **双离散对数的知识签名**

双离散对数知识签名的符号表示为$SKLOGLOG[\alpha: y = g^{(a^\alpha)}](m)$。只有当签名者知道群中元素 y 对底数 g 和 a 的双离散对数 α 的值时才能进行签名。给定一个安全参数$l \le k$（其中 k 为哈希函数 H 的参数），签名者需要给出一个(c, s)对，其中s是一个l维向量。满足：

$$c = H(m\|y\|g\|a\|t_1\|\dots\|t_l)，其中 t_i = \begin{cases} g^{a^{s_i}}, if\ c[i] = 0 \\ y^{a^{s_i}}, otherwise \end{cases} \tag{2.4}$$

其中 m, y, g，a 均为该签名算法的参数。

签名的构造方法如下：

① 设 α 的位数上界是 γ，选择一个常数$\epsilon > 1$。对于$i = 1, \dots, l$，随机生成$r_i \in \{0, \dots, 2^{\epsilon\gamma} - 1\}$

② 计算

$$t_i^* = g^{(a^{r_i})}, for\ i = 1, \dots, l \tag{2.5}$$

③ 计算

$$c = H(m\|y\|g\|a\|t_1^*\|\ldots\|t_l^*) \quad (2.6)$$

④ 计算

$$s_i = \begin{cases} r_i & ,if\ c[i]=0 \\ r_i - \alpha, otherwise \end{cases} ,\text{for } i = 1,\ldots,l \quad (2.7)$$

可以证明，按照上述步骤构造出来的(c,s)对满足式（2.4）。由于(c,s)对的构造过程中需要使用 α，故能证明签名者知道 α.

(3) **离散对数e次方根的知识签名**

离散对数 e 次方根知识签名的符号表示为$SKROOTLOG[\alpha: y = g^{\alpha^e}](m)$。只有当签名者知道群中元素 y 对底数 g 的离散对数值的 e 次方根 α 的值时才能进行签名。给定一个安全参数$l \le k$，签名者需要给出一个(c,s)对，其中s是一个l维向量。满足：

$$c = H(m\|y\|g\|e\|t_1\|\ldots\|t_l)，\text{其中} t_i = \begin{cases} g^{s_i{}^e}, if\ c[i]=0 \\ y^{s_i{}^e}, otherwise \end{cases} \quad (2.8)$$

其中 m, y, g，e 均为该签名算法的参数。$s_i,\ldots,s_l$均属于Z_n^*，其中 n 是故均不为 0.

签名的构造方法如下：

① 对于$i = 1,\ldots,l$，随机生成$r_i \in Z_n^*$

② 计算

$$t_i^* = g^{(r_i{}^e)} ,\text{for } i = 1,\ldots,l \quad (2.9)$$

③ 计算

$$c = H(m\|y\|g\|e\|t_1^*\|\ldots\|t_l^*) \quad (2.10)$$

④ 计算

$$s_i = \begin{cases} r_i & ,if\ c[i]=0 \\ \frac{r_i}{\alpha}\ (mod\ n), otherwise \end{cases} ,\text{for } i = 1,\ldots,l \quad (2.11)$$

可以证明，按照上述步骤构造出来的(c,s)对满足式（2.8）。由于(c,s)对的构造过程中需要使用 α，故能证明签名者知道 α。

(4) **群签名方案**

A. **系统建立**

指定一个群管理员，由群管理员计算如下值：

① 一个 RSA 公钥对(n,e)

② 一个满足群签名方案要求的群$G =< g >$，其阶为 n

③ 元素$a \in Z_n^*$

④ 群成员私钥长度的上界λ和一个大于 1 的常数ϵ

则该群组的公钥是$\varphi = (n,e,G,g,a,\lambda,\epsilon)$.

B. **产生群成员密钥和证书**

当新成员 Alice 想要加入群时，需要进行的步骤如下：

步骤 1. Alice 产生她的私钥$x \in \{0,\ldots,2^{\lambda}-1\}$，并计算$y = a^x (mod\ n)$，以及她的群成员密钥$z = g^y$

步骤 2. 将 y 和 z 发送给群管理员，并利用离散对数的知识签名证明自己知道 y 对底数 a 的离散对数值 x，即自己拥有私钥

当群管理员收到消息之后，利用知识签名验证 Alice 是否具有私钥，如果验证通过则为 Alice 颁发群成员证书：

$$v \equiv (y+1)^{\frac{1}{e}} (mod\ n) \tag{2.12}$$

C. 对消息进行签名

对于待签名的消息 m，需要计算如下值：

$$gg = g^r \text{ for } r \in Z_n^* \tag{2.13}$$

$$zz = gg^y \tag{2.14}$$

$$v_1 = SKLOGLOG[x: zz = \ gg^{(a^x)}](m) \tag{2.15}$$

$$v_2 = SKROOTLOG[v: zz * gg = \ gg^{(v^e)}](m) \tag{2.16}$$

签名的结果即为(gg, zz, v_1, v_2).

D. 验证签名

消息接收者接收到消息和签名后，通过知识签名的验证来证明签名的有效性。注意在知识签名的验证过程中并不需要知道消息发送者的公钥。

E. 打开签名

在正常情况下，给定两个签名(gg, zz, v_1, v_2)和(gg', zz', v_1', v_2')，欲判断二者是否由同一个成员签发，需要确定

$$\log_{gg} zz = \ \log_{gg'} zz' \tag{2.17}$$

是否成立。通常情况下该问题是难解的，故群签名方案可以使得群签名是匿名且不可关联的。而在需要确认消息具体签发者的情况下，由于群管理员知道所有群成员的 y 值，故给定一个签名(gg, zz, v_1, v_2)，群管理员可以通过遍历所有y，判断当前y_i是否满足

$$gg^{y_i} = zz \tag{2.18}$$

若满足，则该成员是消息的签发者。

1.3.3 GDH.2 密钥交换协议

Group Diffie Hellman 协议 Diffie–Hellman 密钥交换协议在有限群上的拓展,用于进行群组密钥交换。GDH.2 是其中的一个变种[9]。假设群成员为P_1, P_2, P_3, P_4，x_i是P_i的私钥，则群密钥产生的方式如下：

P_1将$\{g^{x_1}\}$发送给P_2.

P_2将$\{g^{x_1}, g^{x_2}, g^{x_1 x_2}\}$发送给$P_3$.

P_3将$\{g^{x_1 x_2}, g^{x_1 x_3}, g^{x_2 x_3}, g^{x_1 x_2 x_3}\}$发送给$P_4$.

P_4将群密钥设为$g^{x_1 x_2 x_3 x_4}$，并将$\{g^{x_1 x_2 x_4}, g^{x_1 x_3 x_4}, g^{x_2 x_3 x_4}\}$广播给所有群成员。

其他群成员收到广播消息后即可分别计算$g^{x_1 x_2 x_3 x_4}$.

以上方法可能遭到中间人攻击。可以让消息发送者对消息进行签名以证明自己的合法身份。

1.3.4 加密算法

本方案中使用 AES 算法进行加密和解密。

AES 是一高效安全的对称加密算法。AES 加密过程是在一个 4×4 的字节矩阵上运作，其初值就是一个明文区块（矩阵中一个元素大小就是明文区块中的一个 Byte）。加密时，各轮 AES 加密循环（除最后一轮外）均包含以下 4 个步骤[10]：

AddRoundKey—矩阵中的每一个字节都与该次回合密钥（round key）做 XOR 运算；每个子密钥由密钥生成方案产生。

SubBytes—透过一个非线性的替换函数，用查找表的方式把每个字节替换成对应的字节。

ShiftRows—将矩阵中的每个横列进行循环式移位。

MixColumns—为了充分混合矩阵中各个直行的操作。这个步骤使用线性转换来混合每内联的四个字节。最后一个加密循环中省略 MixColumns 步骤，而以另一个 AddRoundKey 取代。

2 详细设计

本章介绍了本系统的详细设计流程。包括了具体的报文交互协议，包括了从系统建立、成员入群、群密钥交换、群成员之间通信等软件流程细节的设计。详细介绍了实现细节，并将关键算法用伪代码的形式呈现出来。

2.1 交互报文协议设计

根据群签名方案与群密钥交换技术，确定了网络发包时的数据类型与发包顺序，为了整个系统网络的规范化表达，依据 linux 网络编程的基本规则[11]，制定了本系统的网络报文协议格式,如下：

阅读说明

设备：指成员 Member

服务器：指运行 Server 端的 GM

表格的每行为 4 字节，按长中短三种形式分别表示为 4、2、1 字节，均为无符号进制：表格中所有数据均为 16 进制

数据格式：报文传输时所有 2/4 字节均为网络序，报文解释时已转为主机序

(1) TCP 设备连接

功能：设备向 Server 段的指定端口发起 TCP 连接

方向：设备=>服务器

(2) 设备向服务器请求公共参数 public-para

- 功能：设备 TCP 连接成功后，设备向服务器发送请求公共参数 public-para 的请求
- 方向：设备=>服务器

0 7 8 15 16

31

0x00	0x01	Len（报文的总长度）

图 3.1 通信报文 1

[0x00]：1 字节，表示发包方向为从设备到 GM 服务器

[0x01]：1 字节，表示请求类型为 1

[Len]：2 字节，表示即将发送的报文的总长度

(3) 服务器向设备发送公共参数 public-para

- 功能：服务器接收到设备的请求后，服务器向设备发送公共参数 public-para
- 方向：服务器=>设备

0 7 8 15 16

31

0x01	0x01	Len（报文的总长度）
public-para		

图 3.2 通信报文 2

[0x01]：1 字节，表示发包方向为从 GM 服务器到设备

[0x01]：1 字节，表示请求类型为 1

[Len]：2 字节，表示即将发送的报文的总长度

[public-para]：公共参数 public-para 转换的字符串

(4) 设备向服务器请求加入群

• 功能：获得公共参数后，设备向服务器发送请求入群的请求

• 方向：设备=>服务器

0 7 8 15 16

31

0x00	0x02	Len（报文的总长度）
y,z,m,sig		

图 3.3 通信报文 3

[0x00]：1 字节，表示发包方向为从设备到 GM 服务器

[0x02]：1 字节，表示请求类型为 2

[Len]：2 字节，表示即将发送的报文的总长度

[y,z,m,sig]：表示由 ZZ 型数据 y,z,m,sig 转换成的字符串，以空格分隔

(5) 服务器向设备发送成员证书 v

• 功能：服务器接收到设备的请求后，服务器向设备发送成员证书 v

• 方向：服务器=>设备

0 7 8 15 16

31

0x01	0x02	Len（报文的总长度）
v		

图 3.4 通信报文 4

[0x01]：1 字节，表示发包方向为从 GM 服务器到设备

[0x02]：1 字节，表示请求类型为 2

[Len]：2 字节，表示即将发送的报文的总长度

[v]：经 GM 验证并计算生成的群成员证书 v 转换的字符串

(6) 服务器向设备发送密钥

• 功能：服务器向设备发送密钥链

• 方向：服务器=>设备

0 7 8 15 16

31

0x01	0x03	Len（报文的总长度）
keyChain(g^n…)		

图 3.5 通信报文 5

[0x01]：1 字节，表示发包方向为从 GM 服务器到设备

[0x03]：1 字节，表示请求类型为 3

[Len]：2 字节，表示即将发送的报文的总长度

[keyChain(g^n…)]：密钥链 keyChain 转换的字符串

(7) 设备向服务器发送接收密钥链应答

• 功能：获得 keyChain 后，设备向服务器发送接收密钥的应答

• 方向：设备=>服务器

0 7 8 15 16

31

0x00	0x03	Len（报文的总长度）
onRequestMsg(g^n1，g^n2，…)		

图 3.6 通信报文 6

[0x00]：1 字节，表示发包方向为从设备到 GM 服务器

[0x03]：1 字节，表示请求类型为 3

[Len]：2 字节，表示即将发送的报文的总长度

[onRequestMsg(g^n1，g^n2，…)]：表示由回复的信息 onRequestMsg 转换成的字符串

(8) 服务器向设备发送广播

• 功能：通过 UDP 连接，服务器向设备群广播

• 方向：服务器=>设备群

0 7 8 15 16

31

0x01	0x04	Len（报文的总长度）
broadcastMsg		

图 3.7 通信报文 7

[0x01]：1 字节，表示发包方向为从 GM 服务器到设备

[0x04]：1 字节，表示请求类型为 4

[Len]：2 字节，表示即将发送的报文的总长度

[broadcastMsg]：即将广播的字符串

(9) 设备向服务器请求转发发送信息

• 功能：设备向服务器请求转发信息

• 方向：设备=>服务器

0 7 8 15 16

31

0x00	0x05	Len（报文的总长度）
from		
to		
commuMsg		

图 3.8 通信报文 8

[0x00]：1 字节，表示发包方向为从设备到 GM 服务器

[0x05]：1 字节，表示请求类型为 5

[Len]：2 字节，表示即将发送的报文的总长度

[from]：16 字节，表示发送者 id

[to]：16 字节，表示接受者 id

[commuMsg]：表示要转发的字符串

(10) 服务器转发设备信息

• 功能：服务器向设备正常发送信息

• 方向：服务器=>设备

0 7 8 15 16

31

0x00	0x05	Len（报文的总长度）
from		
to		
commuMsg		

图 3.9 通信报文 9

[0x01]：1 字节，表示发包方向为从 GM 服务器到设备

[0x05]：1 字节，表示请求类型为 5

[Len]：2 字节，表示即将发送的报文的总长度

[from]：16 字节，表示发送者 id

[to]：16 字节，表示接受者 id

[commuMsg]：表示要转发的字符串

2.2 软件流程

软件流程示意图如图 3.10 所示：

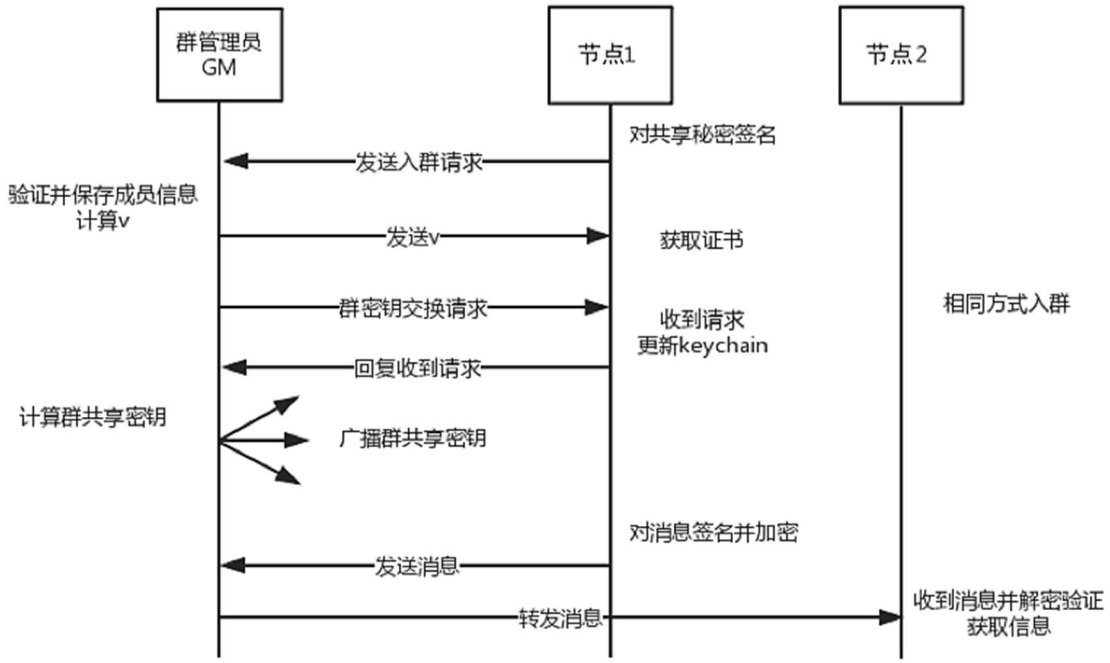

图 3.10 软件流程示意图

2.2.1 系统建立

初始时，选定群组中的一个节点作为群管理员。然后生成一个 512 位的素数 G 用于创建交换群Z_G^*. 令g为该群的生成元。n为该群的阶，即$n = G - 1$. 然后从Z_n^*中随机选择一个数作为 RSA 算法的 rsa_b，其对 n 的逆作为 RSA 算法的 rsa_a。最后从Z_n^*中随机选取一个数作为系统参数a，指定一个数作为私钥长度的上界λ，并选取一个大于 1 的常数作为系统参数 ϵ. 则群的公开参数为$(n, rsa_b, G, g, a, \lambda, \epsilon)$.

2.2.2 成员入群

对于新入群的成员 Alice，其首先向群管理员发送获取公共参数请求。获得公共参数后，创建一个新的 Member 对象。每一个群成员用一个硬件 id 标识，硬件 id 具有唯一性。群成员随机生成一个λ位的正整数作为其私钥x.

然后 Alice 向群管理员发起加入群组的申请，入群消息的生成流程：

表 3.1 JoinGroupMsg 算法

算法　入群消息生成算法 JoinGroupMsg
输入：共享秘密psk，私钥x，群参数a
输出：入群申请消息msg
$y = a^x (mod\ n)$

续表 3.1

算法　入群消息生成算法 JoinGroupMsg
$z = g^y (mod\ n)$
$p = SKLOG(psk, y, a, x)$
$msg = y \| z \| y \| a \| p.c \| p.s$
RETURN msg

在上述算法中，第 1-2 行生成群成员参数 y 和 z，第 3 行获取知识签名的结果，第 4 行生成最终的签名结果。调用上述算法时，提供的共享秘密 psk 是在协议运行之前就由所有群成员共享的，只有拥有 psk 者才有资格入群，且无需将签名的消息本身发送给群管理员。故没有 psk 者将无法通过群管理员的验证。

以上算法中生成的 y 和 z 需要作为 Member 的成员变量保存。由于 y 可以用于确认发送者的身份，故一般情况下需要将入群申请消息使用 RSA 算法利用 RSA 公钥 rsa_b 进行加密，群管理员收到消息后利用 RSA 私钥 rsa_a 进行解密。

上述算法中使用的$SKLOG$知识签名算法的流程：

表 3.2 SKLOG 算法

算法　离散对数知识签名算法 SKLOG

输入：待签名的消息m，群中元素y，底数g，对数值x，群参数n

输出：(c,s)对p

$ax = a^{\alpha}(mod\ n)$

$concatStr = (m\|y\|g)$

$r = getRandInZn()$

$concatStr += g^{r}(mod\ n)$

$p.c = H(concatStr)$

$p.s = r - p.c * x\ (mod\ n)$

RETURN p

在上述算法中，第 1-4 行生成用于计算 c 的串，第 5-6 行生成了(c,s)对。其中，$getRandInZn$函数用于在Z_n^*中生成一个随机数。

群管理员在收到入群申请消息后，需要验证 y 和 z 的合法性，即$z = g^{y}(mod\ n)$是否成立。然后通过知识签名在验证 Alice 拥有私钥 x 的同时确认 Alice 知道共享秘密 psk。否则如果验证不通过则无法通过群成员的入群请求。

其中，知识签名验证的算法：

表 3.2 SKLOGver 算法

算法　离散对数知识签名验证算法 SKLOGver

输入：待验证的消息m，群中元素y，底数g，群参数n，(c,s)对p

输出：签名是否有效

$temp = g^{p.s} * y^{p.c}(mod\ n)$

$concatStr = (m\|y\|g\|temp)$

IF $H(concatStr) == p.c$

　　RETURN true

Else

　　RETURN false

在上述算法中，第 1-2 行计算用于生成比对数的串。3-6 行验证知识签名是否有效，即验证群成员是否知道私钥。验证通过之后，群管理员计算$v \equiv (y+1)^{\frac{1}{e}}\ (mod\ n)$并将其发送给 Alice。

2.2.3 群密钥交换

与上文中 GDH.2 方案的原始方法不同，当新成员加入群后，由群管理员主动发起群密钥的更新，密钥交换信息也是在群管理员和群成员之间传输。使用这种方法，使得群成员无需知道其他群成员的具体信息，而统一交给群管理员进行中转，更加便捷。首先，群管理员向该新成员发送群密钥交换请求，将当前已有的密钥链 keyChain 发送给新成员。群成员收到请求后，用自己的私钥更新 keyChain，并将 keyChain 发回给群管理员。群管理员根据收

到的 keyChain 消息结合 rsa 私钥设置群密钥，并且将 keyChain 的各部分发送给相应的群成员。为了防止中间人攻击，群成员在向群管理员回消息时将消息进行群签名。密钥交换过程中使用了多个算法。算法流程依次如下：

在收到了群密钥交换请求之后，群成员运行算法：

表 3.3 onKeyExchangeRequestRecv 算法

算法　群密钥交换算法-1 onKeyExchangeRequestRecv
输入：密钥链消息msg**，群参数**g**，群参数**n**，私钥**x
输出：新的密钥链消息$keyChainStr$
$gnBuf = split(msg)$
IF $gnBuf.empty()$
$gnOut.append(g^x(mod\ n))$
ELIF $gnBuf.size == 1$
$gnOut.append(gnBuf[0])$
$gnOut.append(g^x(mod\ n))$

续表 3.1

算法　群密钥交换算法-1 onKeyExchangeRequestRecv
$gnOut.append(gnout[0]^x(mod\ n))$
ELSE
$gnOut.append(gnBuf.rbegin())$
FOR i **in** $gnBuf$
$gnOut.append(i^x(mod\ n))$
$keyChainStr = str(gnOut)$
RETURN $keyChainStr$

在上述算法中，第 1 行解析出密钥链，第 2-11 行根据 GDH.2 算法生成新的密钥链，第 12-13 行返回密钥链信息。其中，密钥链生成的基本原理是：保留当前密钥链的最后一个值，然后将当前密钥链的每个值都作该成员的私钥的幂，加入新的密钥链中。

在收到群成员的密钥交换消息之后，群管理员运行如下算法生成群密钥：

表 3.4 onKeyExchangeResponseRecv 算法

算法　群密钥交换算法-2 onKeyExchangeResponseRecv
输入：新的密钥链消息msg**，RSA 私钥**rsa_a
输出：无
$keyChain = split(msg)$
$groupKey = keyChain.rbegin()^{rsa\_a}(mod\ n)$

在上述算法中，第 1 行得到新的密钥链信息。第 2 行根据$g^{n_1 \cdots n_{k-1}}$再作 RSA 私钥的幂得到新的群密钥。该算法中的$keyChain$和$groupKey$均作为群管理员的成员变量保存下来。

然后，群管理员需要将群密钥消息发送给各群成员，运行算法：

表 3.5 getBroadcastMsg 算法

算法　群密钥交换算法-3 getBroadcastMsg

输入：密钥链$keyChain$**，群成员信息**$info$**，RSA 私钥**rsa_a

输出：群密钥更新信息msg

FOR k **in** $keyChain$ **&&** i **in** $reverse(info)$

　　$msg += i \rightarrow id$

　　$msg += k^{rsa\_a} (mod\ n)$

RETURN msg

在上述算法中，第 1-3 行生成各个群成员对应的密钥广播信息，使得每个成员对应的信息再作自己的私钥的幂就可以得到群密钥。

群成员在收到广播的消息后，需要根据密钥链来更新算法，运行 onGroupKeyBoardcastRecv 算法更新群密钥：

表 3.6 onGroupKeyBoardcastRecv 算法

算法　群密钥交换算法-4 onGroupKeyBoardcastRecv

输入：群密钥更新信息msg**，群成员的身份信息**id**，私钥**x

输出：无

WHILE $curId = split(msg)$ && $gn = split(msg)$

　　IF $curId == id$

　　$groupKey = gn^x (mod\ n)$

　　RETURN

在上述算法中，第 1-4 行中群成员遍历广播消息寻找到自己对应的消息，然后将该消息作自己私钥x的幂就得到了群密钥，并保存下来用以加密和解密。

2.2.4 群成员间通信

当群成员之间需要进行通信时，消息发送者先对消息进行群签名，然后使用 AES 算法，利用群密钥对消息及签名进行对称加密。群成员之间的消息传输通过群管理员进行中转，发送者将消息发给群管理员，由群管理员将消息转发给接收者。消息接收者使用 AES 算法利用群密钥对消息进行解密，然后验证群签名的合法性，确认消息发送者是合法的群成员，获取消息的内容。

如果遇到需要追查消息确切发送者的情况，则由群管理员根据其保存的所有群成员的 y 值，逐一验证$gg^y = zz$是否成立，从而确定消息的签发者。

（1） 消息签名的算法流程

表 3.7 sig 算法

算法 消息签名算法 sig

输入：待签名的消息m**，群的参数**$para$**，私钥**x**，成员变量**y

输出：对消息的签名$result$

$gg = g^r$ for $r \in Z_n^*$

$zz = gg^y$

$v_1 = SKLOGLOG(m, zz, gg, para \rightarrow a, x)$

$result = gg \| zz \| y \| v_1.c \| v_1.cnt \| v_1.s$

RETURN $result$

在上述算法中，第 1-2 行计算$SKLOGLOG$函数所需要的参数，第 3-4 行调用$SKLOGLOG$函数并得到签名。$SKLOGLOG$算法用于证明消息发送者拥有私钥x. 与原 CS97 方案的实现不同，由于消息需要经过加密，而只有群成员才能拥有群密钥，因此能以此验证消息发送者是否是群成员，而无需再使用$SKROOTLOG$算法。其中，$SKLOGLOG$算法流程：

表 3.8 SKLOGLOG 算法

算法 双离散对数知识签名算法 SKLOGLOG

输入：待签名的消息m**，群中元素**y**，底数**g**，底数**a**，对数值 α，群参数**n, ϵ, λ

输出：(c, s)**对**p

$ax = a^{\alpha} (mod\ n)$

$concatStr = (m \| y \| g \| a)$

FOR $i = 1, \dots, l$

$r[i] = findRandInLamda(\epsilon\lambda, \alpha)$

$ar = a^{r[i]} (mod\ n)$

$t[i] = g^{ar} (mod\ n)$

$concatStr += t[i]$

$p.c = H(concatStr)$

FOR $i = 1, \dots, l$

IF $p.c[i] == 0$

$p.s[i] = r[i]$

ELSE

$p.s[i] = r[i] - \alpha$

RETURN p

在上述算法中，第 1-8 行用于生成c，第 9-13 行用于生成s。其中$findRandInLamda$函数用于用于生成一个位数上界为$\epsilon\lambda$且大于α的随机数。

（2） 消息验证算法流程

表 3.9 ver 算法

算法 消息验证算法 ver
输入：解密后的消息msg**，消息的签名**sig**，群参数**a
输出：签名是否正确
$gg = split(sig)$
$zz = split(sig)$

续表 3.9

算法 消息验证算法 ver
$y = split(sig)$
$v_1 = split(sig)$
IF $SKLOGLOGver(msg, zz, gg, y, a, v1)$
RETURN true
Else
RETURN false

在上述算法中，第 1-4 行拆分出各个参数，第 5-8 行验证签名是否正确并返回。

其中，$SKLOGLOGver$算法的流程：

表 3.10 SKLOGLOGver 算法

算法 双离散对数知识签名验证算法 SKLOGLOGver
输入：待签名的消息m**，群中元素**y**，底数**g**，参数**ax**，底数**a**，**(c, s)**对**p**，群参数**n
输出：签名是否有效
$concatStr = (m \| y \| g \| a)$
FOR i **in** $p.s.size()$
$as = a^{p.s[i]} (mod\ n)$
$exp = ax * as (mod\ n)$
$t = p.c[i] == 0 ? g^{as} (mod\ n) : g^{exp} (mod\ n)$
$cocatStr += t$
IF $H(concatStr) == p.c$
RETURN true
Else
RETURN false

在上述算法中，第 1-6 行计算用于生成比对数的串。第 7-10 行验证知识签名是否有效，即验证消息发送者是否知道私钥。

（3） 打开签名算法的流程

表 3.11open 算法

算法　签名打开算法 open
输入：签名组成部分gg，签名组成部分zz，群成员信息$info$
输出：消息的签发者

续表 3.11

算法　签名打开算法 open
FOR i **in** $info$
IF $gg^{i.y}(mod\ n) == zz$
RETURN $i.id$

在上述算法中，群管理员遍历其存储的群成员信息，依次判断其是否是消息的真实签发者。