doi:10.19665/j.issn1001-2400.2022.01.008

密码累加器研究进展及应用

苗美霞，武盼汝，王赟玲
（西安邮电大学 网络空间安全学院，陕西 西安 710121）

摘要：密码累加器能够将集合中的所有元素进行累加，并高效地给出任意元素的(非) 成员证明，即该元素是否存在于集合中。密码累加器主要分为静态累加器、动态累加器以及通用累加器三种类型。静态累加器针对静态集合中元素的累加；动态累加器进一步允许从累加集合中动态地添加和删除元素；通用累加器能够同时支持成员证明和非成员证明(元素不在集合中)。针对上述不同类型的密码累加器，许多学者基于不同的密码工具给出了具体构造，可分为基于 RSA 的密码累加器、基于双线性映射的密码累加器和基于 Merkle 哈希树的密码累加器。密码累加器有广泛的应用场景，如群签名、环签名、匿名凭证、时间戳、外包数据验证等。近年来，密码累加器开始应用于区块链中用来解决存储开销大的问题。文中首先从密码累加器的构造方案和功能应用等方面对现有方案进行了分类、分析、总结，其次介绍了密码累加器的主要应用场景，最后指出了现有方案面临的一些问题，以及未来的发展趋势和研究方向。

关键词：密码累加器；RSA 累加器；双线性映射累加器；基于 Merkle 哈希树的累加器
中图分类号：TP309.2 **文献标识码**：A **文章编号**：1001-2400(2022)01-0078-14

Research progress and applications of cryptographic accumulators

MIAO Meixia, WU Panru, WANG Yunling
(School of Cyberspace Security, Xi'an University of Posts and Telecommunications, Xi'an 710121, China)

Abstract: Cryptographic accumulators can accumulate all the elements in a set and efficiently give the (non)membership proof of any element, that is, to prove whether an element exists in the set. Cryptographic accumulators are mainly divided into three types: static accumulators, dynamic accumulators and universal accumulators. Specifically, static accumulators aim at accumulating the elements in the static set; dynamic accumulators further allow the dynamic addition and deletion of elements from the accumulation set; universal accumulators support both membership proof and non-membership proof (elements are not in the set). For the above different types of cryptographic accumulators, many scholars have given specific structures based on different cryptographic tools, which can be divided into RSA based cryptographic accumulator, bilinear mapping based cryptographic accumulator and Merkle hash tree based cryptographic accumulator. Cryptographic accumulators have a wide range of application scenarios, such as group signature, ring signature, anonymous certificate, timestamp, outsourced data verification and so on. In recent years, cryptographic accumulators have been applied to the blockchain to solve the problem of high storage overhead. This paper first classifies, analyzes and summarizes the existing scheme from the aspects of the construction scheme and function application of the cryptographic accumulators, then introduces the main application scenarios of the cryptographic accumulators, and finally points out some problems faced by the

收稿日期：2021-05-13 **网络出版时间**：2021-09-13
基金项目：国家自然科学基金青年科学基金(61902315)
作者简介：苗美霞(1980—)，女，讲师，博士，E-mail：miaofeng415@163.com
武盼汝(1996—)，女，西安邮电大学硕士研究生，E-mail：18734444225@163.com
王赟玲(1990—)，女，讲师，博士，E-mail：ylwang0304@163.com
网络出版地址：https://kns.cnki.net/kcms/detail/61.1076.TN.20210910.1404.002.html

existing scheme, as well as the future development trend and research direction.

Key Words: cryptographicaccumulator; RSA accumulator; accumulator based on bilinear mapping; accumulator based on Merkle hash tree

密码累加器能够高效地证明元素是否存在于集合中。具体来讲,首先将集合 $X=\{x_1,\cdots,x_n\}$ 中的所有元素累加到累加器 acc_X 中,然后计算元素 $x_i\in X$ 的证据 w_i,最后利用证据 w_i 和累加值 acc_X 来证明元素 $x_i\in X$。密码累加器与向量承诺[1-2]、零知识集合(ZK-sets)[3-4]等原语有紧密的联系,三者都能解决(非)成员验证的问题。但是,三者在验证内容、隐私性等方面存在一定的区别。向量承诺[1-2]针对有序集合(向量)提供验证,即不仅能够证明元素 m_i 存在于有序集合中,而且能证明 m_i 是第 i 个位置上的消息,但是其计算复杂性较大。零知识集合[3-4]不仅能够证明某个元素是否属于集合,同时不泄漏任何关于集合的信息,如集合的大小。然而,零知识集合的验证开销与集合中元素的数量线性相关,效率较低。

BENALOH 等[5]于 1993 年首次提出密码累加器的概念,但是该构造为静态累加器,即累加集合固定不变。CAMENISCH 等[6]于 2002 年提出了动态累加器的概念,可以支持累加集合动态地添加和删除元素。然而,上述两类累加器仅支持集合中元素的成员证明,即 $x_i\in X$,而无法支持非成员证明,即无法提供 $y\notin X$ 的证据。为了解决此问题,LI 等[7]于 2007 年首次提出了通用累加器的构造,能够同时实现成员和非成员证明。

密码累加器自提出以来,许多学者对其进行了大量的研究并给出了具体的构造方案。基于不同的密码工具,密码累加器可分为基于 RSA 体制的累加器[5-8]、基于双线性映射的累加器[9-12]和基于 Merkle 哈希树的累加器[13]。密码累加器有着广泛的应用场景。在访问控制系统中,将授权用户的访问权限聚合到累加器中,授权用户通过将成员证据作为访问凭证来访问系统。在匿名凭证系统中[6],需要进一步隐藏用户的身份,将累加器和零知识证明[14]结合是解决这类问题的有效方法。在数据外包存储中,密码累加器可用作认证数据结构(Authenticated Data Structure, ADS)[15-18],用户利用计算结果和相应证据来证明计算结果的正确性。在加密货币中,密码累加器通过代替 Merkle 哈希树来降低通信开销和提高验证效率[19]。此外,累加器还可应用于时间戳[5]、隐私保护数据外包[20]、认证字典[21]、编辑[22]、负责任的证书管理[23-24]等场景中。

笔者首先从密码累加器的构造方案和功能应用等方面对现有方案进行了分类、分析和总结;其次介绍了密码累加器的主要应用场景;最后指出了现有方案面临的一些问题以及未来的发展趋势和研究方向。

1　累加器模型与定义

1.1　系统模型

一般来讲,一个密码累加器主要包括以下 3 个主体。

(1) 累加器管理员:生成密钥对,创建并发布累加器。此外,动态累加器可以添加和删除元素,并创建这些元素的成员证据。通用累加器可以对没有被聚合到累加器中的元素创建非成员证据。

(2) 用户:接受累加器管理员提供的证据,证据是他们在累加器系统中的凭证,可以提供给验证方进行验证。之后有新的元素添加到累加器时,用户可以在本地更新证据。

(3) 验证方:通过用户提供的证据和累加器公开的累加值,来检查元素 x 是否在此累加器中。

下面分别对静态累加器、动态累加器以及通用累加器进行了定义。用 X 表示累加器的累加值域,用 $X=\{x_1,\cdots,x_n\}$ 表示累加器中元素的集合,用 t 表示累加元素的上限,t 可以是无穷的,表示累加域无界。此外,陷门信息(即私钥 $\mathrm{sk}_{\mathrm{acc}}$)是可选择的输入,因为有些累加器需要陷门信息才能更新累加器或证据,而有些不需要。

定义 1　一个静态累加器方案可以描述成一个四元组 $\Pi=(\mathrm{Gen},\mathrm{Eval},\mathrm{WitCreate},\mathrm{Ver})$。

(1) $\mathrm{Gen}(1^\lambda,t)$:累加器管理员输入安全参数 λ、累加元素上限 t,输出密钥对 $(\mathrm{sk}_{\mathrm{acc}},\mathrm{pk}_{\mathrm{acc}})$,如果累加器不存在陷门信息,则陷门信息 $\mathrm{sk}_{\mathrm{acc}}=\varphi$。

(2) $\mathrm{Eval}((\mathrm{sk}_{\mathrm{acc}},\mathrm{pk}_{\mathrm{acc}}),X)$:累加器管理员计算累加值和辅助信息。输入集合、密钥对 $(\mathrm{sk}_{\mathrm{acc}},\mathrm{pk}_{\mathrm{acc}})$,输出累加值 acc_X,并将其公布给用户和验证方;输出辅助信息 aux,并发送给用户,使得用户可以在本地更新

证据。

(3) WitCreate((sk_{acc}, pk_{acc}), x_i, acc_X, aux):累加器管理员创建用户 x_i 的证据。输入密钥对(sk_{acc}, pk_{acc})、累加值 acc_X、元素 $x_i \in X$ 及辅助信息 aux,生成 x_i 的证据 w_i。

(4) Ver(pk_{acc}, x_i, w_i, acc_X):验证方验证元素是否在累加器中。输入公钥 pk_{acc}、累加值 acc_X、元素 x_i 及其证据 w_i。如果 w_i 是 $x_i \in X$ 的证据,则返回 1;否则,返回 0。

在此基础上,如果累加器可以动态地更新集合 X,并可以有效地更新集合中元素的证据,那么该累加器为动态累加器。通过对静态累加器定义扩展,可以得到一个动态累加器。

定义 2 动态累加器在静态累加器的基础上添加了一个三元组 $\Pi=$(Add,Del,MemWitUp)。

(1) Add((sk_{acc}, pk_{acc}), acc_X, aux, x):累加器管理员添加元素 $x \notin X$ 到累加器中,并更新累加值 acc_X。输入密钥对(sk_{acc}, pk_{acc})、累加值 acc_X、辅助信息 aux 和要添加的元素 $x \notin X$,输出更新后的累加值 $acc_{X'} = acc_{X \cup \{x\}}$,并更新辅助信息 aux。

(2) Del((sk_{acc}, pk_{acc}), acc_X, x, aux):累加器管理员删除元素 $x \in X$ 时,更新累加值 acc_X。输入密钥对(sk_{acc}, pk_{acc})、累加值 acc_X、辅助信息 aux 和被删除的元素 $x \in X$,输出更新后的累加值 $acc_{X'} = acc_{X \setminus \{x\}}$,并更新辅助信息 aux。

(3) MemWitUp((sk_{acc}, pk_{acc}), x, w_i, aux):添加或删除元素 x 后,用户更新元素 x_i 的证据 w_i。输入密钥对(sk_{acc}, pk_{acc})、x、辅助信息 aux 和 x_i 的证据 w_i,输出 x_i 更新后的证据 w_i'。

上述累加器可以对元素 $x \in X$ 构造成员证据,但无法对元素 $x \notin X$ 构造非成员证据。而通用累加器既可以对元素 $x \in X$ 构造成员证据,又可以对元素 $x \notin X$ 构造非成员证据。

定义 3 通用累加器在上述 WitCreate 算法中,可以添加一个布尔参数 type,创建成员证据时 type=0,创建非成员证据时 type=1。如果通用累加器不具备动态累加器添加或删除元素的功能,则为通用静态累加器,否则为通用动态累加器。对于通用动态累加器,则需要在定义 1 和定义 2 的基础上添加 NonMemWitUp 算法,可以在累加集合更新时,对非成员证据进行更新。

(1) NonMemWitUp((sk_{acc}, pk_{acc}), x, w, aux):在添加或删除某个元素 x 时,用户更新非成员元素 $y \notin X$ 的证据 u 为 u'。输入密钥对(sk_{acc}, pk_{acc})、x、辅助信息 aux 和 y 的证据 u,输出 y 更新后的证据 u'。

1.2 累加器的性质

累加器有 3 个安全性质:正确性、健壮性(包括无碰撞性和不可否认性)、不可区分性。3 个安全性质定义如下:

(1) 正确性:对于所有诚实生成的密钥、所有诚实计算的累加值和证据,验证算法 Ver 将始终返回 1。

(2) 健壮性:传统上,健壮性指无碰撞性,对于元素 $y \notin X$,很难找到其成员证据[8],对于元素 $x_i \in X$,也很难找到其非成员证据[7];健壮性的另一种扩展形式是不可否认性[23],不可否认性是通用累加器特有的性质,它表示为元素 $x \in X$ 或 $x \notin X$ 计算两个相互矛盾的证据在计算上是不可能的。

(3) 不可区分性:不可区分性[25-26]与安全隐私相关,指累加器和持有证据的用户都不会泄露有关累加集合 X 的信息。

分别对静态累加器、动态累加器和通用累加器进行形式化定义。具体如下:

(1) 安全性(静态):一个静态累加器是安全的,如果对于所有概率多项式时间(Probabilistic Polynomial Time,PPT)对手 A_λ,有

$$\Pr\left[\begin{array}{l}((sk_{acc}, pk_{acc}), f) \leftarrow G(1^\lambda); g \leftarrow_R Z_A; \\ (x, w_x) \leftarrow A_\lambda(f, Z_A, g) \leftrightarrow M(f, pk_{acc}, g) \rightarrow (X, acc) \\ x \in X; w_x \subset X; x \notin X; f(w_x, x) = f(g, X)\end{array}\right] = neg(\lambda) \quad 。$$

(2) 安全性(动态):令 M 为接收输入(f, aux, g) 的交互式图灵机,其中,aux 是有关 f 的辅助信息,$g \in Z_A$。M 维护一个元素集合 X,该集合最初为空,初始累加器 acc 设置为 g。M 响应两种消息:对于消息 D(add, x),它确保 $x \in X$,将 x 添加到集合 X 中,通过运行 D 修改 acc,然后将更新后的 acc 发回;对于消息 D(delete, x),它将检查 $x \in X$,将其从集合 X 中删除,通过运行 D 更新 acc,然后将更新后的 acc 发回。最后,M 输出 X 和 acc 的当前值。如果对于所有 PPT 的对手 A_λ,动态通用累加器方案是安全的,则有

$$\Pr\left[\begin{array}{l} f \leftarrow G(1^\lambda); g \leftarrow_R Z_A; \\ (x, w_x, X) \leftarrow A_\lambda(f, Z_A, g) \leftrightarrow M(f, \text{aux}, g) \rightarrow (X, \text{acc}): \\ x \in X; X \subset X; \text{acc} = f(g, X); \rho_1(\text{acc}, x, w_x) = 1 \end{array}\right] = \text{neg}(\lambda) \quad 。$$

(3) 安全性(通用):一个通用累加器是安全的,如果对于所有 PPT 对手 A_λ,有

$$\Pr\left[\begin{array}{l} f \leftarrow G(1^\lambda); g \leftarrow_R Z_A; (x, \text{wit}_1, \text{wit}_2, X) \leftarrow A_\lambda(f, Z_A, g): \\ x \in X; X \subset X; \rho_1(f(g, X), x, \text{wit}_1) = 1; \rho_2(f(g, X), x, \text{wit}_2) = 1 \end{array}\right] = \text{neg}(\lambda) \quad 。$$

2　基于 RSA 体制的累加器

1993 年,BENALOH 等[5]首次提出 RSA 累加器。之后,BARIĆ等[8]完善了该方案中的安全性定义,介绍了无碰撞累加器的概念,即对于元素 $y \notin X$ 找到成员证据在计算上是不可能的。在此基础上,对累加器进行功能扩展的方案是 CAMENISCH 等[6]提出的动态累加器和 LI[7]等提出的通用累加器。上述经典的静态、动态和通用累加器方案的构造都需要陷门信息。SANDER[27]首次提出了使用未知分解的 RSA 模数来构造无陷门累加器,LIPMAA[28]将 RSA 累加器推广到欧几里得环模上,也提出了无陷门累加器的构造。上述所有方案,累加器都被限制为素数,以确保无碰撞。TSUDIK 等[29]提出了动态累加器的变体,该方案允许累加任意整数,并支持证据的批量更新。然而,该方案被 CAMACHO 等[30]证明不可行。RSA 累加器批处理(非)成员证明非常简单,只需将被证明的元素相乘作为一个值 x^*,并为它们提供一个共同的证据 w,即可一次对多个元素进行验证,但通信成本与 x^* 大小成正比。为了解决这个问题,2019 年,BONEH 等[19]提出了批处理(非)成员证据的证明技术(Proof of Exponetiation,PoE)来解决验证效率的问题。

2.1　静态累加器

BENALOH 等[5]首次提出的累加器是 RSA 静态累加器。该方案将有限集合 $X=\{x_1,\cdots,x_n\}$ 累加为一个简洁的值 acc_X。对于每个元素 $x_i \in X$,都能够有效地计算出其证据 w_i,来证明元素 $x_i \in X$。BARIĆ等[8]在上述方案的基础上,提出了累加器的无碰撞性,并基于强 RSA 假设给出了无碰撞累加器的构造。该方案的具体构造如下:

(1) $\text{Gen}(1^\lambda)$:累加器管理员输入安全参数 λ,生成累加器初始值 $g \in Z_n$,输出密钥对 $(\text{sk}_{\text{acc}}, \text{pk}_{\text{acc}})=((p, q), N)$,其中 N 由两个大的强素数 p、q 组成,且 $N=pq$。

(2) $\text{Eval}(\text{pk}_{\text{acc}}, X)$:累加器管理员计算累加值。累加值 $\text{acc}_X = g^{x_1,\cdots,x_n} \bmod N$,其中 $X=\{x_1,\cdots,x_n\}$,且 x_i 是素数。

(3) $\text{WitCreate}(\text{pk}_{\text{acc}}, x_i, \text{acc}_X)$:累加器管理员创建用户的证据。输入公钥 pk_{acc}、累加值 acc_X 和元素 x_i,生成 x_i 的证据 $w_i = g^{x_1,\cdots,x_{i-1},x_{i+1},\cdots,x_n} \bmod N$。

(4) $\text{VerMem}(\text{pk}_{\text{acc}}, x_i, w_i, \text{acc}_X)$:验证方通过判断等式 $\text{acc}_X = w_i^{x_i} \bmod N$ 是否成立来验证元素 x_i 是否在累加器中。如果验证成功,返回 1,否则返回 0。

值得注意的是,RSA 累加器在计算过程中需要进行模幂运算,集合 X 中的元素在指数中是乘法关系,所以上述构造的元素必须限制为素数,如果聚合的是非素数,则可能存在某个值是另 2 个值的乘积,进而敌手可以很容易伪造证据进行验证。

2.2　动态累加器

在一些需要添加或撤销成员的应用场景中,如在群签名、匿名凭证撤销系统中,群管理员需要动态添加或撤销成员资格。然而,上述静态累加器的集合元素固定,不具备添加或删除的功能。因此,CAMENISCH 等[6]为了解决匿名凭证系统的撤销等问题,将累加器扩展为动态。动态累加器允许累加器管理员动态地添加或删除元素,并且可以动态地更新累加器和成员证据。在添加元素时,更新累加器、成员证据只需要进行简单的模指运算,可以很容易实现;而在删除元素时,CAMENISCH 等[6]采用了 RSA 算法的基本思想来更新累加器,通过使用陷门信息 sk_{acc} 进行求逆运算,可以将元素从累加器中删除。删除元素时需要用陷门信息更新累加值,但更新成员证据时无需陷门信息。方案具体如下:

(1) Gen(1^λ)：累加器管理员生成公私钥对，初始空累加值 $\mathrm{acc}_0=g\in Z_N$。输入安全参数 1^λ，输出累加器管理员公私钥对$(\mathrm{sk}_{\mathrm{acc}},\mathrm{pk}_{\mathrm{acc}})=((p,q),N)$，其中 $N=pq$，且 p、q 为大的强素数。

(2) Add($\mathrm{acc}_X,x,X,\mathrm{pk}_{\mathrm{acc}}$)：累加器管理员添加元素 x 后更新累加值 acc_X。添加元素 $x\notin X$ 到累加器中，X 更新为 $X\bigcup\{x\}$，更新后的累加值为

$$\mathrm{acc}_{X'}=\mathrm{acc}_{X\bigcup\{x\}}=\mathrm{acc}_X^x \bmod N \quad 。$$

(3) Del($\mathrm{acc}_X,x,X,(\mathrm{sk}_{\mathrm{acc}},\mathrm{pk}_{\mathrm{acc}})$)：累加器管理员删除元素 x 后更新累加值 acc_X。删除元素 x 时，需要陷门信息 $\mathrm{sk}_{\mathrm{acc}}$，通过计算的累加值为

$$\mathrm{acc}_{X'}=\mathrm{acc}_{X\setminus\{x\}}=\mathrm{acc}_X^{x^{-1}\bmod(p-1)(q-1)} \bmod N \quad 。$$

(4) VerMem($\mathrm{acc}_X,x_i,w_i,\mathrm{pk}_{\mathrm{acc}}$)：验证方通过等式 $\mathrm{acc}_X=w_i^{x_i} \bmod N$ 验证元素 x_i 是否在累加器中。如果在，输出 1，否则输出 0。

(5) MemWitUp($x,w_i,x_i,\mathrm{pk}_{\mathrm{acc}},\mathrm{acc}_X,\mathrm{acc}_{X'}$)：用户更新元素 x_i 的证据。添加元素 $x\notin X$ 到累加器时，用户需要更新 x_i 的证据 $w_i'=w_i^x \bmod N$；删除元素 $x\neq x_i\in X$ 时，更新 x_i 的证据通过：首先用扩展欧几里得算法计算 $a,b\in Z$，使得 $ax_i+bx=1$；然后从 x_i 的旧证据 w_i 计算新证据：

$$w_i'=W(w_i,x_i,x,\mathrm{acc}_X,\mathrm{acc}_{X'})=w_i^b\mathrm{acc}_{X'}^a \quad 。$$

值得注意的是，执行删除操作时，累加器管理员需要用到陷门信息 $\mathrm{sk}_{\mathrm{acc}}$ 来更新累加值，此时不诚实的累加器管理员可以通过陷门信息随意更改累加值、对成员创建假的证据等，所以必须要求累加器管理员是可信的。

2.3 通用累加器

动态累加器只能对聚合到累加器中的元素进行成员身份验证，而不能对不在该累加器中的元素进行非成员身份验证。为了解决非成员身份验证的问题，LI 等[7] 提出了通用累加器方案，该方案同时支持成员和非成员身份验证。该方案主要思想为：对于素数集合 $X=\{x_1,\cdots,x_n\}$，让 $x^*=\prod_{i=1}^n x_i$，由于非成员 y_i 与 x^* 互素，使用扩展欧几里得算法可以求得 a、b，使得 $ax^*+by_i=1$，然后可以用 a、b 构造非成员证据。

该方案给出了两种创建非成员证据的方法，一种需要陷门信息 $\mathrm{sk}_{\mathrm{acc}}$，另一种不需要。不需要陷门信息的方法在计算非成员证据时需要找到 a、b，使得 $ax^*+by_i=1$，而 x^* 与累加器成员个数线性相关，计算开销很大。需要陷门信息的方法在计算证据时，可通过 $x^{*'}=x^* \bmod (p-1)(q-1)$，其中 p、q 为陷门信息，使得 $x^{*'}$ 在进行模运算后很小，从而在 $ax^{*'}+by_i=1$ 中，证据的计算开销仅为 $O(1)$。因此在某些可以使用陷门信息的场景下，选择需要陷门信息计算非成员证据的计算量小；而对于某些无法使用陷门信息的场景下，不需要陷门信息计算非成员证据是一个很好的选择。下面给出了两种计算非成员证据的具体构造。

2.3.1 需要陷门信息

使用 $\mathrm{sk}_{\mathrm{acc}}$ 计算非成员证据。假设存在受信任的累加器管理员知道 $\mathrm{sk}_{\mathrm{acc}}=(p,q)$、集合 $X=\{x_1,\cdots,x_n\}$、累加器 $\mathrm{acc}_X=g^{x_1,\cdots,x_n} \bmod N$，其中 $g\in QR_N$。累加器管理员计算元素 $y_i\notin X$ 的非成员证据如下：

(1) 如果 $\gcd(y_i,x^{*'})=1$，其中 $x^{*'}=x^* \bmod (p-1)(q-1)$。累加器管理员需要找到 a、b，使得 $ax^{*'}+by_i=1$，则 y_i 的非成员证据为 $u_i=(a,B)$，其中 $B=g^b \bmod N$。通过检查等式 $\mathrm{acc}_X^a B^{y_i}=g$ 是否成立来进行非成员身份的验证。该处非成员验证等式 $\mathrm{acc}_X^a B^{y_i}=g$ 是正确的，因为 $\mathrm{acc}_X^a B^{y_i}=g^{x^*a}g^{by_i}=g^{x^*a+by_i}=g$。

(2) 如果 $\gcd(y_i,x^{*'})\neq 1$，累加器管理员需要找到 a、b，使得 $ax^*+by_i=1$；然后计算 b'，使得 $b'=b \bmod (p-1)(q-1)$。元素 y_i 的非成员证据为 $u_i=(a,B')$，其中 $B'=g^{b'} \bmod N$，通过检查等式 $\mathrm{acc}_X^a B'^{y}=g$ 是否成立来进行非成员身份验证。该处非成员验证 $\mathrm{acc}_X^a B'^{y_i}=g$ 是正确的，因为 $\mathrm{acc}_X^a B'^{y_i}=g^{x^*a}g^{b'y_i}=g^{x^*a+b'y_i}=g$。

2.3.2 不需要陷门信息

不使用 $\mathrm{sk}_{\mathrm{acc}}$ 计算非成员证据。元素 y_i 的非成员证据为 $u_i=(a,B)$，找到 $a\in Z_{2^l}$，$b\in Z$，使得 $ax^*+by_i=1$。a、b 的计算过程如下：首先使用欧几里得算法找到 a' 和 b'，使得 $a'x^*+b'y_i=1$；然后找到整数 k，使得 $a'+ky_i\in Z_{2^l}$，$(a'+ky_i)x^*+(b'-kx^*)y_i=1$；最后得到 $a=a'+ky_i$，$b=b'-kx^*$，$B=g^b \bmod N$。验证

方验证元素 y_i 为累加器的非成员时:通过检查 $y_i \in X$、$a \in Z_{2^l}$ 和 $\mathrm{acc}_X^a B^{y_i} = g$ 是否成立。该处非成员验证 $\mathrm{acc}_X^a B^{y_i} = g$ 是正确的,因为 $\mathrm{acc}_X^a B^{y_i} = g^{x^* a} g^{by_i} = g^{x^* a + by_i} = g$。

该方案同时支持元素的动态添加和删除,并实现相应证据的高效更新,具体的非成员证据更新算法 NonMemWitUp 如下:

NonMemWitUp(x,(a,B),acc_X,$\mathrm{acc}_{X'}$):

添加:添加元素 $x \in \mathrm{X}\backslash X$,且 $x \neq y_i$,给定更新前后的累加器 acc_X 和 $\mathrm{acc}_{X'}$,则 $y_i \in \mathrm{X}\backslash X$ 的证据 $u_i = (a, B)$ 更新如下:

(1) 找到两个整数 a_0' 和 r_0,使得 $a_0' x + r_0 y_i = 1$,由于 x、y_i 都为素数,所以 a_0'、r_0 可以很容易找到;

(2) 两边同乘更新前的证据 a,使得 $a_0' a x + r_0 a y_i = a$;

(3) 元素 y_i 的证据更新为 $u'_i = (a', B')$:其中 $a' = a_0' a \bmod y_i$;B 的更新通过找到 $r \in Z$,使得 $a' x = a + r y_i$,其中 $a' \in Z_{2^l}$,最后得出 $B' = B \mathrm{acc}_X^{-r} \bmod N$。该处非成员身份验证等式 $\mathrm{acc}_{X'}^{a'} = g B'^{-y_i}$ 是正确的,因为 $\mathrm{acc}_{X'}^{a'} = \mathrm{acc}_X^{a' x} = \mathrm{acc}_X^{a + r y_i} = \mathrm{acc}_X^a \mathrm{acc}_X^{r y_i} = g B^{-y_i} \mathrm{acc}_X^{r y_i} = g (B \mathrm{acc}_X^{-r})^{-y_i} = g B'^{-y_i}$。

删除:删除元素 $x \in X$ 时,元素 $y_i \in \mathrm{X}\backslash X$ 的非成员证据 $u'_i = (a', B')$ 如下:

(1) 选择一个整数 r,使得 $ax - r y_i \in Z_{2^l}$;

(2) 计算集合更新后元素 y_i 的证据 u'_i:其中,$a' = ax - r y_i$,$B' = B \mathrm{acc}_{X'}^r \bmod N$。该处非成员身份验证等式 $\mathrm{acc}_{X'}^{a'} = g B'^{-y_i}$ 是正确的,因为 $\mathrm{acc}_{X'}^{a'} = \mathrm{acc}_{X'}^{ax - r y_i} = \mathrm{acc}_{X'}^a \mathrm{acc}_{X'}^{-r y_i} = g B^{-y_i} \mathrm{acc}_{X'}^{-r y_i} = g (B \mathrm{acc}_{X'}^r)^{-y_i} = g B'^{-y_i}$。

2.4　支持批处理更新的累加器

批处理(Batching)性质是指能够一次性给出多个元素同时在集合中的成员证明(批量成员验证),或者一次性给出多个元素都不在集合中的证明(批量非成员验证)。上述通用累加器方案可以简单扩展实现批量(非)成员的添加和删除。比如当要添加 m 个元素时,只需要将这 m 个元素相乘,得到 $x^* = \prod_i^m = 1 x_i$,此时累加器更新为 $\mathrm{acc}_{X'} = \mathrm{acc}_X^{x^*} \bmod N$。删除元素更新累加器、更新(非)成员证据计算也类似。然而,在批量验证时,通过观察验证等式 $\mathrm{acc}_X = w_{x^*}^{x^*} \bmod N$ 发现,指数 x^* 的大小与要验证的成员元素的个数线性相关,因此验证方的计算开销较大。批处理非成员验证也存在同样问题。为了解决上述问题,2019 年,BONEH 等[19]基于通用累加器方案提出了一种支持(非)成员批处理验证的累加器构造方案,其在进行批处理验证时,验证大小恒定,不随着要验证元素数量的增加而增加,极大降低了计算开销。具体方案如下:

LI 等[7]简要地概述了累加器可以批量添加和删除元素,并未给出具体的算法,而 BONEH 等[19]给出了具体的批量添加(BatchAdd)和批量删除(BatchDel)的算法。在 BatchDel 中,存在一个 ShamirTrick 算法,可以将两个元素的证据聚合在一起。具体的算法如下:

(1) BatchAdd(acc_X,{x_1,…,x_m})

① $x^* \leftarrow \prod_{i=1}^m x_i$

② $\mathrm{acc}_{X'} \leftarrow \mathrm{acc}^{x^*}$

(2) BatchDel(acc_X,(x_1,w_1)…(x_m,w_m))

① $\mathrm{acc}_{X'} \leftarrow w_1$

② $x^* \leftarrow x_1$

③ for $i \leftarrow 2, i \leqslant m$

④ $\mathrm{acc}_{X'} \leftarrow$ShamirTrick($\mathrm{acc}_{X'}$,$w_i$,$x^*$,$x_i$)

⑤ $x^* \leftarrow x^* \cdot x_i$

⑥ return $\mathrm{acc}_{X'}$

(3) ShamirTrick(w_1,w_2,x_1,x_2)

① if $w_1^{x_1} \neq w_2^{x_2}$ return $\perp$

② a,$b \leftarrow$Bezout(x_1,x_2)

③ return $w_1^b w_2^a$

在进行批处理成员验证时，一种思路是将所有元素相乘，为其计算一个证据来进行验证，另一种思路是将所有证据聚合为一个证据，然后进行验证。然而无论哪种思路，在进行验证时，都需要验证等式 $\mathrm{acc}_X = w_x^{x^*} \bmod N$ 是否成立，批量验证的元素数量越多，x^* 越大，计算 w_{x^*} 的指数运算 $w_x^{x^*}$ 与 x^* 大小线性相关。为了提高验证效率，BONEH 等[19]改进了由 WESOLOWSKI[31]提出的指数证明(PoE)协议，使之更适用于累加器元素的批处理验证。此指数证明协议可以让验证方信服而无需进行整个求幂运算 $w_x^{x^*}$。在该协议中，证明方和验证方都知道 3 个数 x、w 和 u，验证方仅验证 $u^x = w$ 而无需进行整个求幂运算。该协议的构造思路为：把指数 x 分解为商 q 和余数 r，使得 $x = ql + r$，其中 l 是验证方发送给证明方的素数，构造验证等式 $Q^l u^r = u^{ql} u^r = u^{ql+r} = u^x = w$，其中 $Q = u^q$。验证方在验证时需要进行的指数运算是 Q^l 和 u^r，而 l、r 都远小于 x，并且不会随着 x 的增大而增大。PoE 协议具体构造如下：

PoE 协议输入：u、$w \in G$，$x \in Z$；证明：$u^x = w$。

① 验证方发送随机的 λ 比特的素数 l 给证明方；

② 证明方计算 (q, r)，使得 $x = ql + r$，并将 $Q = u^q$ 发送给验证方；

③ 验证方计算 $r = x \bmod l$，当 $Q^l u^r = w$ 成立时接受。

此外，为了减小通信开销，BONEH 等[19]也使用了 Fiat-Shamir 启发式方法[32]及多轮协议的概括[33]，将 PoE 协议改成了非交互式协议 NI-PoE。并将 NI-PoE 协议运用到累加器中来提高批处理成员验证的效率，具体构造如下：

$\mathrm{MemWitCreate}^*(\mathrm{acc}_X, \{x_1, \cdots, x_n\})$

① $x^* = \prod_{i=1}^{n} x_i$

② $w_{x^*} \leftarrow \mathrm{WitCreate}(\mathrm{acc}_X, x^*)$

③ return w_{x^*}，$\pi = \mathrm{NI-PoE}(x^*, w_{x^*}, \mathrm{acc}_X)$

$\mathrm{VerMem}^*(\mathrm{acc}_X, \{x_1, \cdots, x_n\}, w = \{w_{x^*}, \pi\})$

① return NI-PoE. verify($\prod_{i=1}^{n} x_i, w_{x^*}, \mathrm{acc}_X, \pi$)

除了成员验证效率得到了提高，非成员验证的效率也提高了。当进行批处理非成员验证时，一种方法是通过 $\mathrm{NonMemWitCreate}^*$ 算法将多个非成员元素直接相乘来创建证据，验证时只需要验证 $\mathrm{VerNonMem}^*$ 算法中的协议运行是否成功。另一种方法是通过 AggNonMemWit 算法聚合这些非成员证据来创建证据。

$\mathrm{NonMemWitCreate}^*$ 算法和 $\mathrm{VerNonMem}^*$ 算法具体如下：发现此时非成员证据为 $u = \{V, B\}$，其中 $V = \mathrm{acc}_X^a$。而之前 LI 等人的方案[7]中非成员证据为 $u = \{a, B\}$。作者把 a 替换为 V 的原因在于：批处理非成员证据的验证为 $\mathrm{acc}_X^a B^{y^*} = g$，其中 y^* 是要批处理验证的非成员元素的乘积。验证方在验证时需要计算 acc_X^a，而证据 a 是通过扩展欧几里得算法 $ax^* + by^* = 1$ 得到的，a 与 y^* 线性相关；验证方在计算 acc_X^a 时，计算量随着 y^* 的增大而呈线性增长。因此作者直接设置证据为 $V = \mathrm{acc}_X^a$，减少了验证方的计算量，但此时还存在一个问题，验证等式 $VB^{y^*} = g$ 恒成立，敌手可以在不知道 a 的情况下设置 $V = gB^{-y^*}$ 来进行欺骗。为了解决这个问题，作者把 PoE 协议扩展到对离散对数知识的证明协议(Proof of Knowledge of Exponent，PoKE2)，该协议中，验证方不知道指数的知识，而在 PoE 协议中，指数的知识是被验证方掌握的。采用此协议可以证明证明方知晓 $V = \mathrm{acc}_X^a$ 中的 a，来确保 V 被诚实创建。

PoKE2 协议输入：u、$w \in G$；证据：$x \in Z$；证明：$u^x = w$。

① 验证方发送 g 给证明方；

② 证明方发送 $z = g^x$ 给验证方；

③ 验证方发送随机的 λ 比特的素数 l 和一个挑战值 α 给证明方；

④ 证明方计算 (q, r)，使得 $x = ql + r$，并将 $Q = u^q g^{\alpha q}$ 和 r 发送给验证方；

⑤ 验证方检查 $Q^l u^r g^{\alpha r} = w z^{\alpha}$ 是否成立。

$\mathrm{NonMemWitCreate}^*(\mathrm{acc}_X, x^*, y^*)$：//$\mathrm{acc}_X = g^{x^*}$，$x^* = \prod_{x \in X} x$，$y \in \prod y_i$

① $a, b \leftarrow \mathrm{Bezout}(x^*, y^*)$ //$ax^* + by^* = 1$

② $V\leftarrow \mathrm{acc}_X^a, B\leftarrow g^b // u=\{V,B\}$

③ $\pi_V\leftarrow$NI-PoKE2$(\mathrm{acc}_X, V; a) // V=\mathrm{acc}_X^a$

④ $\pi_g\leftarrow$NI-PoE$(y^*, B, gV^{-1}) // B^{y^*}=gV^{-1}$

⑤ return$\{V,B,\pi_V,\pi_g\}$

VerNonMem$^*(\mathrm{acc}_X, u=\{V,B,\pi_V,\pi_g\},\{y_1,\cdots,y_n\})$

① return NI-PoKE2. verify$(\mathrm{acc}_X,V,\pi_V)\wedge$NI-PoE. verify$(\prod_{i=1}^{n} y_i, B, gV^{-1}, \pi_g)$

AggNonMemWit 算法将多个独立的非成员证据聚合为一个恒定大小的证据。该算法首先使用 Bezout 公式通过扩展欧几里得算法求(a,b),使得 $ay_1+by_2=1$,其中 y_1 和 y_2 是两个非成员元素;然后聚合 y_1 和 y_2 的证据 u_1 和 u_2,得到证据 $u_{12}=(V,B_{12})=(\mathrm{acc}_X^{a_{12}}, B_{12})$。验证时同上述一样。AggNonMemWit 具体构造如下:

AggNonMemWit$(y_1,y_2,u_1=(\mathrm{acc}_X^{a_1},B_1),u_2=(\mathrm{acc}_X^{a_2},B_2),\mathrm{acc}_X=g^{x^*}, x^*=\prod_{x\in X}x)$

① $a,b\leftarrow$Bezout(y_1,y_2)

② $B'\leftarrow B_1^b B_2^a$

③ $a'\leftarrow ba_1y_2+aa_2y_1$

④ $a_{12}\leftarrow a' \bmod y_1y_2$

⑤ $V\leftarrow \mathrm{acc}_X^{a_{12}}$。

⑥ $B_{12}\leftarrow B'\mathrm{acc}_X^{a'/y_1y_2}$

⑦ $\pi_V\leftarrow$NI-PoKE2$(\mathrm{acc}_X, V; a_{12}) // V=\mathrm{acc}_X^{a_{12}}$

⑧ $\pi_g\leftarrow$NI-PoE$(y_{12}, B_{12}, gV^{-1}) // B_{12}^{y_{12}}=gV^{-1}, y_{12}=y_1y_2$,

⑨ return$\{a_{12},B_{12},\pi_V,\pi_g\}$

3　基于双线性映射的累加器

RSA 累加器方案中,累加器聚合的元素被限制为素数,需要把元素转变为素数[34]后才能进行累加。为了解决这个问题,NGUYEN 等[9]首次提出了基于强 Diffie-Hellman(SDH)假设的双线性映射动态累加器,此累加器不要求累加值域为素数。DAMGARD 等[10]和 AU 等[11]补充了 LI 等[7]提出的通用累加器构造,使得双线性映射累加器同 RSA 累加器一样,都具备了动态和通用的功能。然而,上述累加器中累加集合的大小需提前设定,如累加的元素数量上限为 q。因此,TOLGA 等[35]通过一组累加器突破了上述累加集合大小受限的问题。此外,CAMENISCH 等[12]也给出了一种基于 q-DHE 假设的方案,此方案对证据和累加器的更新无需陷门参与,即支持公开更新。

3.1　动态累加器

RSA 累加器的累加域元素要求为素数,为了解决这一问题,NGUYEN 等[9]提出了基于 q-SDH 假设的双线性映射累加器,其中累加的元素的数量上限为 q。相比于 RSA 累加器采用模幂运算聚合累加元素,此构造以双线性对运算为基础。具体构造如下:

(1) Gen(1^λ):累加器管理员生成公私钥对。输入 1^λ,生成 $\mathrm{pk}_{\mathrm{acc}}=(g,g^s,\cdots,g^{s^q},u)$、$\mathrm{sk}_{\mathrm{acc}}=s\in_R Z_p^*$,其中 q 是累加元素的上界,累加元素值域为 $Z_p\backslash\{-s\}$,u 由 Z_p^* 随机生成。

(2) Eval$(X,\mathrm{pk}_{\mathrm{acc}})$:累加器管理员计算累加值 acc_X。输入集合 $X=\{x_1,\cdots,x_n\}\subset Z_p\backslash\{-s\}$,其中 $n\leqslant q$,使用 $\mathrm{pk}_{\mathrm{acc}}$ 而无需私钥 s 的情况下计算累加器 $\mathrm{acc}_X=g^u\prod_{x\in X}(x+s)$。

(3) WitCreate$(\mathrm{pk}_{\mathrm{acc}},x_i,\mathrm{acc}_X,X)$:累加器管理员创建用户 x_i 的证据。输入公钥 $\mathrm{pk}_{\mathrm{acc}}$、累加值 acc_X、集合 X 和元素 x_i,生成 $x_i\in X$ 的证据为 $w_i=g^u\prod_{x\in X\backslash\{x_i\}}(x+s)$。

(4) VerMem(acc_X,x_i,w_i):验证方验证 x_i 是否在累加器中。通过验证 $e(w_i,g^{x_i}g^s)=e(\mathrm{acc}_X,g)$ 是否成立。如果成立,则输出 1,否则输出 0。

(5) Add$(\mathrm{acc}_X,x,\mathrm{sk}_{\mathrm{acc}})$:添加元素 $x\in X$ 到累加器,更新后的累加器 $\mathrm{acc}_{X'}=\mathrm{acc}_{X\cup\{x\}}=\mathrm{acc}_X^{x+s}$。

(6) Del($\mathrm{acc}_X, x, \mathrm{sk}_{\mathrm{acc}}$):从累加器中删除值 x 时,更新后的累加器 $\mathrm{acc}_X' = \mathrm{acc}_{X\setminus\{x\}} = \mathrm{acc}_X^{1/(x+s)}$。

(7) MemWitUp($x, w_i, x_i, \mathrm{pk}_{\mathrm{acc}}, \mathrm{acc}_X, \mathrm{acc}_{X'}$):用户更新元素 x_i 的证据。当添加元素 x 到累加器时,更新 x_i 的证据 $w_i' = \mathrm{acc}_X w_i^{x-x_i}$。该更新操作是正确的,因为 $\mathrm{acc}_X w_i^{x-x_i} = w_i^{x_i+s} w_i^{x-x_i} = w_i^{x+s} = w_i'$;而在删除元素 x 时,其中 $x \neq x_i \in X$,更新 x_i 的证据通过计算 $w_i' = w_i^{1/(x-x_i)} \mathrm{acc}_{X'}^{1/(x_i-x)}$。$w_i'$ 计算正确,因为

$$w_i^{1/(x-x_i)} \mathrm{acc}_{X'}^{1/(x_i-x)} = w_i^{1/(x-x_i)} \mathrm{acc}_X^{1/(x_i-x)(x+s)} = w_i^{1/(x-x_i)} w_i^{(x_i+s)/(x-x_i)(x+s)} = w_i^{1/(x+s)} = w_i' \quad 。$$

要注意的是,在上述构造中,公钥大小与累加元素的上限 q 线性相关,如果 q 很大,则元素公钥将会非常大,而在 RSA 累加器中,公钥是常数 N。此外,RSA 动态累加器只有在删除操作更新累加值时会用到陷门信息 $\mathrm{sk}_{\mathrm{acc}}$,而在此构造中,更新累加值时,包括添加和删除都需要用到陷门信息。

双线性映射动态累加器除了该基于 q-SDH 假设的方案之外,CAMENISCH 等[12] 也给出了一种基于 q-DHE 假设的构造。此方案采用了两个公开的集合 V 和 V_w,分别是被累加到累加器中的元素集合和证据 w_i 被创建时,累加器添加或删除了哪些元素的集合。当证据需要更新时,可以通过 V_w 对所有元素的证据公开更新。具体构造如下:

(1) Gen($1^\lambda, t$):输入参数 1^λ 和累加元素上限 t,累加器管理员随机选择值 $\gamma \in Z_q$,生成初始累加器 $\mathrm{acc}_\phi = 1$、初始状态 $\mathrm{state}_\phi = (\phi, g^{\gamma^1}, \cdots, g^{\gamma^q}, g^{\gamma^{q+2}}, \cdots, g^{\gamma^{2q}})$ 和公私钥对:

$$(\mathrm{pk}_{\mathrm{acc}}, \mathrm{sk}_{\mathrm{acc}}) = ((g_1, \cdots, g_q, g_{q+2}, \cdots, g_{2q}, e(g,g)^{\gamma^{q+1}}, \mathrm{pk}_{\mathrm{sig}}), (\gamma, \mathrm{sk}_{\mathrm{sig}})) =$$
$$((g^{\gamma^1}, \cdots, g^{\gamma^q}, g^{\gamma^{q+2}}, \cdots, g^{\gamma^{2q}}, e(g,g)^{\gamma^{q+1}}, \mathrm{pk}_{\mathrm{sig}}), (\gamma, \mathrm{sk}_{\mathrm{sig}})) \quad 。$$

(2) Add($\mathrm{sk}_{\mathrm{acc}}, i, \mathrm{acc}_V, \mathrm{state}_U$):累加器管理员在添加某个元素后计算新的累加器和状态。计算 $w = \prod_{j\in V}^{j\neq i} g_{q+1-j+i}$,用私钥 $\mathrm{sk}_{\mathrm{sig}}$ 通过签名 σ_i 使得 $g_i \| i$,输出证据 $w_i = (w, \sigma_i, g_i)$。添加元素 i 到累加器 acc_V 中并输出新的累加器 $\mathrm{acc}_{V\cup\{i\}} = \mathrm{acc}_V g_{n+1-j}$ 和新的状态:

$$\mathrm{state}_{U\cup\{i\}} = \{U \cup \{i\}, g_1, \cdots, g_n, g_{n+2}, \cdots, g_{2n}\} \quad 。$$

(3) Update($\mathrm{pk}_{\mathrm{acc}}, V, \mathrm{state}_U$):用户输出 $\mathrm{acc}_V = \prod_{v\in V} g_{n+1-v} (V \subset U)$。

(4) WitUp($\mathrm{pk}_{\mathrm{acc}}, w_i, V_w, V, \mathrm{acc}_V, \mathrm{state}_U$):任何不可信实体都可以进行证据的更新。如果 $i \in V$ 且 $V \cup V_w \subset U$,计算

$$w' = w \frac{\prod_{j\in V\setminus V_w} g_{n+1-j+i}}{\prod_{j\in V_w\setminus V} g_{n+1-j+i}} \quad 。$$

输出 $w_i' = (w', \sigma_i, g_i)$。

(5) VerMem($\mathrm{pk}_{\mathrm{acc}}, i, w_i, \mathrm{acc}_V$):验证方验证 i 是否在累加器中。不可信实体更新证据 w_i 后,用户 i 获得新的证据 w_i' 后,通过等式 $e(g_i, \mathrm{acc}_V) = ze(g, w)$ 来验证 i 是否在累加器中。

在删除成员时,RSA 累加器[6-7] 和基于 SDH 假设的双线性映射累加器[9-10] 更新累加值都会用到陷门信息,并且更新操作都进行了幂运算。而该方案在删除成员时更新累加器不需要陷门信息,并且更新操作无需进行幂运算,只用进行简单的乘法运算,因此计算效率大大提升。

3.2 通用累加器

上述累加器无法支持非成员验证,因此,DAMGARD 等[10] 首次基于 SDH 假设将双线性映射累加器扩展为通用累加器,具体非成员证据的构造和验证方法如下:

(1) WitCreate($\mathrm{pk}_{\mathrm{acc}}, y, s, X$). :累加器管理员创建 y 的非成员证据。输入公钥 $\mathrm{pk}_{\mathrm{acc}}$,累加值域 X 和元素 y,生成 $y \notin X$ 的非成员证据为(a_y, B_y):$B_y = -f(-y) = -\prod_{x\in X}(x-y) \bmod p \neq 0$,$a_y = g^{[f(s)-f(-y)]/(y+s)}$,其中 $f(s) = \prod_{x\in X}(x+s)$,且$(y+s) \mid [f(s)+B_y]$。

(2) VerNonMem($\mathrm{acc}_X, y, (a_y, B_y)$):验证方验证元素 y 的非成员身份。已知公共参数 g^s,验证通过检查 $e(a_y, g^y g^s) = e(\mathrm{acc}_X g^{B_y}, g)$ 是否成立。如果成立,则输出 1,否则输出 0。

在该方案的基础上,AU 等[11] 也提出了基于 SDH 假设的通用累加器方案,使得无需使用陷门信息便可

进行累加器的更新。

4　基于 Merkle 哈希树的累加器

Merkle 哈希树累加器是一类设计简洁且高效的累加器。Merkle 哈希树中,叶子节点为集合元素的哈希值,内部节点为左右孩子的哈希值,根节点为最终累加值。元素 x_i 的成员证据由 x_i 到根节点路径上的所有兄弟节点组成,通过重构根节点来判断元素 x_i 是否存在于集合中。然而,RSA 累加器和双线性映射累加器只需 $O(1)$大小的证据,而 Merkle 哈希树累加器需要 $O(\log N)$大小的证据,其中 N 表示累加元素的总数。

CAMACHO 等[13]给出了一个通用累加器方案的简单构造,该方案和 LI 等[7]的通用累加器方案在功能上是相似的,适用于动态集合并允许非成员身份证明,并且该方案在存在抗碰撞哈希函数的假设下可证明是安全的。该方案实现非成员证明的主要思想为:集合中元素按顺序存储,若某元素的左右相邻元素为连续元素,则该元素不在集合中。例如,x 的左右相邻元素为 x_α、x_β,且 x_α、x_β 连续,则 x 为非成员元素。方案具体构造如下:

(1) Gen(1^λ):累加器管理员输入安全参数 λ,生成初始累加器 acc_0 和 M_0。初始集合 X 为空,是 $M=\{0,1\}^\lambda$ 的子集;然后选择一个哈希函数 H,计算一个随机索引 $i\in K$,并设置 $H=H_i$;最后将 m 的初始值设置为 $H(-\infty\parallel+\infty)$ 的根节点 N_m,并且累加器管理员公布 $\mathrm{acc}_0=\mathrm{Proof}_{N_m}$。

(2) WitCreate(m,x):累加器管理员创建证据。输入 $x\in M$ 和内存 m,计算证据 $w=(w_1,w_2)$ 如下:

① 如果 $x\in X$,则证据 $w_1=(x_\alpha,x_\beta)$,其中 $x=x_\alpha$ 或 $x=x_\beta$;否则 $x\notin X$,证据 $w_1=(x_\alpha,x_\beta)$,其中 $x_\alpha<x<x_\beta$。

② 设置 w_2 是由 $H(x_\alpha\parallel x_\beta)$ 生成的 m 的最小子树。

(3) Update(m,x):累加器管理员更新累加器、内存 m 和证据 w。输入 $x\in X$,acc 和 m。

① 添加:$x\notin X$ 时,添加 x 操作如下:将 m 中适当节点处的值 $H(x_\alpha\parallel x_\beta)$ 替换为 $H(x_\alpha\parallel x)$,其中 $x_\alpha<x<x_\beta$,更新后的 m' 是一棵平衡树。让 $V_{\mathrm{Par}(N)}$ 为更新的叶子的父节点,更新后的 $\mathrm{acc}'=\mathrm{Proof}_{m'}$,证据 $w'=(\mathrm{add},w_1,w_2)$。

② 删除:删除 x 时,查找包含 x 的两个节点,用 V_α,V_β 表示。删除 x 后,用 $H(x_\alpha\parallel x_\beta)$ 代替之前的 $H(x_\alpha\parallel x)$ 和 $H(x\parallel x_\beta)$,并保持树平衡。更新后的树为 m'、累加器 $\mathrm{acc}'=\mathrm{Proof}_{m'}$、证据 $w'=(\mathrm{del},w_1,w_2,w_3)$。

(4) CheckUpdate(acc,acc$'$,w'):在添加或删除元素后,用户检查累加器管理员提供的更新后的元素,更新正确,则输出 OK。输入 $x\in M$、更新前的累加器 acc、更新后的累加器 acc$'$ 和更新后的证据 w'。如果 $w'=(\mathrm{add},w_1,w_2)$ 且符合下列前提,则算法输出 1;否则输出 0。删除操作也类似,这里省略。

① w_1 是通过在 w_2 上添加叶子获得的树。

② 除了值是 $H(x_\alpha\parallel x_\beta)$ 的节点以外,w_1、w_2 树中其余所有节点都相同。

③ $\mathrm{Proof}_{w_1}=\mathrm{acc}$,$\mathrm{Proof}_{w_2}=\mathrm{acc}'$。

④ $H(x_\alpha\parallel x),H(x\parallel x_\beta)\in V_{w_2}$。

(5) VerMem(x,w,acc):验证方验证某个元素是否在集合中。输入 $x\in M$、证据 $w=((w_1,w_2),u)$ 和累加器 acc。检查(a)$\mathrm{Proof}_u=\mathrm{acc}$;(b)$H(x'\parallel x'')\in V_u$ 和(c)$(x=x')$,(c)$(x=x'')$ 或(c$'$)$(x'<x<x'')$。如果(a)(b)(c) 成立,则输出 1;如果(a)(b)(c$'$) 成立,则输出 0,否则输出 $\perp$。

首先,在 RSA 累加器和双线性映射累加器中,需要累加器管理员生成陷门信息来更新累加器或证据,而 Merkle 哈希树累加器[13]不需要陷门信息来更新累加器或证据,因而无需管理员参与。其次,Merkle 哈希树累加器主要基于哈希算法,而 RSA 累加器和双线性映射累加器分别需要模指数运算和双线性对运算,所以 Merkle 哈希树累加器计算效率高。然而,Merkle 哈希树累加器的证据大小与集合大小呈对数关系,所以通信代价较高。

5　三类密码累加器的分析与比较

对上述 3 类密码累加器的构造方案进行了比较和总结。RSA 累加器基于强 RSA 假设,具有较短的公

共参数，且可以很容易扩展到通用累加器[7]或动态累加器[6]，但此类型下的累加器通常需要昂贵的操作才能将元素以素数形式编码。双线性映射累加器有基于 q-SDH 假设和 q-DHE 假设的两种类型，它们不需要素数编码，但需要最大元素数量的公共参数。大多数 RSA 累加器和双线性映射累加器都需要可信累加器管理员生成陷门信息。如在 RSA 累加器中，任何知道模 $N=pq$ 分解的对手都可以轻易地作弊。而 Merkle 哈希树累加器不需要陷门信息，把这类不需要陷门信息的累加器也称为强累加器。Merkle 哈希树累加器基于抗碰撞的哈希函数假设，是一类高效简洁的累加器，但是该类构造不需要可信的累加器管理员，所以需要 $O(\log N)$大小的证据（N 表示累加集合的元素数量），而前两类需要可信设置的累加器只需 $O(1)$大小的证据。此外，RSA 累加器和双线性映射累加器也有一些构造[23,28]。这些构造下，累加器管理员完全不受信任，但这些构造要么证据数量随集合 X 的大小呈对数增长，要么依赖尚未在密码学进行深入研究的代数群。

表 1 给出了 3 类密码累加器构造方案的详细比较[26,36]。其中，S 代表静态累加器，D 代表动态累加器，U 代表通用累加器，w 代表成员证据的更新，u 代表非成员证据的更新，a 和 d 分别代表添加和删除操作时累加值的更新，acc 代表累加值的更新，B 代表累加器累加元素是否具有上限，$|\mathrm{pk_{acc}}|$、$|\mathrm{w}|$、$|\mathrm{u}|$分别代表公共参数，成员证据和非成员证据的大小。

表 1 不同类型累加器的比较

实现工具	假设	方案	类型	公共更新				B	$\lvert\mathrm{pk_{acc}}\rvert$	$\lvert\mathrm{w}\rvert$	$\lvert\mathrm{u}\rvert$
				w	u	a	d				
RSA 体制	S-RSA	BM[5]	S	—	—	—	—	×	$O(1)$	$O(1)$	—
		BP[8]	S	—	—	—	—	×	$O(1)$	$O(1)$	—
		CL[6]	D	√	—	√	×	×	$O(1)$	$O(1)$	—
		LLX[7]	DU	√	√	√	×	×	$O(1)$	$O(1)$	$O(1)$
		BBF[19]	DU	√	√	√	√	×	$O(1)$	$O(1)$	$O(1)$
双线性映射	q-SDH	NY[9]	D	√	—	×	×	√	$O(q)$	$O(1)$	—
		DT[10]	DU	√	√	×	×	√	$O(\mathrm{q})$	$O(1)$	$O(1)$
		ATSM[11]	DU	√	√	×	×	√	$O(\mathrm{q})$	$O(1)$	$O(1)$
	q-DEH	CKS[12]	D	√	—	×	√	√	$O(\mathrm{q})$	$O(1)$	—
Merkle 哈希树	CRH	CHKO[13]	U	—	—	—	—	×	$O(1)$	$O(\log\mathrm{N})$	$O(\log\mathrm{N})$

注：√表示符合此特征，×表示不符合此特征，—表示不存在此功能或此性质。

6 应 用

累加器自提出以来，有着广泛的应用场景。包括时间戳[5]、群签名[6,37]、环签名[38]、访问控制系统[39]、匿名凭证系统[6]、加密货币[19]、认证数据结构（ADS）[15-18]等。此外，还被用在隐私保护数据外包[20]、认证字典[21]、编辑[22]、负责任的证书管理[23-24]等应用中。介绍几种典型应用：累加器在时间戳、成员资格撤销中的应用。此外，还会介绍最近比较流行的应用：累加器在区块链[40-42]中的应用。

6.1 时间戳

BENALOH 等[5]首次提出了密码学累加器，并应用于时间戳中。时间戳可以在任何文档上附加一个“发布”日期，以证明原始文件在签名时间之前已经存在。在传统的时间戳系统中，机构需要对文件逐个添加时间戳，验证方也需要逐个验证，使得时间戳的生成和验证代价高。为了提高时间戳系统的效率，BENALOH 等[5]设计了基于密码累加器的时间戳系统，该系统可以批量的生成和验证时间戳。具体方法为：将某个时刻产生的所有文档聚合到累加器中，然后对该累加器附加时间戳。验证 t 时刻的文档时，仅需验证该文档存在于 t 时刻的累加器中，并且可以实现多个文档时间戳的批量验证，大大提高了时间戳系统的生成和验证效率。

6.2 成员资格撤销

静态累加器[5]可以用来解决时间戳、成员资格测试等集合成员无需变动的问题。然而，存在一些应用场

景需要对成员进行添加或删除操作,如匿名凭证撤销系统、群签名方案、身份托管方案等[43-45]会进行撤销成员的操作。传统的撤销方案[43]如下:一种是当用户尝试访问资源时,验证方通过在数据库中查找用户的证书,使得证书被撤销的用户无法访问资源,但此查询操作与用户数线性相关;另一种方法是每天为所有合格用户重新颁发新的证书,使得证书被撤销的用户无法使用旧证书访问资源,此方法中,合法用户进行证书验证和管理员重新颁发证书的通信成本高,与成员数线性相关。上述撤销方案的困难度与合法成员数线性相关。为了解决此类撤销问题,CAMENISCH 等[6]提出了基于动态累加器的成员撤销系统。在该方案中,群管理员充当累加器管理员的角色,将合法证书聚合成累加值。当撤销成员时,群管理员用累加器陷门信息更新累加值。由于动态累加器在撤销证书时,更新累加值和证据的计算量与合格成员数和撤销用户数量无关,所以大大提高了成员资格撤销的效率。

6.3　无状态区块链

近年来,区块链的扩容技术[19,40,46-47]有很多,其中无状态区块链[19]是一项很重要的扩容技术。如在比特币中,当节点 A 向 B 发起一笔转账时,除了需要通过签名来验证这笔交易是否是 A 发起的,还要验证 A 是否有这笔钱。在传统验证方法中,验证节点维护网络中每个节点的未花费交易集合 UTXO[19],验证节点通过查看节点 A 发起的交易是否在 UTXO 集合中来判断 A 是否发起了合法的转账。然而,随着参与比特币交易的用户越来越多,UTXO 集合越来越大,验证节点的存储压力也越来越大。除了比特币等基于 UTXO 模型的区块链面临这样的问题,基于账户的模型,如以太坊也面临着同样的问题。为了解决此类问题,BONEH 等[19]提出了基于支持批处理累加器的无状态区块链。在该无状态区块链的构造中,验证节点无需存储整个 UTXO 集合,只需存储该 UTXO 集合中元素的累加值,用户节点存储与自己相关的 UTXO 及证据。当进行一笔交易时,交易发起者会提供附带证据的交易给验证节点,验证节点通过判断交易和证据是否通过验证来确定这笔交易是否合法。此外,批处理累加器可以对交易进行高效批量验证,不仅减轻了全节点的存储压力,而且进一步提高了验证效率。在此之后,也产生了很多区块链扩容技术[19,40,46,47]。

7　总结和展望

笔者主要围绕密码累加器的相关研究内容展开综述,并按实现方式、功能对累加器进行了分类,详细描述了基于 RSA 的累加器、基于双线性映射的累加器和基于 Merkle 哈希树的累加器的经典构造。经过分析,现有的累加器方案仍然存在一些不足,未来的研究方向可以关注以下几点:

(1)基于双线性对的累加器解决了累加元素必须为素数的不足,然而,该类累加器的公开参数的大小与集合的大小相同,增加了通信代价。此外,累加器的大小需提前设置,即累加器包含的元素个数固定。在动态应用场景中,元素的个数不断增加,若参数设置过大,则增加公开参数的传输开销;若参数设置过小,则应用场景受限。因此,如何突破双线性映射累加器的大小限制成为未来研究的方向之一。

(2)在可验证外包数据存储中,使用数字签名、布隆过滤器、Merkle 哈希树等验证数据结构验证外包数据的完整性,具有证据生成和验证效率高等优点。然而,该类数据验证结构为私有验证,即只有拥有私钥的用户才能验证结构,使得使用场景受限。向量承诺、密码累加器可以实现公开验证,然而,构造和验证效率低下。因此,如何构造高效的密码累加器,应用于动态密文检索、无状态区块链、分布式存储系统是未来研究的方向之一。

参考文献:

[1] CAMPANELLI M, FIORE D, GRECO N, et al. Incrementally Aggregatable Vector Commitments and Applications to Verifiable Decentralized Storage [C]//International Conference on the Theory and Application of Cryptology and Information Security. Berlin: Springer, 2020: 3-35.

[2] WANG Q, ZHOU F, XU J, et al. A (Zero-Knowledge) Vector Commitment with Sum Binding and its Applications[J]. The Computer Journal, 2020, 63(4): 633-647.

[3] MICALI S, RABIN M, KILIANJ. Zero-Knowledge Sets [C]//Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science. Piscataway: IEEE, 2003: 80-91.

[4] WU C, CHEN X, SUSILO W. Concise ID-Based Mercurial Functional Commitments and Applications to Zero-Knowledge Sets[J]. International Journal of Information Security, 2020, 19(2): 453-464.

[5] BENALOH J, MD M. One-Way Accumulators: A Decentralized Alternative to Digital Signatures[C]//Advances in Cryptology. Berlin: Springer, 1993: 274-285.

[6] CAMENISCH J, LYSYANSKAYA A. Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials[C]//Proceedings of the Advances in Cryptology. Berlin: Springer, 2002: 61-76.

[7] LI J, LI N, RUI X. Universal Accumulators with Efficient Nonmembership Proofs[C]//Proceedings of the Applied Cryptography and Network Security. Berlin: Springer, 2007: 253-269.

[8] BARIC N, PFITZMANN B. Collision-Free Accumulators and Fail-Stop Signature Schemes Without Trees[C]//Proceedings of the International Conference on the Theory & Applications of Cryptographic Techniques. Berlin: Springer, 1997: 480-494.

[9] NGUYEN L. Accumulators from Bilinear Pairings and Applications to ID-Based Ring Signatures and Group Membership Revocation[C]//Proceedings of the Topics in Cryptology. Berlin: Springer, 2005: 275-292.

[10] DAMGARD I, TRIANDOPOULOS N. Supporting Nonmembership Proofs with Bilinear-Map Accumulators(2008)[EB/OL]. [2008-1-1]. http://eprint.iacr.org/2008/538.pdf.

[11] AU M H, TSANG P P, SUSILO W, et al. Dynamic Universal Accumulators for DDH Groups and Their Application to Attribute-Based Anonymous Credential Systems[C]//Proceedings of the The Cryptographers' Track at the RSA Conference 2009 on Topics in Cryptology. Berlin: Springer, 2009: 295-308.

[12] CAMENISCH J, KOHLWEISS M, SORIENTE C. An Accumulator Based on Bilinear Maps and Efficient Revocation for Anonymous Credentials[C]//Proceedings of the 12th International Conference on Practice and Theory in Public Key Cryptography. Berlin: Springer, 2009: 481-500.

[13] CAMACHO P, HEVIA A, KIWI M, et al. Strong Accumulators from Collision-Resistant Hashing[C]//Proceedings of the Information Security Conference. Berlin: Springer, 2008: 471-486.

[14] GOLDREICH O, OREN Y. Definitions and Properties of Zero-Knowledge Proof Systems[J]. Journal of Cryptology, 1994, 7(1): 1-32.

[15] PENNINO D, PIZZONIA M, GRISCIOLI F. Pipeline-Integrity: Scaling the Use of Authenticated Data Structures up to the Cloud[J]. Future Generation Computer Systems, 2019, 100(1): 618-647.

[16] SUN Y, LIU Q, CHEN X, et al. An Adaptive Authenticated Data Structure with Privacy-Preserving for Big Data Stream in Cloud[J]. IEEE Transactions on Information Forensics and Security, 2020, 15: 3295-3310.

[17] MARTEL C, NUCKOLLS G, DEVANBU P, et al. A General Model for Authenticated Data Structures[J]. Algorithmica, 2004, 39(1): 21-41.

[18] XU J, WEI L, WU W, et al. Privacy-Preserving Data Integrity Verification by Using Lightweight Streaming Authenticated Data Structures for Healthcare Cyber-Physical System[J]. Future Generation Computer Systems, 2018, 108(1): 1287-1296.

[19] BONEH D, BENEDIKT B, FISCH B. Batching Techniques for Accumulators with Applications to IOPs and Stateless Blockchains[C]//Proceedings of the 39th Annual International Cryptology Conference. Berlin: Springer, 2019: 561-586.

[20] SLAMANIGO D. Dynamic Accumulator Based Discretionary Access Control for Outsourced Storage with Unlinkable Access[C]//Proceedings of the 16th International Conferenceon Financial Cryptography and Data Security. Berlin: Springer, 2012: 215-222.

[21] GOODRICH M T, TAMASSIA R, HASI J. An Efficient Dynamic and Distributed Cryptographic Accumulator[C]//Proceedings of the 5th International Conferenceon Information Security. Berlin: Springer, 2002: 372-388.

[22] PHLS H C, PETERS S, KAI S, et al. Malleable Signatures for Resource Constrained Platforms[C]//Proceedings of the 7th IFIP WGInternational Workshop. Berlin: Springer, 2013: 18-33.

[23] BULDAS A, LAUD P, LIPMAA H. Accountable Certificate Management Using Undeniable Attestations[C]//Proceedings of the 7th ACM conference on Computer and Communications Security. New York: ACM, 2002: 9-17.

[24] SK A, LZ A, XY B, et al. Accountable Credential Management System for Vehicular Communication[J]. Vehicular Communications, 2020, 25(4): 100279.

[25] MEER H D, LIEDEL M, POHLS H C, et al. Indistinguishability of One-Way Accumulators(2012)[EB/OL]. [2012-12-20]https://www.fim.uni-passau.de/fileadmin/dokumente/fakultaeten/fim/forschung/mip-berichte/MIP_1210.pdf.

[26] DERLER D, HANSER C, SLAMANIG D. Revisiting Cryptographic Accumulators, Additional Properties and Relations to Other Primitives[C]//Proceedings of the Topics in Cryptology. Berlin: Springer, 2015: 127-144.

[27] SANDER T. Efficient Accumulators without Trapdoor[C]//Proceedings of the Second International Conference on Information and Communications Security. Berlin:Springer, 1999:252-262.
[28] LIPMAA H. Secure Accumulators from Euclidean Rings without Trusted Setup[C]//Proceedings of the 10th International Conference. Berlin:Springer, 2012:224-240.
[29] TSUDIK G, XU S. Accumulating Composites and Improved Group Signing[C]//Proceedings of the Advances in Cryptology, Berlin:Springer, 2003:269-286.
[30] CAMACHO P, HEVIA A. On the Impossibility of Batch Update for Cryptographic Accumulators[C]//Proceedings of the Progress in Cryptology. Berlin:Springer. 2009:178-188.
[31] WESOLOWSKI B. Efficient Verifiable Delay Functions[J]. Journal of Cryptology, 2019, 33(4):379-407.
[32] FIAT A, SHAMIR A. How to Prove Yourself:Practical Solutions to Identification and Signature Problems[C]//Proceedings of the Advances in Cryptology. Berlin:Springer, 2000:186-194.
[33] BEN-SASSON E, CHIESA A, SPOONER N. Interactive Oracle Proofs[C]//Proceedings of the Theory of Cryptography. Berlin:Springer, 2016:31-60.
[34] FORTNOW L, GOLDWASSER S, MICALI S, et al. The Knowledge Complexity of Interactive Proof Systems[J]. SIAM Journal on Computing, 1989, 18(1):186-208.
[35] TOLGA A, LANN. Revocation for Delegatable Anonymous Credentials[C]//International Workshop on Public Key Cryptography. Berlin:Springer, 2011:423-440.
[36] BADIMTSI F, CANETTI R, YAKOUBOV S. Universally Composable Accumulators[C]//Proceedings of the Topics in Cryptology. Berlin:Springer, 2020:638-666.
[37] 程小刚,王箭,杜吉祥.群签名综述[J].计算机应用研究,2013,30(10):2881-2886.
CHENG Xiaogang, WANG Jian, DU Jixiang. Overview of Group Signatures[J]. Application Research of Computers, 2013, 30(10):2881-2886.
[38] GU K, DONG X, WANG L. Efficient Traceable Ring Signature Scheme without Pairings[J]. Advances in Mathematics of Communications, 2020, 14(2):207-232.
[39] BERA B, SAHA S, DAS A K, et al. Designing Blockchain-Based Access Control Protocol in IoT-Enabled Smart-Grid System[J]. IEEE Internet of Things Journal, 2020, 8(7):5744-5761.
[40] 王慧,王励成,柏雪,等.区块链隐私保护和扩容关键技术研究[J].西安电子科技大学学报,2020,47(5):28-39.
WANG Hui, WANG Licheng, BAI Xue, et al. Research on Key Technology of Blockchain Privacy Protection and Scalability[J]. Journal of Xidian University, 2020, 47(5):28-39.
[41] 陈思吉,翟社平,汪一景.一种基于环签名的区块链隐私保护算法[J].西安电子科技大学学报,2020,47(5):86-93.
CHEN Siji, ZHAI Sheping, WANG Yijing. A Blockchain Privacy Protection Algorithm Based on Ring Signature[J]. Journal of Xidian University, 2020, 47(5):86-93.
[42] OZCELIK I, MEDURY S, BROADDUS J, et al. An Overview of Cryptographic Accumulators[C]//International Conference on Information Systems Security and Privacy. Berlin:Springer, 2021:661-669.
[43] ATENIESE G, CAMENISCH J, JOYE M, et al. A Practical and Provably Secure Coalition-Resistant Group Signature Scheme[C]//Proceedings of the Advances in Cryptology. Berlin:Springer, 2000:255-270.
[44] BINDEL N, HERATH U, MCKAGUE M, et al. Transitioning to A Quantum-Resistant Public Key Infrastructure[C]//Proceedings of the Post-Quantum Cryptography. Berlin:Springer, 2017:384-405.
[45] CAMENISCH J, LYSYANSKAYA A. An Identity Escrow Scheme with Appointed Verifiers[C]//Proceedings of the Advances in Cryptology. Berlin:Springer, 2001:388-407.
[46] REDDYB S. SecurePrune:SecureBlock Pruning in UTXO Based Blockchains Using Accumulators[C]//2021 International Conference on COMmunication Systems &NETworks. Piscataway:IEEE, 2021:174-178.
[47] TOMESCU A, ABRAHAM I, BUTERIN V, et al. Aggregatable Subvector Commitments for Stateless Cryptocurrencies [C]//Proceedings of the Security and Cryptography for Networks, SCN 2020, in Lecture Notes in Computer Science. Berlin:Springer, 2020:45-64.

(编辑:牛姗姗)