Riffle：An anonymous Messaging System Handling Millions of Users

王发星
Lab of future network
USTC

摘要—Riposte是要一个匿名的消息广播系统。此工作主要有三点贡献：第一他是抵抗流量分析的系统，第二能够防止匿名的恶意用户破坏系统发送拒绝服务攻击。第三Riposte的可扩展性较号，能够同时支持成千上万的用户数量。本文主要使用的技术的分布式点函数式的PIR和多方计算（一个额外的审查服务器用于检查用户请求的合法性）来达到上述的三点贡献。

I. Introduction

在一个网络监控日益严峻的世界，互联网的机以和来自国家机器的强力审查使得严重威胁着人们的日常生活隐私。虽然现在有Tor等时下非常流行的匿名访问工具，但是此类工具发布的时间较为长久，因此对于当前拥有很强大资源的攻击者来说，基于代理的匿名通信工具很容易收到流量分析攻击而破坏用户的匿名性。当然人们已经意识到流量分析攻击的强大和危害，因此研究者也有提出相关的抗流量分析攻击的工具[1][2][3][4]，这些工具都之只能支持一个较小的匿名集合。因此在使匿名系统能够免疫流量分析的同时拥有一定的扩展性，让系统能够同时支持成千上万各用户进行匿名通信是一项值得研究的课题。另外需要指出的是，增强系统的可扩展性能够同时扩大系统的匿名集合，增强系统的匿名性。

首先Riposte使用了基于客户端/服务器的DC-net网络来对抗流量分析攻击。现有技术环境下，能够有效对抗流量分析的方法只有两种：DC-net和可验证的混合网络。可验证的混淆过程因为其巨大的计算开销（多次的公钥密钥学操作）难以适用在大量用户的场景中，并且在混淆过程中会引入不诚实的服务器行为，要检测这类行为需要更多的计算。与mixnet相比，基于DC-net的类似系统只需要简单快速的亦或操作就能提供更强于（信息论安全的）混合网络的抗流量分析保证。

另外为解决DC-net中固有恶意行为的问题，作者使用了一个多方安全计算协议来快速的检测恶意的用户行为且驱逐这类用户。

第三为了减少用户的带宽消耗，提高系统的可扩展性，作者受到PIR和分布式点函数（DPF）启发设计了一个协议来减少用户在每个周期上传到服务器的消息比特。值得一提的是作者使用了一种不同寻常的方式来适用PIR，在本文的目标场景下：公告板，推特等非即时交互场景，发送一条公告或者发表一份推特可以约等价于向数据库中写入一则消息，整个问题就能够转化为如何匿名的向数据库中写入数据。一般说来PIR技术是用来使得一个用户在一个或者多个服务器之间的读取匿名的数据， Riposte使用与传统PIR相反的做法，将PIR作为用户向数据写入匿名数据方法，来保证用户写入匿名的同时提高系统的性能。

总的来说消息发布以周期为单位，当用户想要发布一条消息时，用户生成一条写入请求，使用密码学方法将请求分割后分别存入多个服务器，小于特定数量的服务器合谋也不能得到关于用户写入消息位置和内容的任何信息。

II. 相关工作

虽然mixnet和DC-net已经被广泛研究，但是在面临恶意的服务器或者客户端时，两者都需要昂贵的零知识证明来检测恶意行为和保护用户隐私，并且为匿名发送一条消息DC-net系统要求各个用户总共发送与匿名集合数量成正比的消息量，这样的明显系统牺牲了

系统的可扩展性，使得系统只能支持少量的用户，在一定程度上弱化了系统的匿名保证。

III. 系统目标和问题陈述

A. 系统目标

在多个至少一个是诚实服务器的环境下，我们认为服务器是一个保存固定长度数据项的数据库。Riposte可被认为是一个供用户发布消息的公告板，用户向多个服务器使用私密的数据写入技术来写入想要公布的消息，这种方法保证了单个（小于一定数量）服务器无从得知用户写入数据得位置和内容。

服务器在一个时间段内收集所有用户的写入请求，这个时间段可以由几种方法来确定：首先可以依据时间测度来例如十分钟，其次可以按照写入的数据量来例如有一万条写入请求，还可以依据某些设定条件。如何对周期进行定义是系统安全的关键，因为用户匿名集合的大小就只是诚实用户提交消息的数目。当服务器对周期达成一致时，在每个周期结束，服务器之间共享写入的数据来还原用户写入的真实数据。

B. 威胁模型

在Riposte中，所有的用户都被认为时不可信的。用户可能会提交不合法的消息来污染数据来破坏系统的可用性，或者跟服务器和其他用户合谋来破坏诚实用户的匿名性。

系统中的服务器被认为是可用的，服务器的恶意或者无意发生的故障只是让数据库的状态不可恢复但并不会对用户的匿名性造成威胁。

C. 安全目标

正确性：当所有服务器都诚实执行协议时，最后各个服务器所存储数据计算的结果即分别为各个用户想要发送的明文消息。

（s，t）-写入隐私：直观上来说，写入隐私保证了即使在攻击者控制了$n-2$各用户和t个服务器中的s个服务器下，攻击者想要得知用户具体在数据库某一行写入那些数据的正确概率和随即猜测大致相当。

对抗破坏攻击：系统当且仅当在攻击者控制了n个客户端时，在一个周期内只能向数据库中写入n条消息，这样的系统被认为是抗破坏攻击的。本文不考虑服务器自身破坏自身状态。

D. 稻草人协议

由前述所知，Riposte主要有少量维护数据库状态的服务器和大量的用户组成。用户将消息切分成多个消息片发送给每个服务器，每个服务器更新其数据库的状态，在周期结束后各服务器共享所有数据库状态来披露用户写入的真实数据。

最简单的方式是使用类似DC-net的客户端/服务器模式，存在两个服务器A，B，每个服务器存储L长度的初始化零的字符串。假设在这两个服务器不共谋的情况下，用户对数据的隐私读写可按照下面方式进行。假设用户想向数据库中第l行写入一个“1”，客户端首先随机生成L长度的字符串r，发送到服务器A，然后向服务器B发送$r \oplus e_l$，e_l是一个在位置l为1，其他位置为零的长度为L的字符串。在收到写入请求受，服务器将字符串 XOR运算后写入共享的数据库。在n个写入操作过后，数据库在服务器A处状态是：

$$d_A = r_1 \oplus ... \oplus r_n$$

在服务器B出的状态是：

$$d_B = (e_{l1} \oplus ... \oplus eln) \oplus (r_1 \oplus .... \oplus r_n)$$

这样在周期结束时，服务器A、B将两者的状态共享出来发布明文的数据库。

只要两个服务器不共谋，协议则满足写入隐私的要求。但是这个协议有两个明显的缺点。首先的限制是此协议的带宽利用很低效，如果上百万的用户在一个周期内请求想数据库写入数据，那么数据库的长度至少要变成上百万比特，用户为了给数据库的单个比特取反也都需要发送上百万比特的数据。第二个缺点是这个协议无法抵抗恶意的破坏攻击。一个随机的字符串就可以破坏整个数据库的可用性，并且由于正常请求也近似于随机化，故这类攻击还难以检测。因此如何更加有效的利用带宽并且检测破坏攻击时接下来我们的贡献。

E. 冲突

把通信开销和抵抗破坏攻击放在一边，上述协议中还有一个严重问题。即即使在所有客户端都是诚实的情况下，虽然用户数据的写入位置看起来是随机的，但是存在两个用户将数据写入到同一个位置的情况，这样最终恢复出来的消息就会变成： $m_A \oplus m_B$，显

然A和B的消息都不能够完整的恢复出来。

为了解决这个问题，一个直观的想法是将数据库的写入空间设计的足够的大，使得任意两个用户写入同一位置的概率足够的小。少数写入失败的用户可在下一个周期再次提交写入请求。根据计算，每次写入成功的概率为：

$$E[SuccessRate] = \frac{n}{m} Pr[O_i^{(1)}] \approx 1 - \frac{m}{n} + \frac{1}{2}(\frac{m}{n})^2$$

其中n为数据库的行数， m为单个周期用户的写入请求数，$O_i^{(1)}$表示第i行只有一个写入请求。由上式可知，如果我们希望由95%的成功率，那么我们需要$n \approx 19.5m$。如$m = 2^{10}$个请求，那么需要大概20000个可供写入的数据库位置。

处理冲突。我们还有一个方法来尽量的缩减所需要的数据库的大小。想法很简单：如果在两次（同时对一个位置进行三次写入的概率极小）写入同一位置的情况下，通过适当的编码方案，能够有效的恢复出用户各自写入的原始数据。为了保存更多的信息，我们将每行虽保存的数据加倍，值得指出的是，虽然每行的数据量增加了，但是数据库的总行数在此技术下能够缩减超过两倍。假设在一行中所存储的是(m_A, m_A^2)，那么在发生冲突时假设是服务器 B在同一位置写入了(m_B, M_B^2)。因此最终的输出数据是

$$S_1 = m_A + m_B(modp) S_2 = m_A^2 + m_B^2(modp)$$

由观察可知

$$2S_2 - S_1^2 = (m_A - m_B)^2(modp)$$

这样我们可以得到$m_A - m_B$，结合存储的$m_A + m_B$，能够完整的恢复出原始消息$m_A$fim_B。计算两个值同时写入同一行的概率是：

$$Pr[O_i^{(2)}] = \frac{m}{2n^2}(1 - \frac{1}{n})^{m-2}$$

则最终的期望成功概率是：

$$E[SuccessRate] \approx 1 - \frac{1}{2}(\frac{m}{n})^2 + \frac{1}{3}(\frac{m}{n})^3$$

即现在达到95%的成功率只需要$n \approx 2.7m$，这个结果比前文19.5小了很多。

IV. 使用分布式点函数来改进带宽利用率

稻草人协议的低效是由于用户必须向每个服务器发送一个L比特的向量来在数据库中反转甚至只是一个比特。为了降低这个$O(L)$的通信开销，我们使用由隐私数据检索所启发的技术。

PIR所解决的问题本质刚好与我们这里的场景相反。在PIR中客户端必须在复制的数据库中读取一比特但是并不会让服务器知道所被检索数据的所存放的位置。在我们的设定中，客户端需要向数据库中写入一比特数据而不让数据库本身知晓数据写入的位置。已经有研究者将这类问题归结为私有信息存储协议。

PIR允许客户端将他的请求拆解开发送给多个服务器，并且保证：（1）分解后的请求的部分子集不会暴露用户所写入位置的信息。（2）用户发送的请求长度远小于数据库的长度。我们采用的来构建满足条件的PIR方法的核心技术是分布式点函数。一个点函数的定义如下：一个固定的正整数L和有限域$\mathbb{F}$，对于所有的$\ell \in \mathbb{Z}_L, m \in \mathbb{F}$，点函数是$P_{\ell,m} : \mathbb{Z}_L \to \mathbb{F}$，除$P_{\ell,m} = m$之外，对于所有的$\ell \neq \ell', P_{\ell,m'} = 0$。即除了在位置$\ell$之外，函数的所有值都为零。这即是说，点函数$P_{\ell,m}$对所有的输入都输出零，为一个例外是在输入为$\ell$时函数输出m。举例来说，如果$L = 5, \mathbb{F} = \mathbb{F}_2$，点函数$P_{3,1}$的输入为$(0,1,2,3,4)$求的的返回值为$(0,0,0,1,0)$。

分布式点函数，同上，对于一个(s,t)的分布式点函数由一对算法组成，其中(s,t)表示在s个服务器中最多只有t个服务器之间进行共谋:

$Gen(\ell, m) \to (k_0, \cdots, k_{s-1})$。在域$\mathbb{Z}_L$中给出一个$\ell$和域$\mathbb{F}$中给出一个$m, Gen$输出s个密钥。

$Eval(k, \ell') \to m'$。给定一个由Gen中输出的密钥k，和相关的索引$\ell' \in \mathbb{Z}_L$，函数返回$\mathbb{F}$中的值m'。

满足$\forall l, l' \in \mathbb{Z}_L, \forall m \in \mathbb{F}$

$$\Pr[(k_0, \cdots, k_{s-1}) \leftarrow \mathrm{Gen}(l, m) : \sum_{i=0}^{s-1} \mathrm{Eval}(k_i, l') = \mathrm{P}_{l,m}(l')] = 1$$

一个（s,t）的分布式函数使用Gen算法生成s个密钥，用户分别将密钥k_i发送到k_i，这些密钥同时对消息$m \in \mathbb{F}$和写入目标索引$\ell \in \mathbb{Z}_L = \{0,1,2,3,;L-1\}$。当服务器$s_i$接收到密钥 k_i时，服务器对L行中的每一行索引ℓ'运算$Eval(k, \ell')$得到$m' \in \mathbb{F}$，这个值如最初的协议一样与已经保存在服务器的值相加后更新数据库状

态。

换句话说，如果用户将$Gen(\ell, m)$生成的密钥$(k_0, \cdots, k_{s-1})$分别发送给服务器$s_0, \cdots, s_{s_1}$，服务器计算得到向量$\vec{m_i}' = \begin{pmatrix} \mathrm{Eval}(k_i, 0) \\ \mathrm{Eval}(k_i, 1) \\ \cdots \\ \mathrm{Eval}(k_i, L-1) \end{pmatrix}$。当所有服务器一起共享各自的$\vec{m_i}'$，相加可得到$\vec{m}'$：$\sum_{i=0}^{s-1} \vec{m_i}' = \begin{pmatrix} P_{l,m}(0) \\ P_{l,m}(1) \\ \cdots \\ P_{l,m}(L-1) \end{pmatrix}$

A. 矩阵技巧

直接应用分布式点函数技术并不为直接带来通信开销的减少，本节我们的主要目的是使用写入PIR技术降低系统的通行开销。现在的问题是直接使用DPF会得到稻草人协议一样的与数据库大小成正比的通信开销，受到计算型PIR的启发，本质上来说用户发送$O(L)$长度的消息是为了在保证在服务器看来用户在任意位置写入数据得概率相等，用户写入数据得问题可直接归结为寻址的问题。以往的线性寻址方案当然需要很大的性能开销（特别是数据库大小特别大的时）。那么我们可以将整个线性的空间转化为一个矩阵，这样我们使用坐标就可以唯一的进行寻址。

我们将L行的数据表$\vec{t}$转化成一个x行y列的矩阵$xy \geqslant L$，即表示$x \times y$的矩阵足够大来包含数据表$\vec{t}$中的所有数据项。当然做法并不是将原始数据项直接拷贝到一个矩阵中去，而是将其存储在一个巨大的有xy元素的表中。然后我们使用$\vec{t}[l_x y + l_y]$来进行对位于ℓ_x行ℓ_y列的数据单元寻址。

这样做之后很明显需要的发送的数据量$x = O(\sqrt{L}), y = O(\sqrt{L})$ 通过这种方法,针对最初的协议来说要传送密钥的长度大约263 KB,而不是1 GB。这不仅是相当令人印象深刻的节省了客户端和服务器之间的带宽，而且甚至能够使得大多数资源受限的移动应用程序场景也能够使用这种匿名技术。

接下来的问题就是如何在最简单的双服务器场景下定义$Gen(\ell, m)$和$Eval(k, \ell')$函数。函数$Gen(\ell, m)$生成两个密钥$\vec{k_0}\vec{k_1}$。服务器s_0使用$\vec{k_0}$求得矩阵M_0'，服务器s_1使用$\vec{k_1}$求得矩阵M_1'。将这些矩阵加到它们自己的内部状态(矩阵)中后，服务器共同来将更新(分布式)数据库。

$$M_0' + M_1' = \begin{pmatrix} 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & m_{l_x l_y} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 \end{pmatrix}$$

$(\ell_x, \ell_y) \in \mathbb{Z}_x \times \mathbb{Z}_y, \ell = \ell_x y + \ell_y$。

Gen算法： $\mathrm{Gen}(l, m) \to (\vec{k}_0, \vec{k}_1)$

首先用户计算$l_x \in \mathbb{Z}_x l_y \in \mathbb{Z}_y$满足$\ell = \ell_x y + \ell_y$。接着随机生成$\vec{b_0} = (b_0, \cdots, b_{l_x}, \cdots, b_{x-1}) \xleftarrow{R} \{0,1\}^x$和$\vec{b}_1 = (b_0, \cdots, b_{l_x}^*, \cdots, b_{x-1})$。两只之间的差别仅在于$b_{\ell_x}$。接着随机生成种子$\vec{s_0} := (s_0, \cdots, s_{l_x}, \cdots, s_{x-1}) \xleftarrow{R} \mathbb{S}^x$和一个随机种子$s_{l_x}^* \xleftarrow{R} \mathbb{S}$来得到$\vec{s_1} := (s_0, \cdots, s_{l_x}^*, \cdots, s_{x-1})$。定义$m \cdot \ell_y \in \mathbb{F}$为一个只在$\ell_y$处为m，其他位置为0的向量。设置$\vec{v} \leftarrow m \cdot \vec{e}_{l_y} + \mathrm{G}(\vec{s_0}[l_x]) + \mathrm{G}(\vec{s_1}[l_x])$。则Gen的输出为

$$k_A = (b_A, s_A, v), k_B = (b_B, s_B, v)$$

。

事实上，v的形式如下：

$$\begin{pmatrix} v_0 \\ \vdots \\ v_{l_y} \\ \vdots \\ v_{y-1} \end{pmatrix} \leftarrow \begin{pmatrix} 0 \\ \vdots \\ m \\ \vdots \\ 0 \end{pmatrix} + \begin{pmatrix} G(s_{l_x})[0] \\ \vdots \\ G(s_{l_x})[l_y] \\ \vdots \\ G(s_{l_x})[y-1] \end{pmatrix} + \begin{pmatrix} G(s_{l_x}^*)[0] \\ \vdots \\ G(s_{l_x}^*)[l_y] \\ \vdots \\ G(s_{l_x}^*)[y-1] \end{pmatrix}$$

可以看出，我们将处于ℓ_y的消息m成功的混淆到整个向量中，服务器无法得知具体哪一行包含消息m。那么我们的Eval函数的重点就是如何将ℓ_x的值计算出来。$Eval(\vec{k}, \ell')$。服务器在收到用户发送过来的密钥k后先按照密钥格式解析$(\vec{b}, \vec{s}, \vec{v}) := \vec{k}$，计算$l_x' \in \mathbb{Z}_x, l_y' \in \mathbb{Z}_y$，根据得到的$\ell_x'$生成$\vec{g} \leftarrow \mathrm{G}(\vec{s}[l_x'])$，最后$m' \leftarrow (\vec{g}[l_y'] + \vec{b}[l_x']\vec{v}[l_y'])$。对于数据的每一个$\ell'$我们将其计算出矩阵的位置行坐标$\ell_x'$和列坐标$\ell_y'$，然后将位于$\ell_x'$的种子扩展为长度y，由公式$l' = l_x' y + l_y'$可知寻址的计算出的m'为$(l_x', l_y')$，当位于$\ell_x'$为1时将$\vec{g}[\ell_y']$加到$\vec{v}[\ell_y']$。

举例来说，假设总共由6行，我们想要在第3行

第4列写入数据，那么这时s_0生成

$$\left(\begin{pmatrix}1\\0\\1\\1\\0\\0\end{pmatrix},\begin{pmatrix}s_0\\s_1\\s_2\\s_3\\s_4\\s_5\end{pmatrix},\begin{pmatrix}\mathrm{G}(s_0)+\vec{v}\\\mathrm{G}(s_1)\\\mathrm{G}(s_2)+\vec{v}\\\mathrm{G}(s_3)+\vec{v}\\\mathrm{G}(s_4)\\\mathrm{G}(s_5)\end{pmatrix}\right)$$

s_1计算

$$\left(\begin{pmatrix}1\\0\\1\\0\\0\\0\end{pmatrix},\begin{pmatrix}s_0\\s_1\\s_2\\s_3^*\\s_4\\s_5\end{pmatrix},\begin{pmatrix}\mathrm{G}(s_0)+\vec{v}\\\mathrm{G}(s_1)\\\mathrm{G}(s_2)+\vec{v}\\\mathrm{G}(s_3^*)\\\mathrm{G}(s_4)\\\mathrm{G}(s_5)\end{pmatrix}\right)$$

此时$\vec{v} = m \cdot \vec{e}_4 + G(\vec{s}_0[\ell_x] + G(\vec{s}_1[\ell_x]))$。容易看出，在v中我们加入了$\ell_y$的相关信息，在生成的随机比特向量b中我们加入了$\ell_x$的信息，结合两者我们就能够实现$O(\sqrt{L})$的高效的PIR写入操作。

V. 对抗破坏攻击

在稻草人方法中我们指出本系统主要解决原始方案中存在的两个问题：第一是通信开销过大，特别是在数据库比较大的时候。然后是这样的方法很容易收到来自用户/服务器的破坏攻击，即使是一次随机的写入都会导致整个数据库的污染而不可用。整个系统现在看来可以认为是一个多方计算协议，用户只需要生成k个密钥来隐私请求写入数据，而服务器的输入是一个密钥k_i，所有服务器加在一起为整个协议的输出，输出为一个单独的有效比特。在这里我们认为服务器是可用的，因此我们不考虑来自服务器的恶意攻击来试图操纵系统的最后输出。因为这样的攻击只会是破坏了数据的状态或者使得系统对某个用户来说变得不可用，这都是属于拒绝服务攻击的范畴，因此我们不考虑此类攻击。

我们考虑的是在面对恶意服务器的时候如何保证用户写入的隐私，即参与协议的服务器无法得知用户私密输入的额外信息。作者在文中提出了两种协议来检测用户写入请求的合法性，第一个协议只需要很少的计算量，但是需要引入一个可信的第三方验证者。还有一个方案相对来说计算开销较大，使用零知识证明的方法，但只要在系统中存在一个诚实的服务器这种方法都能够保证系统的保证系统安全。

A. 三方协议

三方协议引入一个不与任何服务器合谋的审计服务器，审计服务器所做的计算非常简单，即哈希和比较操作。协议中总共有三个参与者，服务器A，B和一个审计服务器。服务器A的输入是一个私密的输入$v_A \in \mathbb{F}^n$，服务器B的输入是一个私密输入$v_B \in \mathbb{F}^n$。审计服务器的输入为v_A, v_B，输出为0或1来表示v_A, v_B是否只是在一个位置上不同其他位置都相同，这个算法作者称为大致相同算法。协议过程具体如下：

- 1)服务器A和B使用投硬币协议来从成对的独立哈希函数H中取得n个哈希函数$h_0, \cdots, h_{n-1}$。并且服务器之间对一个随机偏移达成一致。
- 2)服务器A对每一个索引值$i \in \{0, \cdots, n-1\}$计算值$m_i \leftarrow h_i(v_A[i])$。然后将$(m_f, m_{f+1}, \cdots, m_{n-1}, m_0, \cdots, m_{f-1})$
- 3)服务器B重复上一步骤。
- 4)当两者提交给审计服务器的值大致相同时，审计服务器同时对A，B返回1，不然则同时对它们返回0。

需要注意的一点是，当服务器想要来验证密钥中的向量v时，做法稍微有点不同，此时服务器A，B分别给审计服务器发送：

$$u_A = \sum_{i=0}^{x-1} G(s_A[i]) \quad u_B = v + \sum_{i=0}^{x-1} G(s_B[i])$$

参考文献

[1] Daniel J Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: Elliptic-curve points indistinguishable from uniform random strings. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, pages 967–980. ACM, 2013