

保护搜索内容的字符串模式匹配算法研究

王健旭, 李睿, 李银

(东莞理工学院, 广东 东莞 523808)

摘要: 高效的在线字符串模式匹配算法对云数据库检索至关重要, 然而搜索内容的泄露会威胁用户隐私。现有的字符串模式匹配算法没有考虑用户搜索内容的保护, 可搜索加密方案虽然可以保护用户的搜索内容, 但存在索引构建代价大、检索效率低等问题。因此, 提出了两种保护用户搜索内容的模式匹配算法: 基于分布式点函数的模式匹配 (pattern matching based on distributed point function, PMDPF) 算法和基于分布式点函数的跳跃式模式匹配 (jumping pattern matching based on distributed point function, JPMPDPF) 算法。PMDPF算法利用指纹函数以及分布式点函数构造模式串真值表, 并分发给两台独立的服务器, 把搜索中字符对比操作转换为查表操作, 从而保护搜索内容。为了提升搜索效率, 提出了JPMPDPF算法。通过字符跳转, JPMPDPF算法以泄露更多信息为代价, 其搜索效率比PMDPF算法平均提高了约 m 倍, 其中 m 为搜索内容长度, 同时显著降低了因指纹函数碰撞而导致的误判的概率。实验结果表明, PMDPF算法的搜索效率比基于指纹函数的经典算法提高约5%, 并优于现有的可搜索加密方案, PMDPF算法的搜索耗时在搜索内容长度为4时是JPMPDPF算法的4.2倍。

关键词: 模式匹配; 指纹函数; 私有信息检索; 分布式点函数

中图分类号: TP393

文献标志码: A

DOI: 10.11959/j.cnki.cn2096-109x.2024061

Efficient algorithms for string pattern matching to protect search content

WANG Jianxu, LI Rui, LI Yin

Dongguan University of Technology, Dongguan 523808, China

Abstract: Efficient algorithms for online string pattern matching have been deemed crucial for cloud database retrieval. However, the potential leaking of search content has been recognized as a threat to users' privacy. Existing string pattern matching algorithms have not been designed to consider the protection of users' search content. Although searchable encryption schemes can protect users' search content, they have been associated with high index construction costs and inefficient retrieval. Two pattern matching algorithms were proposed to protect users' search

收稿日期: 2024-03-19; 修回日期: 2024-07-09

通信作者: 李睿, ruili@dgut.edu.cn

基金项目: 国家重点研发计划 (2021YFB3101303); 国家自然科学基金 (61972089, 62372107)

Foundation Items: The National Key Research and Development Project of China (2021YFB3101303), The National Natural Science Foundation of China (61972089, 62372107)

引用格式: 王健旭, 李睿, 李银. 保护搜索内容的字符串模式匹配算法研究[J]. 网络与信息安全学报, 2024, 10(4): 159-174.

Citation Format: WANG J X, LI R, LI Y. Efficient algorithms for string pattern matching to protect search content[J]. Chinese Journal of Network and Information Security, 2024, 10(4): 159-174.

content: PMDPF (pattern matching based on distributed point function) and JPMPDF (jumping pattern matching based on distributed point function). In the PMDPF algorithm, a pattern string was provided to protect the search content, and a distributed point function combined with a fingerprint function was utilized to construct true value tables, which were then sent to two independent servers. Character comparison operations were transformed into table lookup operations to conceal the characters that needed to be checked. To enhance search efficiency, a jump pattern matching algorithm (JPMPDF) was proposed, based on the distributed point function. In this approach, character skipping was employed to induce the JPMPDF algorithm, which achieved a significant improvement in search efficiency over the PMDPF algorithm, albeit with the trade-off of revealing more information. On average, the search efficiency of JPMPDF was increased by approximately m -fold, where m represented the length of the pattern string. Concurrently, there was a significant reduction in the probability of misjudgments due to collisions within the fingerprinting function. The experimental results demonstrate that the search efficiency of the PMDPF algorithm is approximately 5% higher than that of the classical algorithm based on the fingerprint function. It outperforms existing searchable encryption schemes. The search time of the PMDPF algorithm is found to be 4.2 times that of JPMPDF when the length of the search content is 4.

Keywords: pattern matching, fingerprint function, private information retrieval, distributed point function

0 引言

模式匹配是计算机科学领域的典型问题,其在生物基因序列分析、数据库检索、文本处理等领域具有重要作用,具有速度快通信开销小的特点,在明文数据库检索中被广泛应用。目前在线字符串模式匹配算法主要分为基于字符比较的算法^[1-3]、基于自动机的算法^[4]和基于位并行的算法^[5]等。然而,已有模式匹配算法虽然能很好地解决大规模明文数据的检索问题,但均会泄露搜索内容。搜索内容的泄露会导致两个问题的出现。

1) 根据搜索内容可以推断出用户隐私信息。例如通过收集用户在浏览器中所搜索的内容,攻击者就可以利用数据挖掘技术推知用户的年龄、性别和教育背景等相关信息^[6],服务提供商甚至可能实现人脸信息、身份信息甚至亲属关系的匹配,导致个人隐私信息和财产安全受到严重威胁。

2) 基于垂直分割的隐私保护方案失效。垂直分割是分布式环境中保护数据隐私的一种有效方法,该方法依据明确的隐私约束将数据集按属性垂直划分为特定子集以满足特定的隐私需求^[7],攻击者在进行搜索时,可以根据搜索内容推断出子集间属性的关联性,从而破坏隐私约束,导致隐私保护方案失效。

保护用户搜索内容的主要技术路线包括可搜

索加密方案^[8-9]和私有信息检索 (private information retrieval, PIR)^[10-13],两类方案均支持存储内容的搜索。然而,可搜索加密方案通常需要对服务器存储的数据本身进行大量的预处理,构建与数据规模成正比的索引内容,且搜索速度较慢,当数据规模较大时,难以满足用户实时在线搜索的需求。例如安全最邻近算法^[14]和隐藏谓词加密算法^[15],它们需要对数据库存储的内容进行预处理并存储结果,这可能花费大量的额外空间且不易于扩展,而本文提出的算法不需要预先对数据库中内容进行任何预处理,更适合在不断扩展的明文数据库中执行搜索,可以解决在动态明文环境中保护用户的搜索内容的问题。其次,可搜索加密方案需要对数据整体进行加密或者提取数据中的关键词后加密,既不支持对除密文索引外数据内容的搜索,也无法准确判断原数据中的内容是否包含用户搜索内容以及包含个数,只能判断二者的相似程度或二者是否完全一致,而本文研究的算法可以准确判断。另外,基于被查询数据构建树和索引的字符串搜索算法是离线搜索^[16],而本文提出的基于搜索内容的字符串搜索算法则为在线搜索,更适用于不断更新的明文数据库。最后,这些方案实现的是密文搜索,并不适用明文的搜索。加密数据流模式匹配^[17-18]是可搜索加密方案的拓展,通常采用公钥加密以字符为最小单位的数据,利用私钥构建陷门,解决了传统可搜索加密方案只能搜索关键词

的问题，但密文数据占据的空间是明文数据的近百倍^[18]，且查询效率低于明文模式匹配算法，而本文提出的方案在实现可以搜索任意内容的基础上无须占据额外的存储空间，查询的效率也略优于明文模式匹配方案。传统的PIR方案^[10-11]虽然可以实现隐藏搜索意图的目的，但需要用户知道所搜索的内容在服务器中的具体位置。而支持关键词搜索的PIR方案^[12-13]通常需要用户与服务器进行2轮及以上的通信，并且不支持除关键词以外内容的搜索。

为解决在大规模明文数据库中保护搜索内容的实时在线查询问题，本文提出了基于分布式点函数的模式匹配（pattern matching based on distributed point function, PMDPF）算法，分布式点函数（distributed point function, DPF）^[19]是安全多方计算领域的重要工具之一，广泛应用于解决该领域的相关问题^[20]。本文通过指纹函数^[21]和DPF设计了基于搜索内容的点函数真值表，将传统模式匹配过程中服务器按字符比对是否一致转化为查询真值表并进行累加运算，用户处理返回值后得到搜索结果。在这一过程中，用户无须将搜索内容上传至服务器，服务器也不必构建额外索引，即可完成搜索过程。相较PMDPF算法，基于KR的经典模式匹配算法在发生哈希冲突时，需要按位比对模式串，而查表的开销相对较小，PMDPF算法仅需要查表，因此PMDPF执行效率略快于基于KR的模式匹配算法，实验结果证明PMDPF算法比KR算法快约5%。

为了更好地满足实时在线搜索，本文在PMDPF方案的基础上结合模式匹配算法中字符跳转^[3]的思想提出了基于分布式点函数的跳跃式模式匹配算法，根据字符在搜索内容中所处的位置设计了字符移位表，通过该表实现跳跃式搜索，减少了搜索过程中服务器的计算开销与查表开销。但该表会泄露搜索内容的部分字符位置信息，因此本文设计了混淆和隐藏两种操作，其中混淆操作将不存在于搜索内容中的字符伪装为存在，而隐藏操作将存在于搜索内容中字符隐藏，最终在提高了方案执行效率的同时保护了字符的位置信息。实验结果表明，JPMDPF搜索效率比PMDPF平均提高了约 m 倍，其中 m 为搜索字符

串长度。本文提出的算法与现有方案在通信轮次、是否支持关键词搜索和是否支持对任意内容搜索几个方面进行了对比，如表1所示，JPMDPF和PMDPF算法在通信轮次和查询内容的范围均优于其他方案。特别地，徐琳等提出的方案^[8]仅需要1轮通信，且可以实现对数据库中任意内容的搜索，但其搜索速度较慢，例如当服务器存储内容为长度为1 024个字符的字符串且用户搜索内容为长度为8个字符时，本文提出的PMDPF算法执行平均需要0.1 754ms，JPMDPF算法执行平均仅需要0.035ms。以两个服务器作为参与方为例，PMDPF和JPMDPF算法的执行过程如图1所示。

表1 方案对比

Table 1 Comparison of schemes

算法	通信轮次	支持关键词搜索	支持任意内容搜索
Gilboa ^[19]	2	是	否
Yu ^[13]	1	是	否
Xu ^[8]	1	是	是
PMDPF	1	是	是
JPMDPF	1	是	是

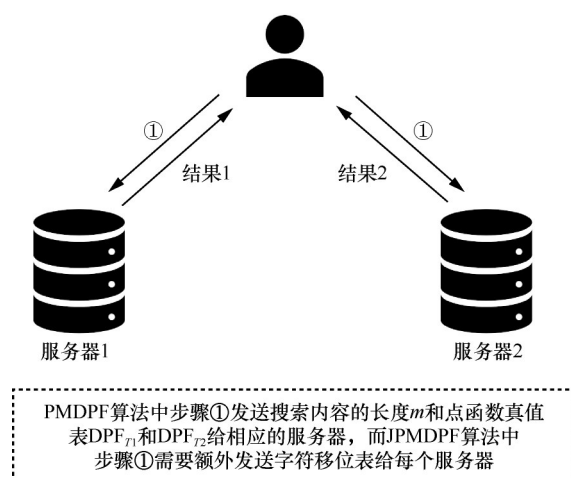


图1 PMDPF和JPMDPF算法的执行过程

Figure 1 The execution process of PMDPF and JPMDPF algorithms

本文的主要贡献如下。

1) 提出了PMDPF算法，解决了实时明文数据库中在线检索时保护搜索内容的问题，用户与服务器间仅需一轮通信，传输带宽与存储数据规

模无关, 且算法效率优于经典指纹搜索算法。

2) 提出了 JPMDPF 算法, 在保留 PMDPF 功能的基础上进一步提高了执行效率, 且显著降低了因指纹碰撞所产生假阳性结果概率。

1 背景知识

1.1 指纹函数

Karp 等^[22]在 1987 年提出了高效的字符串模式匹配算法, 该算法将字符串转换为以字符为系数的多项式, 通过判断多项式的结果是否相等替代了比对两个字符串每一位是否一致。字符串输入该式后的输出结果被称为字符串的指纹值 Fingerprint, 相应的函数称为指纹函数。指纹函数结合滑动窗口的方法能够有效地提高算法效率。Breslauer 等^[21]在 2014 年基于指纹函数设计了实时模式匹配算法, 然而, 指纹函数存在碰撞导致出现假阳性匹配结果。

指纹函数具体定义如下:

$$FP(s_i) = \sum_{j=1}^m s_i[j] \cdot r^j \bmod p \quad (1)$$

其中, $s_i[j]$ 为第 i 个字符串 s_i 的第 j 个字符, m 为字符串 s_i 的长度, p 为素数, r 为小于 p 的任意正整数。下一个字符串 s_{i+1} 的指纹值则可以通过式 (2) 进行计算:

$$FP(s_{i+1}) = FP(s_i) \cdot r^{-1} - s_i[1] + r^m \cdot s_{i+1}[m] \bmod p \quad (2)$$

因此, 该算法可以利用先前字符串的指纹值来快速计算下一个连续字符串的指纹值, 这是相应搜索算法高效执行的关键。

1.2 加法秘密共享

加法秘密共享 (additive secret sharing, ASS) 由 Goldreich 等^[23]在 1987 年首次提出, 最初应用于布尔电路层面, 秘密数据以异或的形式进行拆分。设有 n 个参与方 $P_1, \dots, P_n$, 秘密拥有者持有秘密 s , 加法秘密共享的过程可以简单描述如下。

1) 在秘密生成阶段, 秘密拥有者在定义域中随机选取元素 $s_1, \dots, s_{n-1}$, 然后按照式 (3) 计算 s_n :

$$s_n = s - \sum_{i=1}^{n-1} s_i \quad (3)$$

2) 在秘密分发阶段, 将秘密份额 s_i 发送给参与方 P_i ; 在秘密重构阶段, 任意参与方收到所有其他的秘密份额后即可计算得到秘密 s 。加法

秘密共享是分布式点函数成立的必要条件。

1.3 分布式点函数

DPF 由 Gilboa 等于 2014 年首次提出^[19], 广泛应用于解决私有信息检索和不经意传输^[20]等相关问题。点函数的定义如式(4)所示。

$$f_a^b(x) = \begin{cases} b, & x = a \\ 0, & \text{其他} \end{cases} \quad (4)$$

式(4)中函数定义域为 $\{0, 1\}^l$, 值域为 $\{0, b\}$ 。DPF 将 $f_a^b(x)$ 的真值表排列为长度 2^l 的向量, 并应用两方加法秘密分享方案将该向量分为两个长度 2^l 的向量, 作为函数 $f_a^b(x)$ 的两个秘密片段 k_1 和 k_2 。为了节约片段向量传输带宽, 采用子函数 Gen() 和 Eval() 实现向量的压缩和解压, 其通信开销为 $O(\log n)$, 其中 n 为函数真值表长度^[20]。简单起见, 假设用户和服务方已经完成了函数 Gen() 和 Eval() 的执行, 直接使用加法秘密共享的 2^l 向量作为点函数真值表。

1.4 安全模型

本文考虑在半诚实模型下保护用户的搜索内容。在半诚实模型中, 敌手是诚实且好奇的, 执行搜索过程的参与方完全遵守算法的执行规范, 但可以在算法的执行过程中根据算法的输出信息推断其他参与方的信息。出于实用性的考虑, 目前保护用户搜索内容的可搜索加密方案通常会允许交互过程泄露部分信息给服务器。在本文的研究中, 利用泄露函数 L 描述交互过程中泄露的信息。对于用户的任意搜索内容, 如果半诚实服务器仅能推断出泄露函数输出的信息, 且无法唯一确定搜索内容中任意位置的字符, 则称算法在泄露函数 L 的半诚实模型下保护了用户的搜索内容。

本文后续所使用的符号及其含义如表 2 所示。

2 PMDPF 算法

本节提出支持搜索任意内容的 PMDPF 算法。其核心思想是用户首先根据欲搜索内容 P 的指纹值 $FP(P)$ 构造相应点函数真值表, 并生成相应的秘密片段 k_0 和 k_1 , 分别将其与搜索内容长度 m 发送给两方独立的服务器, 服务器在接收到秘密片段 k_0/k_1 后, 将其解压恢复为真值表, 再从头到尾使用滑动窗口的方法计算出长度与搜索内容相等的子串 ($S[1:m], S[2:m], \dots, S[n-m+1:m]$) 的指纹

值，其中， n 为服务器存储的字符串长度。然后查询收到的真值表，累加表中对应的函数值，将累加结果发送给用户，用户收到所有服务器的返回值后通过计算得到搜索结果。算法包括用户端预处理和服务器端执行模式串搜索算法两个部分。

表2 符号说明
Table 2 Symbol explanation

符号名称	说明
Ω	服务器中存储的字符集合
P	用户欲搜索的字符串
S	服务器存储的字符串
$S[i]$	字符串 S 的第 i 个字符
m	P 的长度
$S[k:m]$	S 以第 k 个字符开始的长度为 m 的子字符串
T	用户可自定义的字符隐藏系数
$FP(S)$	字符串 S 的指纹值
DPF_{Tk}	服务器 k 持有的分布式点函数真值表

2.1 预处理

为了隐藏用户的搜索意图，本算法利用DPF生成用户搜索内容 P 的点函数真值表，生成真值表的过程称为用户端预处理过程。根据式(4)，如果用户搜索的字符串 P 的指纹值为 a ，并想查询其在服务器中的出现次数，那么仅需知道真值表中指纹值所对应函数值为 b 的结果在服务器中出现次数总和，若将 b 设置为1，那么服务器执行过程中所有函数值的累加和就是用户所搜索字符串的查询结果。同时，服务器所获得的点函数真值表是以加法秘密共享的秘密片段形式存在的，服务器无法通过该表确定用户的搜索内容。

以参与方是两个服务器为例，用户端预处理过程如算法1所示。构建真值表的具体过程为：①定义了发送给服务器的点函数真值表数组的长度至多有 p 行（即指纹值的取值范围）；②构造一个取值范围为 $[-5, 4]$ 的随机数 $temp$ 作为其中一个真值表的函数值，选取该范围的原因主要是直观且便于生成秘密片段；③判断当前的指纹值是否与用户搜索内容的指纹值相等，若相等，两个真值表中函数值的加和为1，若不相等，则相应的函数值加和为0。在 DPF_{T1} 和 DPF_{T2} 创建完成

后，用户将其发送给相应的服务器，压缩后的真值表通信开销为 $O(\log p)$ ，算法1需要执行 p 次点函数真值表中函数值的构建，算法1的时间复杂度为 $O(p)$ 。

算法1 构建DPF函数真值表

输入 搜索内容 P 的指纹值 $FP(P)$ ，素数 p

输出 点函数真值表数组 DPF_{T1} , DPF_{T2}

1) 生成2个长度为 p 的数组 DPF_{T1} , DPF_{T2} ;

2) for $i := 0$ to $p - 1$ do

3) $temp := (rand() \bmod 10) - 5$;

4) $DPF_{T1}[i] := temp$;

5) if $i == FP(P)$ then

6) $DPF_{T2}[i] := 1 - temp \bmod p$;

7) else

8) $DPF_{T2}[i] := 0 - temp \bmod p$;

9) end if

10) end for

11) Return DPF_{T1} , DPF_{T2}

2.2 模式串搜索算法

为了提高检索效率，本文不再使用DPF中的函数Eval单独计算不同的指纹值对应的结果真值，而是先解压出真值表，再进行查表操作。服务器在收到用户发送的表 DPF_{Tk} 后，根据该真值表执行搜索功能，该过程如算法2所示。

算法2 模式串搜索算法

输入 分布式点函数真值表 DPF_{Tk} ，素数 p ， r ($r \in \mathbb{Z}^+$, $r < p$)，服务器编号 k ($k = 1, 2$)，搜索内容长度 m ，服务器存储的字符串 S

输出 真值表 DPF_{Tk} 真值累加和 $result_k$

1) $n := |S|$; // n 为服务器存储的字符串长度

2) $temp := FP(S[1:m])$;

3) $result_k := DPF_{Tk}[temp]$;

4) for $i := 2$ to $n - m + 1$ do

5) $temp := ((temp - r \cdot S[i-1]) \cdot r^{-1} + r^m \cdot S[i-1+m]) \bmod p$;

6) $result_k := result_k + DPF_{Tk}[temp]$;

7) end for

8) Return $result_k$

搜索算法具体过程为：①初始化定义了服务器存储的字符串 S 的总长度 n 、字符串 S 的第一个长度为 m 的子串 $S[1:m]$ 的指纹值 $temp$ 和服务器 k

的搜索结果返回值 result_k ; ②定义剩余子串的起始位置的取值范围, 从 2 到 $n-m+1$, 共包含 $n-m$ 个子字符串; ③根据式(2)计算后续每一个子字符串的指纹值; ④通过查询相应的 DPF 表累加得到返回值 result_k 。服务器完成搜索操作后将返回值发送给用户, 其通信开销为 $O(1)$, 算法 2 的时间复杂度为 $O(n)$ 。

2.3 PMDPF 算法分析

用户收到两方服务器的返回值后, 将两返回值相加得到服务器中特定指纹值的字符串 ($\text{FP}(S[i:m]) = \text{FP}(P)$, $1 \leq i \leq n-m+1$) 出现的次数, 即在服务器中符合搜索条件的搜索结果。PMDPF 算法相较经典 KR^[19]算法省去了按字符比对的操作, 增加了查表操作, 可以保护用户的搜索内容, 服务器仅知道用户所搜索字符串 P 的长度 m 。

(1) PMDPF 算法正确性分析

本文从服务器搜索过程和返回结果两个方面给出算法的正确性说明。首先是搜索过程, 令服务器存储的 S 长度为 n , S 包含了 $S[1:m]$, $S[2:m]$, $\dots$, $S[n-m+1:m]$ 共 $n-m+1$ 个与 P 长度 m 相等的子串, 在搜索算法执行过程中, 服务器依次计算上述子串的指纹值, 而对于 $\forall 1 \leq i \leq n-m+1$, 当 $S[i:m] = P$, 总有 $\text{FP}(S[i:m]) = \text{FP}(P)$, 令 $v_i = \text{FP}(S[i:m])$, 根据算法 1 有式(5)成立。

$$\begin{cases} \text{DPF}_{T1}[v_i] + \text{DPF}_{T2}[v_i] = 1, & v_i = \text{FP}(P) \\ \text{DPF}_{T1}[v_i] + \text{DPF}_{T2}[v_i] = 0, & \text{其他} \end{cases} \quad (5)$$

那么两个服务器发送给用户返回值之和为:

$$\begin{aligned} & \sum_{i=1}^{n-m+1} \text{DPF}_{T1}[\text{FP}(S[i:m])] + \\ & \sum_{i=1}^{n-m+1} \text{DPF}_{T2}[\text{FP}(S[i:m])] = \\ & \sum_{i=1}^{n-m+1} (\text{DPF}_{T1}[\text{FP}(S[i:m])] + \\ & \text{DPF}_{T2}[\text{FP}(S[i:m])]) = t \end{aligned}$$

其中 t 为 $\text{FP}(P)$ 在服务器中出现的次数, 因此搜索过程一定不会错过符合搜索条件的 P 。

而对于返回结果, 当字符串指纹值的数量大于 p 时, 指纹值的计算是由较大的空间映射到较小的空间的过程, 必然会产生哈希碰撞, 即输入不同字符串输出相同的指纹值。而 KR 算法在发生 $\text{FP}(S[i:m]) = \text{FP}(P)$ 时, 需要进一步比较各相应

位置的字符是否完全一致。但本算法为了保护搜索内容, 没有将 P 上传给服务器, 也就无法进行类似的操作, 因此搜索结果会出现假阳性结果, 假阳性结果是指服务器返回的搜索结果中会出现字符串内容与用户搜索内容不一致, 但被计入搜索成功的结果中的误判情况。文献[21]给出了两个长度为 m 的不同字符串 s_i 与 s_j 具有相同指纹值的概率:

$$\Pr(\text{FP}(s_i) = \text{FP}(s_j) | s_i \neq s_j) \leq m/p \quad (6)$$

根据式(6), 服务器共计算了 $n-m+1$ 个长度 m 的字符串的指纹值, 为不失一般性, 设长度为 m 的不同字符串在服务器存储的字符串中出现概率均等, 则假阳性结果出现次数 $\mathcal{F} \leq (n-m+1) \cdot (m/p)$, 那么服务器执行搜索算法后返回结果的准确率为 $(n-m+1-\mathcal{F})/(n-m+1) \geq 1-m/p$, 而用户搜索内容的长度 m 往往远小于 p , 因此假阳性结果出现的概率较低, 返回结果接近真实结果。综上所述, PMDPF 算法保证了搜索结果的正确性。

(2) PMDPF 算法安全性分析

定义泄露函数 L_1 , $L_1(P)$: 输入用户搜索内容 P , 输出搜索内容的长度 m 。

定理 1 PMDPF 算法在泄露函数 L_1 的半诚实模型下保护了用户的搜索内容。

证明 假设服务器是拥有无限计算能力的攻击方, 用户选择任意两个长度相同的字符串 P^0 和 P^1 作为搜索内容并生成相应的点函数真值表发送给服务器。服务器知晓泄露函数 $L_1(P^0)$ 和 $L_1(P^1)$ 输出的搜索内容长度 m , 指纹函数的计算方法和执行搜索过程中的所有计算结果, 如果服务器无法区分 P^0 和 P^1 , 那么在服务器的视野中, 搜索内容中任意位置的可能字符为字符集中的任意字符, 保护了用户的搜索内容。因此定理 1 可以归约为证明 Adv 无法区分 P^0 和 P^1 。

令 b_i, b_j 是字符串 P^0 点函数真值表任意两个指纹值对应的函数值, 加法秘密共享该函数值, 有 $\text{Share}(b_i) \rightarrow (k_{1,i}, k_{2,i})$ 和 $\text{Share}(b_j) \rightarrow (k_{1,j}, k_{2,j})$, 且满足 $b_i = k_{1,i} + k_{2,i} \bmod p$ 和 $b_j = k_{1,j} + k_{2,j} \bmod p$, 由于秘密片段 $k_{1,i}$ 和 $k_{1,j}$ 均为 $\mathbb{Z}_p$ 随机元素, 对定义域中的任意元素 a , 有:

$$\Pr_{\text{Share}(b_i) \rightarrow (k_{1,i}, k_{2,i})} [k_{1,i} = \alpha] = \frac{1}{p}$$

$$\Pr_{\text{Share}(b_j) \rightarrow (k_{1,j}, k_{2,j})} [k_{1,j} = \alpha]$$

而对于另外的两个秘密片段 $k_{2,i}$ 和 $k_{2,j}$ 有：

$$\Pr_{\text{Share}(b_i) \rightarrow (k_{1,i}, k_{2,i})} [k_{2,i} = \alpha] =$$

$$\Pr_{\text{Share}(b_j) \rightarrow (k_{1,j}, k_{2,j})} [b_i - k_{1,i} = \alpha \bmod p] =$$

$$\Pr_{\text{Share}(b_i) \rightarrow (k_{1,i}, k_{2,i})} [k_{1,i} = b_i - \alpha] = \frac{1}{p} \quad (7)$$

类似地，可以得到：

$$\Pr_{\text{Share}(b_j) \rightarrow (k_{1,j}, k_{2,j})} [k_{2,j} = \alpha] =$$

$$\Pr_{\text{Share}(b_j) \rightarrow (k_{1,j}, k_{2,j})} [b_j - k_{2,i} = \alpha \bmod p] = \frac{1}{p} \quad (8)$$

式(7)和(8)表明 $\text{Share}(b_i)$ 和 $\text{Share}(b_j)$ 可以为定义域中的任意元素，其概率是均匀的 $1/p$ ，Adv 无法区分二者，类似地， P^1 点函数真值表中指纹值对应的秘密片段也为概率均匀的定义域中任意元素，因此 Adv 无法区分字符串 P^0 和 P^1 的秘密片段，也就无法通过秘密片段区分 P^0 和 P^1 的点函数真值表，而两次搜索过程的其余执行步骤完全相同，因此 Adv 无法区分 P^0 和 P^1 。假设 Adv 已知字符集 Ω ，Adv 通过搜索过程猜测用户所搜索字符串的概率与直接猜测字符串的概率相等，即 $1/|\Omega|^m$ ，因此不共谋半诚实服务器无法推断出真实的搜索内容，PMDPF 算法在泄露函数 L_1 的半诚实模型下保护了用户的搜索内容。

3 JPMDPF 算法

分析 PMDPF 算法的执行流程，可以发现算法的计算开销主要在于计算指纹值 $\text{FP}(S[i:m])$ ，所以减少需要计算的字符串指纹值的数量即可减少计算开销，提升算法的执行效率，因此本文提出了一种更为高效的 JPMDPF 算法。

本节提出了基于 PMDPF 算法的 JPMDPF 算法。两个算法整体执行的流程一致，但 JPMDPF 与 PMDPF 算法的区别包含以下几个方面：①用户需要额外构建一个长度为 $|\Omega|$ 字符移位表 Shift；②服务器在执行搜索算法时需要根据移位表 Shift 进行子字符串之间的跳转；③为提高计算效率，服务器在计算 $S[i:m]$ ($1 \leq i \leq n-m+1$) 的指纹值时需要根据从上一个 $S[j:m]$ ($i-m-2 < j < i$) 移动到当

前位置的跳转距离来决定计算的方式。

3.1 移位表构建

基于字符进行跳跃移位的思想最早被用于解决字符串的高效匹配问题并设计出快速排序算法 (QS, quick search algorithm) [3]，其目的是根据移位表跳转后让子字符串 $S[i:m]$ 恰好有一个字符与模式串 P 相应位置的字符匹配，若无法做到则跳转距离设置为 $m+1$ 。但根据此移位表，服务器可以明确了解模式串 P 内与移位表中所对应位置的字符信息，推测出用户的搜索内容。

为解决这一问题，本文将用户欲搜索的模式串 P 分成混淆 P^g 和隐藏 P^h 两个部分，并分别进行处理，这两部分的定义如下：

$$P^g: \forall i \in [1, \lceil (m+1)/2 \rceil], P[i] \in P^g$$

$$P^h: \forall j \in [\lceil (m+1)/2 \rceil, m], P[j] \in P^h$$

本文分别以长度为奇数和偶数的单词 “jack” 与 “lunch” 为例，混淆和隐藏两个部分的划分如图2所示。其中在 P^h 中所有出现过的字符在移位表中对应的跳转距离设置为1，而字符的跳转距离为1代表 P 的最后一个字符 $P[m]$ 可能为上述字符中的某一个，对于 P^h 中其他位置的字符，服务器仅知道字符的取值范围，却无法判断具体是哪一个字符，通过上述方法实现了保护 P^h 字符位置信息不被服务器直接获悉的目的。而在 P^g 中的字符则应用字符跳转的思想来进行跳转，选择 P^g 中的字符进行跳转有利于提高算法的执行效率，该方法跳转后的效果与 QS 算法类似，即跳转后尽可能有一个位置的字符匹配成功。

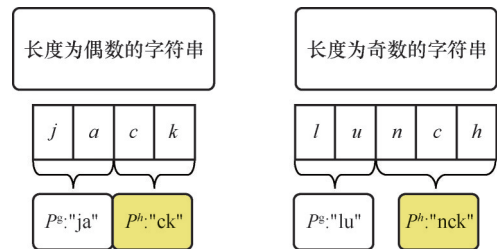


图2 不同长度字符串中混淆部分 P^g 和隐藏部分 P^h 划分
Figure 2 Dividing garbled P^g and hidden P^h parts in strings of different lengths

基于该思想，本文将字符集中的字符分为以下3类，并分别设置其在字符移位表中跳转距离。

1) 第I类字符为从未在 P 出现过的字符,其跳转距离均初始化为 $m+1$ 。

2) 第II类字符为未在 P^h 中出现但存在于 P^g 的字符,即 $c \in P^g \wedge c \notin P^h$,它们的跳转距离等于其距离 P 尾部的最小距离,即字符串长度 m 减去该字符在 P 中最后一次出现的位置。

3) 第III类字符为在 P^h 中出现过的字符,其跳转距离在处理 P^h 的过程中被设置为1。

服务器可以根据第I类和第II类的字符跳转距离来准确推断出 P^g 中的部分字符信息。因此为保护用户搜索内容中 P^g 的信息,且尽量减少对算法执行效率的影响,本算法采取将第I类字符串转化为第II类字符的方法,并将这种方法称之为尽可能混淆策略,具体过程为:随机选取 m 个I类字符,修改其跳转距离,将其跳转距离修改为II类字符,即把这些本不存在于 P 的字符伪装为存在,如果I类字符不足 m 个,则将其全部转化为II类字符;根据移位表, P^h 的字符可能III类字符中任意一个,而 P^g 中某位置对应的可能字符可以分为以下3种情况。

1) 在移位表中某位置对应的可能字符有两个,即该位置字符属于第I类字符,且完成了尽可能混淆操作。

2) 根据移位表,该位置可能的字符仅有一个,但该位置的字符在后续的位置中出现,根据 P^g 和 P^h 的执行过程,该字符所对应的跳转距离在后续执行过程中被修改,而该位置在尽可能混淆策略的执行过程中被其他字符所混淆。

3) 根据移位表,该位置可能的字符仅有一个,由于跳转距离为 $m+1$ 的字符已经全部被用光,无法完成混淆,移位表中该位置所对应的字符就是该字符本身。

当服务器在模式串 P 未知的条件下,仅根据移位表是无法判断该位置的字符属于上述哪一种情况,也就无法判断该位置的字符具体内容。因此实现了隐藏 P^g 中的字符所在实际位置的目的。

为了满足用户对搜索内容保密程度的不同需求,本文设置了用户可以自定义的字符隐藏系数 T , T 表示移位表中跳转距离为1的字符的最小数量,系数越大,泄露信息越少,搜索信息的

保护程度越高,但搜索效率越低。在混淆和隐藏两个部分处理完成后,把移位表中一些字符的跳转距离修改为1,使这些字符被混入隐藏部分 P^h 中,从而令服务器无法推断出这些字符的位置信息。 T 的取值范围为 $2 \leq T \leq |\Omega|$,根据本算法的设定,无论 m 的取值都至少有一个字符的跳转距离被设置为1,因而为了保护隐藏部分的字符位置信息, T 的下限为2(代表隐藏部分的任意位置至少有2个可能字符), T 的上限等于字符集的大小(所有字符的跳转距离均为1), T 值越大,字符集中的字符泄露的位置信息越少,服务器猜中搜索内容的概率越低,但可用于跳转的字符也越少,需要计算的子字符串指纹值 $FP(S[i:m])$ 越多,服务器的搜索效率越低,且当 T 值被设置为上限值时,搜索过程与PMDPF算法的搜索过程一致。基于以上分析,移位表构建算法如算法3所示。

算法3 移位表构建算法

输入: 用户欲搜索模式串 P ,模式串长度 m ,字符集 Ω ,字符隐藏系数 T 。

输出: 移位表数组Shift

1) 生成一个长度为 $|\Omega|$ 的数组Shift,且用 $Shift[\Omega[i] - \Omega[1]]$ 代表字符 $\Omega[i]$ 的跳转距离,并将数组中的值全部初始化为 $m+1$;

2) **for** $i := 1$ **to** $\lceil (m+1)/2 \rceil - 1$ **do**

3) $Shift[P[i] - \Omega[1]] := m - i + 1$;

4) **end for**

5) **for** $i := \lceil (m+1)/2 \rceil$ **to** m **do**

6) $Shift[P[i] - \Omega[1]] := 1$;

7) **end for**

8) **for** $i := 1$ **to** $\lceil (m+1)/2 \rceil - 1$ **and**

Shift表中值为 $m+1$ 的字符数量 > 0 **do**

9) $temp :=$ 值为 $m+1$ 的随机字符;

10) $Shift[temp - \Omega[1]] := m - i + 1$;

11) **end for**

12) **while** Shift表中值为1的字符数量 $< T$ **then**

13) $temp :=$ 选取Shift表中值为 $m+1$ 的随机字符,若没有满足条件的字符,则随机选取Shift表中值不为1的字符;

14) $Shift[temp - \Omega[1]] := 1$;

15) **end while**

16) **Return Shift**

算法3的具体过程为：①初始化字符集 Ω 中所有字符的跳转距离为 $m+1$ ；②按照字符跳转的思想设置移位表中存在于 P^s 的字符跳转的距离；③执行 P^h 部分的字符跳转距离，设置为该字符距离 P 尾部的最小距离；④执行尽可能混淆策略，不断地将III类字符在移位表中的跳转距离修改为II类字符的跳转距离，当没有符合条件的字符时，尽可能结束混淆过程；⑤根据用户设置的隐藏系数，优先将移位表中跳转距离为 $m+1$ 的字符修改其跳转距离为1，若不存在这种字符，则随机选取表中跳转距离不为1的字符，将其跳转距离修改为1。

当移位表创建完成后，用户将其与点函数真值表 DPF_{rk} 和搜索内容的长度 m 一起发送给对应的服务器 k ，等待服务器完成搜索操作，用户给服务器发送压缩后的真值表和字符移位表的通信开销为 $O(\log p + |\Omega|)$ ，算法3的时间复杂度为 $O(m)$ 。

3.2 模式串跳转搜索算法

相较PMDPF算法的模式串搜索算法，本节搜索算法额外增加了跳转的步骤，并根据上次跳转距离决定本次指纹值的计算方法，JPMDPF算法的模式串跳转搜索算法如算法4所示。

算法4 模式串跳转搜索算法

输入 分布式点函数真值表 DPF_{rk} ，移位表Shift，素数 p ， $r(r \in \mathbb{Z}^+, r < p)$ ，服务器编号 k ，欲搜索模式串长度 m ，服务器存储的字符串 S 。

输出 真值表 DPF_{rk} 真值累加和 $result_k$

```

1)  $n := |S|$ ;
2)  $temp := FP(S[1:m]);$ 
3)  $result_k := DPF_{rk}[temp];$ 
4)  $reshift := shift[S[m+1] - \Omega[1]]$ ; // re-
shift表示从上一位置跳转到当前位置跳转距离
5)  $i := 1 + reshift$ ;
6) while  $i < n - m$  do
7)   if  $reshift == 1$  then
8)      $temp := ((temp - r \cdot S[i-1]) \cdot r^{-1} + r^m \cdot$ 
 $S[i-1+m]) \bmod p$ ;
9)   else
```

10) $temp := FP(S[i:m]);$

11) **end if**

12) $reshift := Shift[S[i+m] - \Omega[1]]$;

13) $i := i + reshift$;

14) $result_k := result_k + DPF_{rk}[temp]$;

15) **end while**

16) **Return result_k**

模式串搜索算法的过程为：①初始化服务器存储的字符串的长度 n 、第一个长度为 m 的子字符串的指纹值、 DPF_{rk} 中对应该指纹值的首个返回值、从上个位置跳转当前位置的跳转距离和下一个子串的起始位置；②把当前子串的起始位置是否越界作为搜索终止的条件，当上一次跳转的距离为1时，当前子串的指纹值利用式(2)进行计算，而当跳转距离不为1时，当前子串的指纹值利用式(1)独立计算；③分别更新当前的跳转距离、下一个子串起始位置并与服务器返回结果的查表累加和，直到循环终止，服务器完成搜索算法，并将返回值发送给用户，返回值的通信开销为 $O(1)$ 。

算法4在最坏情况下共需要计算 $n - m + 1$ 个字符串的指纹值，因此算法4的最坏时间复杂度为 $O(n)$ 。

3.3 JPMDPF算法分析

令字符集为小写英文字母和字符隐藏系数 T 为4，JPMDPF跳转搜索算法示例如图3所示。

(1) JPMDPF算法安全性分析

定义泄露函数 L_2 ， $L_2(P, Shift, \Omega)$ ：输入用户搜索内容 P 、字符移位表Shift和字符集 Ω ，输出搜索内容的长度 m 和用户的可能搜索内容 $\mathcal{P} = \{P_1, P_2, \dots, P_m\}$ ，其中 $P_k (1 \leq k \leq m)$ 代表搜索内容 P 的第 k 个字符 $P[k]$ 的取值范围($P[k] \in \mathcal{P}_k$ 且 $|\mathcal{P}_k| \geq 2$)。

定理2 JPMDPF算法在泄露函数 L_2 的半诚实模型下保护了用户的搜索内容。

证明 泄露函数 L_2 输出的泄露信息包含了泄露函数 L_1 输出的搜索内容长度 m ，且点函数真值表的安全性在定理1中已给出证明，所以主要考虑字符移位表Shift对半诚实服务器推断用户搜索内容的影响。假设服务器已知移位表的构建方法和搜索内容中混淆部分 P^s 和隐藏部分 P^h 的划分方法，那么服务器可以将猜测 P 转换为猜测 P^s

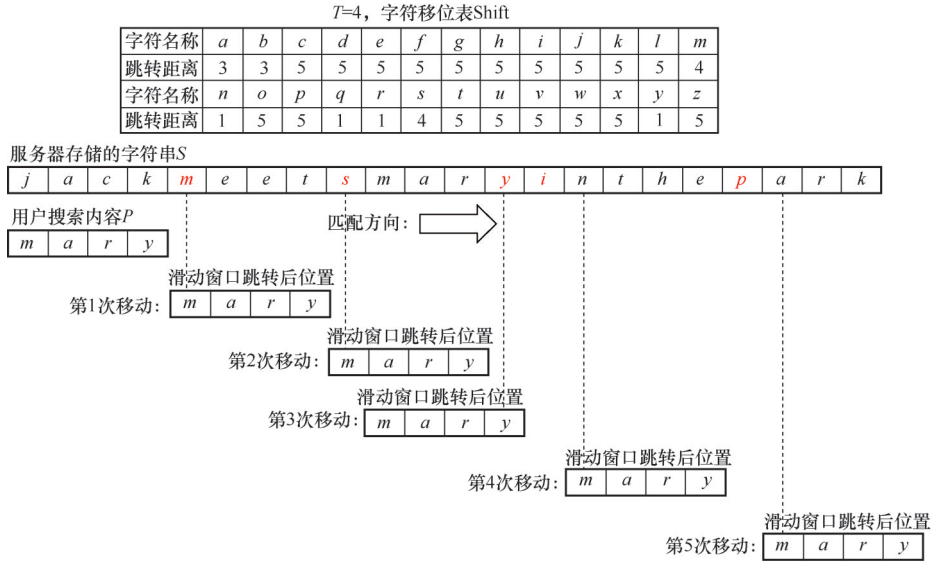


图3 JPMDPF 跳转搜索算法示例
Figure 3 Example of JPMDPF algorithm

和 P^h 。令 Adv 猜测的结果分别为 P^* 、 P^{g*} 和 P^{h*} ，根据移位表的设计和字符隐藏系数 T 的定义，有 $|\text{Shift}[c - \Omega[1]] = 1| \geq T$ ，且字符集 Ω 中的字符 c 在移位表中的取值为：

$$\text{Shift}[c] = \begin{cases} 1 \\ m - i + 1, i \in [1, \lceil (m+1)/2 \rceil] \\ m + 1 \end{cases} \quad (9)$$

对于混淆部分 P^g ，服务器 Adv 猜测位置 i ($i \in [1, \lceil (m+1)/2 \rceil]$) 存储的字符 c 可以分为以下 3 种情况。

1) 存在两个不同的字符 c_1 和 c_2 ，满足： $\exists c_1 \neq c_2, \text{Shift}[c_1 - \Omega[1]] = \text{Shift}[c_2 - \Omega[1]] = m - i + 1$ ，则根据移位表的构建过程，Adv 猜中 c 的概率为：

$$\Pr_{\text{Adv}}[c_1 = c] = \Pr_{\text{Adv}}[c_2 = c] = x$$

$$1/2 \rightarrow \Pr_{\text{Adv}}[P^g[i] = c] = 1/2$$

2) 仅存在一个字符 c_1 ，满足 $\text{Shift}[c_1 - \Omega[1]] = m - i + 1$ ，则位置 i 的字符是 c_1 或者 c_1 尽可能在混淆过程中被用于混淆的字符，即 $P^g[i] = c_1$ 或 $c \in P^h$ ，因此 Adv 猜中该位置存储 c 的概率为：

$$\Pr_{\text{Adv}}[P^g[i] = c] \leq 1/(T+1)$$

3) 对于字符集中的任意字符 c_1 ，都不满足 $\text{Shift}[c_1 - \Omega[1]] = m - i + 1$ ，则代表着 $c \in P^h$ 且尽可能在混淆过程中没有足够的字符用于混淆位置 i ，则 Adv 猜中 c 的概率为 $\Pr_{\text{Adv}}[P^g[i] = c] \leq 1/T$ 。

因此，Adv 猜测 P^g 位置 i 存储 c 的概率为：

$$\Pr_{\text{Adv}}[P^g[i] = c] \leq \max \{1/2, 1/(T+1), 1/T\} = 1/2,$$

Adv 混淆部分猜测结果 P^{g*} 与 P^g 相一致的概率为：

$$\Pr_{\text{Adv}}[P^{g*} = P^g] = 1 / \prod_{i=1}^{\lceil (m+1)/2 \rceil - 1} \Pr_{\text{Adv}}[P^g[i] = c] \leq 1/2^{\lceil (m+1)/2 \rceil - 1}$$

而对于 P^h 部分，不失一般性，令满足 $\text{Shift}[c - \Omega[1]] = 1$ 的字符 c 出现在 P^h 中各位置 j 的概率均等，那么根据该部分的设计，Adv 猜中 P^h 中位置 j 存储的字符 c 的概率为：

$$\Pr_{\text{Adv}}[P^h[j] = c] = 1/|\text{Shift}[c - \Omega[1]] = 1| \leq 1/T$$

其中， $j \in [\lceil (m+1)/2 \rceil, m]$ ，因为满足 $\text{Shift}[c - \Omega[1]] = 1$ 的字符 c 出现在 P^h 中各位置 j 的概率均等，所以 Adv 的猜测结果 P^{h*} 与 P^h 相一致的概率为：

$$\Pr_{\text{Adv}}[P^{h*} = P^h] = 1 / \prod_{j=\lceil (m+1)/2 \rceil}^m \Pr_{\text{Adv}}[P^h[j] = c] \leq 1/T^{m - \lceil (m+1)/2 \rceil + 1}$$

由于 $P^g \cap P^h = \emptyset$ ， P^g 和 P^h 相互独立，Adv 猜测的概率也相互独立，且对于正整数 m 有 $m = \lceil (m+1)/2 \rceil + \lceil (m+1)/2 \rceil - 1 - ((m+1) \bmod 2)$ ，综上所述，Adv 猜测用户真实搜索内容 P^* 与 P 相一致的概率为：

$$\begin{aligned} \Pr_{\text{Adv}} [P^* = P] &= \Pr_{\text{Adv}} [P^{g^*} = P^g] \cdot \Pr_{\text{Adv}} [P^{h^*} = P^h] \leq \\ &= \frac{1}{2^{\lceil (m+1)/2 \rceil - 1}} \cdot \frac{1}{T^{m - \lceil (m+1)/2 \rceil + 1}} = \\ &= \frac{1}{2^{\lceil (m+1)/2 \rceil - 1}} \cdot \frac{1}{T^{\lceil (m+1)/2 \rceil - (m+1) \bmod 2}} = \\ &= \frac{2T^{((m+1) \bmod 2)}}{(2T)^{\lceil (m+1)/2 \rceil}} \end{aligned}$$

综上所述，半诚实服务器获取的信包含于泄露函数 L_2 输出的信息，且不能唯一确定任意位置的字符信息，因此 JPMDPF 算法在泄露函数 L_2 的半诚实模型下保护了用户的搜索内容。

(2) JPMDPF 算法正确性分析

相较 PMDPF 算法的搜索过程，JPMDPF 算法在搜索过程中跳过了服务器中部分长度与搜索内容长度相同的字符串，而这些字符串均与用户搜索内容不匹配，不会影响搜索过程的正确性。根据移位表构建过程， Ω 中的字符 c 可以分成以下4类。

- 1) $c \in P^h$ ，且跳转距离满足 $\text{Shift}[c] = 1$ 。
- 2) $c \in P^g \wedge c \notin P^h$ ，且跳转距离满足 $\text{Shift}[c] = m - \text{pos} + 1$ ，其中， $P[\text{pos}] = c$ 。
- 3) $c \notin P$ ，且未被用于执行尽可能混淆，跳转距离满足 $\text{Shift}[c] = m + 1$ 。
- 4) $c \notin P$ ，但在算法3的步骤8) ~ 11) 中被用于执行尽可能混淆策略，跳转距离满足 $m/2 < \text{Shift}[c] < m + 1$ 。

对于类型1，其跳转距离为1，搜索没有跳过任何的子字符串，因此也不会错过与用户搜索内容一致的字符串；对于类型2，按照跳转距离跳转后其所处的位置恰好是后续搜索过程中可能匹配成功的第一个字符串，这也是其设置跳转距离的原理，所以跳过的所有字符串均无法匹配成功；对于类型3，因为 $c \notin P$ ，所以当前位置后包含 c 的 m 个连续字符串都不会匹配成功，因此全部跳过不会影响搜索过程的正确性；对于类型4，与类型3类似，当前位置后包含 c 的 m 个连续字符串都不会匹配成功，而跳过的字符串包含于上述的字符串中，所以跳转同样不会影响搜索过程的正确性。综上所述，字符集中的所有类型字符的跳转都不会影响搜索过程的正确性，所以 JPMDPF 算法不会跳过 PMDPF 算法中有符合搜索条件的字符串，可以保证搜索过程的正确性。

JPMDPF 算法可以提高搜索结果的准确率。令 S^* 是与模式串 P 具有相同的长度和指纹值，但内容不同的字符串，且 P^* 与 P 均由不重复的字符组

成，令 P^* 在服务器中的存储位置为 loc ，根据移位表的设计，字符的跳转距离至多为 $m+1$ ，因此在搜索过程中一次跳转到达 P^* 所在位置 loc 的可能位置状态至多有 $m+1$ 种，则跳转前可能的位置为 $\{\text{loc} - 1, \text{loc} - 2, \dots, \text{loc} - (m + 1)\}$ ，根据 JPMDPF 的跳转过程，可以将这些位置状态分为以下4类。

- 1) $\{\text{loc} - 1\}$ ，一次跳转到达位置 loc 就需要满足条件： $S^*[m] \in S^h$ ，该事件发生的概率为： $(m+1)/|\Omega|$ 。
- 2) $\{\text{loc} - 2, \dots, \text{loc} - (m - \lceil (m+1)/2 \rceil + 1)\}$ ，根据式(8)，跳转距离的取值范围不包含 $[2, m - \lceil (m+1)/2 \rceil + 1]$ ，因此2类位置一次跳转到达位置 loc 的概率为0。
- 3) $\{\text{loc} - (m - \lceil (m+1)/2 \rceil + 2), \dots, \text{loc} - m\}$ ，一次跳转到达位置 loc 要求当前位置所对应的字符与 P 中相应位置字符相同，或者该字符不存在于 P 中，且恰好被用于混淆该位置，该事件发生概率为： $1/|\Omega| + 1/[(|\Omega| - m) \cdot (\lceil (m+1)/2 \rceil - 1)]$ 。
- 4) $\{\text{loc} - (m + 1)\}$ ，则要求该位置对应的字符不存在于 P ，且未被用于混淆，该事件发生的概率为： $[1 - 1/(\lceil (m+1)/2 \rceil - 1)] / (|\Omega| - m)$ 。

设跳转前位于上述可能位置的概率均等，为 $1/(m+1)$ ，令 $\lceil (m+1)/2 \rceil = M$ ，则跳转到 P^* 概率为： $\frac{1}{m+1} \cdot (\frac{m-M+1}{|\Omega|} + \frac{M-1}{|\Omega|} + \frac{M-1}{(|\Omega|-m) \cdot (M-1)} + \frac{1}{(|\Omega|-m) \cdot (1 - \frac{1}{M-1})}) = \frac{1}{m+1} (\frac{m}{|\Omega|} + \frac{2M-3}{(|\Omega|-m) \cdot (M-1)})$ ，令 $(2M-3)/(M-1) \approx 2$ ，则跳转到 P^* 的概率为： $\approx (m|\Omega| - m^2 + 2|\Omega|) / [|\Omega| \cdot (|\Omega| - m) \cdot (m+1)]$ 。

在搜索过程中服务器共计算了 $n-m+1$ 个长度 m 的字符串的指纹值，那么假阳性结果出现次数为：

$$\mathcal{FP} \leq \frac{m(m|\Omega| - m^2 + 2|\Omega|)}{p(|\Omega| \cdot (|\Omega| - m) \cdot (m+1))}$$

则 JPMDPF 算法中服务器搜索结果的准确率为：

$$1 - m(m|\Omega| - m^2 + 2|\Omega|) / (p|\Omega|(|\Omega| - m)(m+1))$$

因此，JPMDPF 算法在 PMDPF 算法的基础上降低了假阳性结果在搜索结果中的占比，进一步提高了搜索结果的准确率。

(3) JPMDPF 算法效率分析

虽然两个搜索算法的最坏时间复杂度均为

$O(n)$, 但因为在 JPMDPF 算法的搜索过程中加入了跳转操作, 所以跳转后的字符串的指纹值可以采用式(1)和式(2)两种方法计算。当跳转距离为 1 时, JPMDPF 算法利用式(2)计算, 其他情况下则利用式(1)计算, 这提高了搜索算法的执行效率。这是由于当跳转距离为 1 时, JPMDPF 算法与 PMDPF 算法计算指纹值方法一致, 二者执行效率相同。当跳转距离为 k ($k > 1$) 时, 由于指纹函数的参数 r 和 p 始终保持不变, $r^i \bmod p$ ($i = 1, 2, \dots, m$) 的结果也始终不变, 这些值可以预先计算并存储, 且 JPMDPF 与 PMDPF 算法均需要计算这些值, 所以对于上述的计算, 二者的计算开销相等, 那么 JPMDPF 算法还需要 $m - 1$ 次加法操作和 m 次乘法操作, 而 PMDPF 算法还需要 $2k$ 次加法操作和 $3k$ 次乘法操作。令乘法操作和加法操作的计算开销分别为 t_{mul} 和 t_{add} , 如果 JPMDPF 算法的乘法操作计算开销小于 PMDPF 算法, 则有 $mt_{mul} < 3kt_{mul}$, 解得 $k > m/3$ 。类似地, 对于加法操作, 则有 $mt_{add} < 2kt_{add}$, 解得 $k > m/2$, 而由式(8)可知, k 始终大于 $m/2$, 因此 JPMDPF 算法的计算开销小于 PMDPF 算法, 且通过实验发现, 计算机执行乘法操作所消耗的计算开销往往大于加法操作, 所以 JPMDPF 算法的计算开销也小于 PMDPF 算法, 因此 JPMDPF 算法比 PMDPF 算法更高效。

当用户不设置 T 值时, JPMDPF 算法执行效率约是 PMDPF 算法的 m 倍。根据式(8), 字符 c 的跳转距离可以分为 3 种: $\{1, m-i+1(1 \leq i < \lceil (m+1)/2 \rceil), m+1\}$ 。而搜索时字符集中所有字符的平均跳转距离取决于以上 3 种字符的比例, 假设字符集 Ω 足够大且 P 中所有字符都不相同, 则共有 $\lceil (m+1)/2 \rceil$ 个字符跳转距离为 1, 跳转距离为 $m-i+1$ 的字符共 $2\lceil (m+1)/2 \rceil$ 个, 跳转距离为 $m+1$ 的字符共 $|\Omega| - 3\lceil (m+1)/2 \rceil$ 个。所有字符的跳转距离之和为 $\lceil (m+1)/2 \rceil \cdot 1 + 2\lceil (m+1)/2 \rceil \cdot 3m/2 + (|\Omega| - 3\lceil (m+1)/2 \rceil) \cdot (m+1) = |\Omega|m + |\Omega| - m - 1$, 这里记第二种字符平均跳转距离是 $3m/2$ 个字符, 那么字符集中所有字符的平均跳转距离为 $m+1 - (m+1)/|\Omega|$ 。相对地, PMDPF 算法字符的跳转距离为 1, 故 JPMDPF 算法的执行效率约为 PMDPF 算法的 m 倍。

4 实验与分析

4.1 实验方法

影响 PMDPF 算法和 JPMDPF 算法执行效果的因素有用户搜索内容在服务器中占比、用户自定义字符隐藏系数 T 、服务器存储数据规模和用户搜索内容的长度。由于本文的方案没有基于服务器中存储的内容构建索引, 因而主要与不需要构建任何额外索引的字符串模式匹配算法 KR 进行比较。评价模式匹配算法的指标为服务器执行用户搜索所需时间。本文设计了一系列的实验用于评估 PMDPF 算法和 JPMDPF 算法的性能。

本文以小写英文字母为字符集 Ω , 以牛津词典^[24]中的英文单词构成的字符串为服务器中存储的数据集, 以数据集中字符串的总长度为数据集的长度, 以词典中的英文单词和随机字符构成的字符串为用户的查询集, 基于 PMDPF 算法和 JPMDPF 算法, 采用 C++ 编程语言进行实现, 并在 Windows 10 系统上进行实验。其中, CPU 为 Intel(R) Core(TM) i5-10500, 内存为 8 GB, r 值为 26, 服务器的个数为 2, 除预处理时间和空间开销测试实验外 p 值均被设置为 100 591。

1) 用户搜索内容在服务器数据集中的占比对 PMDPF 算法和 KR 算法的影响实验。

在 KR 算法中, 当出现被搜索的字符串与搜索内容具有相同指纹值时, 需要按位进行字符比对, 因此搜索内容在服务器数据集中的占比影响了 KR 算法的搜索效率, 而对 PMDPF 算法的搜索效率几乎没有影响。为尽量减少搜索内容长度对本文实验结果的影响, 分别以搜索长度为 4、6 和 8 的查询集进行实验, 改变搜索内容在服务器数据集中的占比, 对比 KR 算法和 PMDPF 算法的搜索效率。

2) 用户自定义字符隐藏系数 T 对 JPMDPF 算法的影响实验。

T 是该算法中为了满足用户对搜索内容的不同保密要求而设置的, 该值可以调整字符的跳转距离, 改变搜索过程, 影响 JPMDPF 算法的搜索效率。当 T 不超过搜索内容中隐藏部分的字符种类个数时, 字符隐藏系数的数值对 JPMDPF 算法搜索效率没有影响, 当 T 超过搜索内容中隐藏部分的字符种类个数时, 字符隐藏系数越大,

JPMDPF 算法搜索效率越低, 且当 T 增大到与字符集的大小 $|\Omega|$ 时, JPMDPF 算法的搜索过程与 PMDPF 算法的搜索过程一致。因此本文选取了不同的 T 进行实验, 对比 JPMDPF 算法和 PMDPF 算法执行搜索所需时间。

3) 服务器存储数据规模对 PMDPF 算法、JPMDPF 算法和 KR 算法的影响实验。

服务器中存储的数据规模直接影响着模式匹配算法的查询效率, 所以选取了不同的数据规模来对比不同算法的执行效率。在实验中, 2 个服务器中存储相同的数据集, 用户同时向 2 个服务器发送搜索请求, 服务器存储的数据规模即存储字符串的总长度, 从 100 000 增加到 1 000 000, 用户的搜索内容为查询集中长度为 4 的随机字符串进行实验, 对比 3 个算法执行搜索所需时间。

4) 用户搜索内容的长度对 JPMDPF 算法效率影响的实验。

根据第 3 节的描述, 搜索内容的长度对 PMDPF 算法的执行效率几乎没有影响, 但根据第 4 节的分析, 当数据集规模相同时, 搜索内容的长度越长, JPMDPF 算法执行所需时间相较 PMDPF 算法所需时间越少, 而在英文单词中, 大部分的英文单词的长度小于等于 12, 因此选取搜索集中长度为 2 到 12 的字符串进行实验, 对比两个算法的执行效率。

5) 素数 p 的大小对搜索内容预处理时间和空间开销影响的实验。

在本文提出的方案中, 素数 p 决定了搜索内容中分布式点函数真值表的长度, 直接影响了对搜索内容进行预处理所需花费的时间与空间开销, 因此分别选取了从 100 591 到 1 000 003 的不同质数进行实验, 测试本文提出的算法预处理所需的时间。

4.2 实验结果与分析

当服务器存储的数据集的长度为 100 000 时, 查询集的占比分别为 0%、50% 和 100%。查询集(搜索内容)的长度为 4 时, KR 算法执行搜索所需时间分别为 168.74ms、170.41ms 和 172.51ms, PMDPF 算法执行搜索所需时间分别为 168.42ms、168.48ms 和 168.64ms。查询集的长度为 6 时, KR 算法执行搜索所需时间分别为 169.18ms、171.31ms 和 173.54ms, PMDPF 算法执行搜索所需

时间分别为 168.94ms、169.36ms、169.44ms。查询集的长度为 8 时, KR 算法执行搜索所需时间分别为 168.26ms、169.44ms、171.5ms, PMDPF 算法执行搜索所需时间分别为 167.44ms、167.76ms、167.82ms。图 4 分别展示了查询集的长度为 4、6 和 8 时两种算法完成搜索所需要的时间。实验结果表明 PMDPF 算法搜索效率比 KR 算法搜索效率更高。搜索内容在服务器中占比越高, PMDPF 算法搜索效率比 KR 算法搜索效率提升越多。

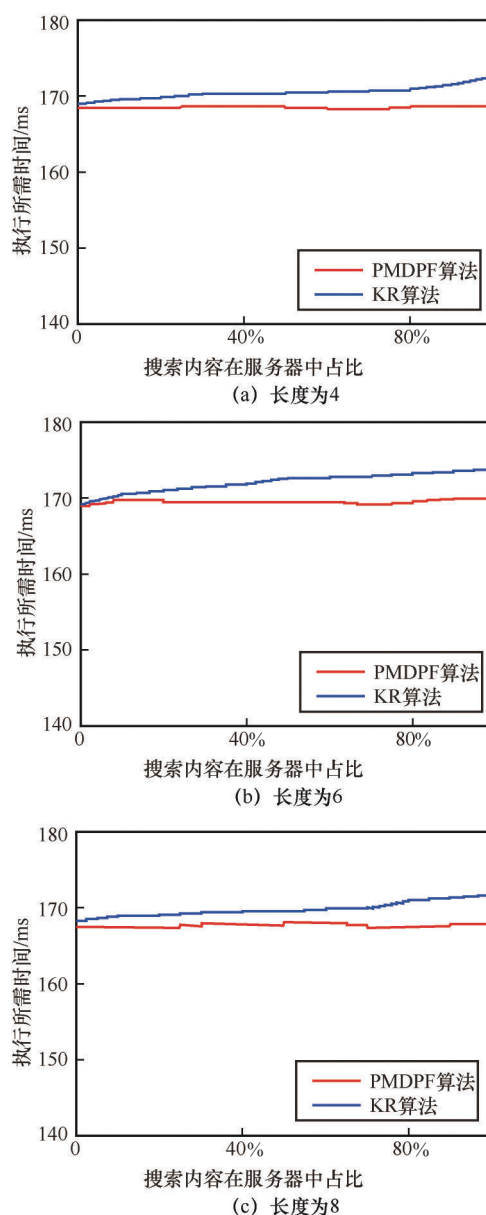


图4 不同长度的搜索内容在服务器中占比不同时执行所需的时间

Figure 4 Execution time for search content of different lengths with variable proportions in a server

当服务器存储的数据集的长度为 100 000 时, 查询集的占比分别为 1%, 查询集 (搜索内容) 的长度为 4, 字符隐藏系数 T 从 0 增加到 4 时, PMDPF 算法执行搜索所需时间从 168.58ms 增加到 169.52ms, JPMPDPF 算法执行搜索所需时间从 40.88ms 增加到 41.22ms。字符隐藏系数 T 从 5 增加到 26 时, PMDPF 算法执行搜索所需时间从 169.24ms 增加到 170.12ms, JPMPDPF 算法执行搜索所需时间从 45.94ms 增加到 174.68ms。图 5 展示了不同字符隐藏系数条件下 PMDPF 和 JPMPDPF 算法执行搜索所需时间。实验结果表明当用户对搜索内容的保密需求不高时, JPMPDPF 算法是高效的。用户通过可自定义的字符隐藏系数 T 来满足对搜索内容的不同保密需求。

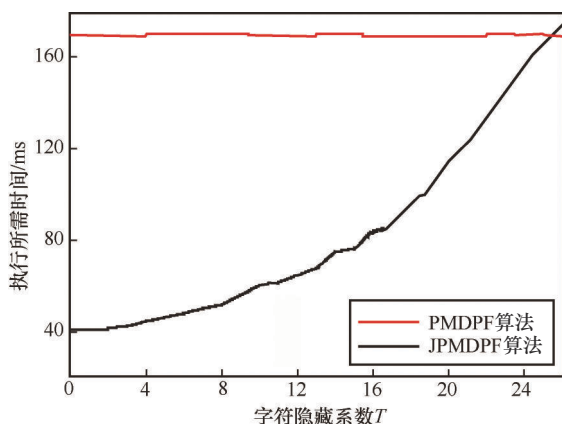


图 5 不同字符隐藏系数执行所需时间

Figure 5 Execution time required for different character hiding coefficients

在数据集的长度相同时, JPMPDPF 算法执行效率最高。当搜索内容的长度为 4, 搜索内容在服务器中占比为 1%, 用户不额外设置字符隐藏系数, 数据集的长度从 100 000 增加到 1 000 000 时, KR 算法执行所需时间从 170.04ms 增加到 1703.14ms, PMDPF 执行所需时间从 168.58ms 增加到 1700.74ms, JPMPDPF 执行所需时间从 41.88ms 增加到 417.56ms, 图 6 展示了查询集长度为 4 时 PMDPF 算法和 JPMPDPF 算法在不同数据集长度下执行搜索所需时间。实验结果表明 JPMPDPF 算法基于字符的跳转是高效的。

当服务器中数据集的长度为 100 000 时, 搜索内容在服务器中占比为 1%, 用户不额外设置字符隐藏系数。搜索内容的长度从 2 增加到 12 时, PMDPF 执行所需时间从 168.53ms 增加到

168.72ms, JPMPDPF 执行所需时间从 52.68ms 减少到 31.16ms。图 7 展示了数据集长度为 100 000 时, PMDPF 算法和 JPMPDPF 算法在不同搜索内容长度下执行搜索所需时间。实验结果表明, 当数据集的长度固定时, 搜索内容长度越长, JPMPDPF 算法效率提升越多。

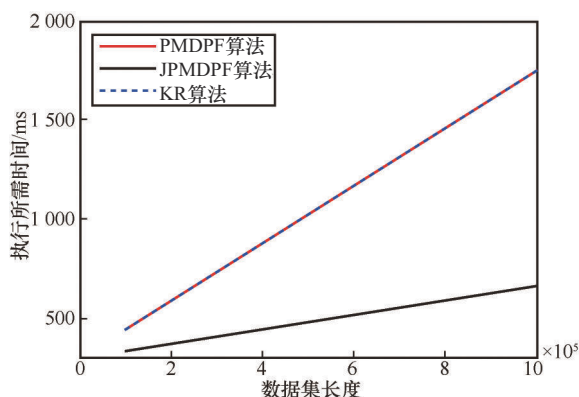


图 6 不同数据集长度执行所需时间

Figure 6 Execution time required for different dataset lengths

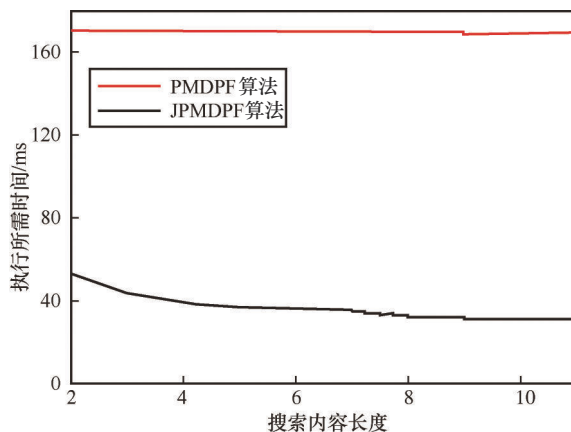


图 7 不同搜索内容长度执行所需时间

Figure 7 Execution time required for different search content lengths

当搜索内容的长度为 4 时, 搜索内容在服务器中占比为 1%, 用户不额外设置字符隐藏系数, 数据集的长度为 100 000 个字符。p 值从 100591 增加到 1000003 时, 预处理所需时间从 3.1ms 增加到 31.9ms, 真值表所占空间从 804.728 KB 增加到 8.0000 24 MB, 而文献[17]中 100 000 个字符的加密预处理需要 5s, 因此本文提出的算法的预处理过程更高效。图 8 展示了不同 p 值条件下预处理过程所需的时间和空间开销。

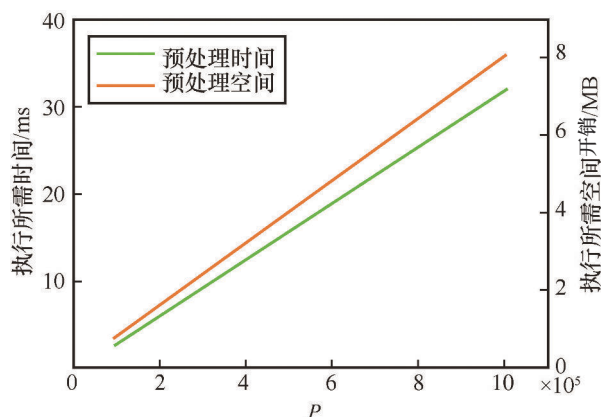


图8 不同质数时预处理执行所需时间和空间开销

Figure 8 Preprocessing time and space required for different primes

5 结束语

本文提出了两个基于分布式点函数的模式匹配算法，它们能够保护用户搜索内容并支持对任意内容的搜索。本文首先提出了PMDPF算法，利用指纹函数以及分布式点函数构造搜索内容真值表，并存储在两台独立服务器中运行，最终由用户合成搜索结果。该算法中服务器不需要提前构建索引，且比经典字符串模式匹配算法KR算法的效率提升约5%。随后，提出了JPMDPF算法，以泄露更多信息为代价，其搜索效率更高，当搜索内容的长度为 m 时，JPMDPF算法的执行效率约是PMDPF算法的 m 倍，且出现假阳性结果概率更低。最后，实验结果表明本文提出的两个算法比明文字符串模式匹配算法KR算法更高效，具有较高的理论意义和应用价值。

参考文献：

- [1] AL-SSULAMI A M, AZMI A M, MATHKOUR H, et al. LsHASHq: a string matching algorithm exploiting longer q-gram shifting[J]. Information Processing & Management, 2022, 59(5): 103057.
- [2] LECROQ T. Fast exact string matching algorithms[J]. Information Processing Letters, 2007, 102(6): 229-235.
- [3] SUNDAY D M. A very fast substring search algorithm[J]. Communications of the ACM, 1990, 33(8): 132-142.
- [4] FARO S, LECROQ T. Efficient variants of the backward-oracle-matching algorithm[J]. International Journal of Foundations of Computer Science, 2009, 20(6): 967-984.
- [5] PELTOLA H, TARHIO J. Alternative algorithms for bit-parallel string matching[C]//International Symposium on String Processing and Information Retrieval. Berlin, Heidelberg: Springer, 2003: 80-93.
- [6] CHEN Y B, HE J S, WEI W, et al. A multi-model approach for user portrait[J]. Future Internet, 2021, 13(6): 147.

- [7] MEHROTRA S, SHARMA S, ULLMAN J, et al. Partitioned data security on outsourced sensitive and non-sensitive data[C]//Proceedings of the 2019 IEEE 35th International Conference on Data Engineering (ICDE). Piscataway: IEEE Press, 2019: 650-661.
- [8] 徐琳, 魏晓超, 蔡国鹏, 等. 一个高效的安全两方近似模式匹配协议[J]. 计算机研究与发展, 2022, 59(8): 1819-1830.
- [9] XU L, WEI X C, CAI G P, et al. An efficient secure two-party approximate pattern matching protocol[J]. Journal of Computer Research and Development, 2022, 59(8): 1819-1830.
- [10] LI R, LIU A X. Adaptively secure and fast processing of conjunctive queries over encrypted data[J]. IEEE Transactions on Knowledge and Data Engineering, 2022, 34(4): 1588-1602.
- [11] CHOR B, KUSHILEVITZ E, GOLDREICH O, et al. Private information retrieval[J]. Journal of the ACM, 1998, 45(6): 965-981.
- [12] DVIR Z, GOPI S. 2-server PIR with subpolynomial communication[J]. Journal of the ACM, 2016, 63(4): 1-15.
- [13] CORRIGAN-GIBBS H, KOGAN D. Private information retrieval with sublinear online time[C]//Annual International Conference on the Theory and Applications of Cryptographic Techniques. Cham: Springer, 2020: 44-75.
- [14] YU M K, YANG K C, WEI L B, et al. Practical private information retrieval supporting keyword search in the cloud[C]//Proceedings of the 2014 Sixth International Conference on Wireless Communications and Signal Processing (WCSP). Piscataway: IEEE Press, 2014: 1-6.
- [15] WONG W K, CHEUNG D W L, KAO B, et al. Secure kNN computation on encrypted databases[C]//Proceedings of the 2009 ACM SIGMOD International Conference on Management of data. New York: ACM, 2009: 139-152.
- [16] SUN J F, BAO Y Y, NIE X Y, et al. Attribute-hiding predicate encryption with equality test in cloud computing[J]. IEEE Access, 2018, 6: 31621-31629.
- [17] FARO S, LECROQ T. The exact online string matching problem: a review of the most recent results[J]. ACM Computing Surveys, 2013, 45(2): 13.
- [18] DESMOULINS N, FOUQUE P A, ONETE C, et al. Pattern matching on encrypted streams[C]//International Conference on the Theory and Application of Cryptology and Information Security. Cham: Springer, 2018: 121-148.
- [19] BOUSCATIÉ É, CASTAGNOS G, SANDERS O. Public key encryption with flexible pattern matching[C]//International Conference on the Theory and Application of Cryptology and Information Security. Cham: Springer, 2021: 342-370.
- [20] GILBOA N, ISHAI Y. Distributed point functions and their applications[C]//Annual International Conference on the Theory and Applications of Cryptographic Techniques. Berlin, Heidelberg: Springer, 2014: 640-658.
- [21] BOYLE E, COUTEAU G, GILBOA N, et al. Efficient pseudorandom correlation generators: silent OT extension and more[C]//Annual International Cryptology Conference. Cham: Springer, 2019: 489-518.
- [22] BRESLAUER D, GALIL Z. Real-time streaming string-matching[J]. ACM Transactions on Algorithms, 2014, 10(4): 1-12.
- [23] KARP R M, RABIN M O. Efficient randomized pattern-matching algorithms[J]. IBM Journal of Research and Development, 1987, 31(2): 249-260.

- [23] GOLDBREICH O, MICALI S, WIGDERSON A. How to play ANY mental game[C]//Proceedings of the Nineteenth Annual ACM Conference on Theory of computing-STOC '87. New York: ACM, 1987: 219-219.
- [24] ENGLISH O. Oxford english dictionary[J]. Encyclopedia of Swearing, 1976: 334.

[作者简介]



王健旭 (1997-), 男, 黑龙江绥化人, 东莞理工学院硕士生, 主要研究方向为计算机网络与信息安全。



李睿 (1975-), 男, 湖南汨罗人, 博士, 东莞理工学院教授, 主要研究方向为数据隐私保护。



李银 (1982-), 男, 河南信阳人, 博士, 东莞理工学院副教授, 主要研究方向为安全数据库、计算机代数。