信州大学審査学位論文

暗号化に対応したXML文書検索手法に関する研究

2014年9月

中村伸一

目次

第 1 章

序論

1.1 はじめに

インターネット回線の高速化と安定性の向上に伴い，インターネットを介してサーバやストレージ等を提供する，クラウドと呼ばれるサービスがAmazon[1]等から始まっている．

クラウドは，利用開始までの時間が短い，サーバやストレージ等の性能の増強が容易である，運用に必要な人件費が少ない等から多くの組織で利用されている．

クラウドで利用されるサーバは，専門の施設で管理者によって複数のサーバを集約して運用されるが, 利用者と専門の施設が地理的に離れている，クラウドのセキュリティ方針により利用者が専門の施設に入ることができない等から，利用者がサーバと管理者を管理するのは難しい．

クラウドに関する情報セキュリティ上の問題として，管理者がサーバに保存されているデータを持ち出す問題がある．解決策として，ファイルの暗号化が挙げられるが，データを検索する場合に暗号化されたデータがサーバ上で復号されるため，管理者の権限を利用してサーバを操作することで，管理者が復号されたデータを持ち出すことが可能である．

そのため，クラウドにあるデータのセキュリティを確保しつつ利用できるように，データを暗号化したままで検索する手法が求められるようになってきている．また，従来の暗号化されたままで検索する手法ではクラウドにあるデータの対応が難しくなってきている．

本論文ではデータを暗号化したままで検索する手法を提案する．提案する手法は適用範囲をクラウドが提供するデータベースまで拡張しようとする試みの１つである．特に，インターネット上における情報交換に利用さ

れているデータ形式の 1 つである Extensible Markup Language(以下 XML)に対する，暗号化に対応した検索手法について研究している．提案手法はクラウドのような情報セキュリティの管理が難しい情報環境でも，安全な XML 文書の検索環境を構築する場合に有効である．

1.2 研究の意義

主要な商用データベースには，データベースのセキュリティを管理する機能が実装されている．しかし，クラウドによるデータベースの提供(Database as a Service)に伴って，利用者からのデータベースに対するセキュリティの要望も変わってきているが，主要な商用データベースはその要望に対応できていない．

データを暗号化したままで検索する手法は，データベースに関するファイルを暗号化しておき検索する場合にメモリ上にデータを復号して検索する手法，ファイルを暗号化するだけでなくメモリ上も暗号化したままで検索する手法に分けられる．

前者の検索手法は実用化され，主要な商用データベースに実装されている．後者の検索手法は実用化されているとは言えず，現在も高速化や問い合わせ方式の拡張やデータの暗号化範囲の拡張等に向けた研究が進められている．本論文で扱うのは後者の検索手法である．

主要な商用データベースのデータを暗号化したままで検索する手法は，データを暗号化してファイルに登録しておき，検索する場合は暗号化された条件とファイルをメモリ上で復号して検索するように実装されている．しかし，この実装はデータベースのファイルの盗難に主眼が置かれており，メモリ上のデータの読み取りについて考慮されていない．従来はデータベースを組織内で構築していたため，利用者がサーバと管理者の管理がしやすく，前者の検索手法でもセキュリティを確保することができていた．しかし，データベースをクラウドで利用する場合，利用者がサーバと管理者を管理することが難しいため，メモリ上のデータの読み取りのような高度な手法の対応が必要となってきている．

これまでに提案されてきた，データベースを暗号化したままで検索する手法について，データベースに暗号化したデータを保存しておき，検索する場合は条件を暗号化したデータと同じ方法で暗号化し，暗号化したデータと一致するか判定することで検索する手法がある．金子[2]らは，データベースを暗号化したままで高速に検索できるように, Semi-ShuffledBF をさらに高速化する手法を提案している．なお，データベースを暗号化したままで検索する手法で XML 文書を登録すれば, XML 文書も暗号化したままで検索できると考えられるが，条件に合う要素及びテキストが取得できるだけで, 利用者は XPath や XQuery 等の問い合わせ方式に対応した処理を行う必要があり, 問い合わせ方式で検索する仕組みは考慮されていない.

データベースに暗号化された XML 文書を登録して検索する手法について，Brinkman[3]らは，Song[4]らが提案した暗号化された文字列を検索する手法から XPath を利用した XML 文書の検索へ拡張した手法を提案している．Xu[5]らは，キーワードを利用した XML 文書の検索手法を提案しているが, Xu[5]らの手法はデータベースに保存する要素名及びテキストを暗号化することで暗号化に対応させることができる(以下 暗号化 Xu[5]).

Brinkman[3]と暗号化 Xu[5]の手法は，要素及びテキストの位置を検索に利用するため，XML 文書中の要素の入れ子によって構成される構造は暗号化されていない．そのため，検索される XML 文書と同じ内容の XML 文書を管理者が入手した場合，検索している要素及びテキストの位置と入手した XML 文書を突き合わせることで，検索内容と結果が推測される問題がある.

この問題に対応するため，XML 文書の構造を暗号化したままで検索する手法の提案が重要である．暗号化したままで要素及びテキストの位置関係を判定し，一定の位置関係を持つ要素及びテキストを抽出する手法を考案する必要がある.

1.3 本論文の内容と構成

本論文では，1.2 節であげたデータを暗号化したままで検索する手法のさらなる実用化に向けた課題である「検索される XML 文書と同じ内容の XML 文書を管理者が入手した場合，検索している要素及びテキストの位置と入手した XML 文書を突き合わせることで，検索内容と結果が推測される問題」を取り上げる．この課題の解決方法として，XML 文書の要素だけでなく要素の入れ子によって構成される構造も暗号化したままで検索する手法の提案を行う．

データを検索するための問い合わせ方法について，XML 文書には問い合わせ言語である XQuery と複数の単語で構成されるキーワードによる問い合わせ方法がある．

XQuery は XML データベースに対するデータ照会機能を提供するため World Wide Web Consortium(以下 W3C)で開発された問い合わせ方法であり，XML データベースで照会したデータを自由に加工できる特徴を持っている．そのため，XQuery はソフトウェアが利用するための XML 文書を XML データベースから効率的に作成することができる. 本論文では，XQuery における検索機能の基盤技術である XPath について, XPath で指定した部分木を暗号化したままで検索する手法を提案する．

キーワードは XQuery や XPath の文法や検索される XML 文書の詳細等を知る必要がなく利用者に扱いやすい問い合わせ方法であるが，検索結果の XML 文書が大きすぎたり小さすぎたりした場合，利用者が検索結果の正しさを確認する必要があるため，利用者にとって適切な大きさとなる検索結果に関する研究が進められている．検索結果に関する代表的な手法として，すべてのキーワードと同じ要素名の要素及びテキストを含む木の最も深い分岐点である Lowest Common Ancestor(以下 LCA), キーワードと同じ要素名の要素及びテキストから LCA までのパス上にキーワードと同じ要素名が存在しない LCA である Valuable Lowest Common Ancestor, 特定の属性を持つ要素名及びテキストをキーワードとして LCA を検索した Meaningful LCA，LCA の子孫にある LCA 以下の要素を除いた要素に

対する LCA である Exclusive Lowest Common Ancestor，子孫に LCA が存在しない LCA である Smallest Lowest Common Ancestor(以下 SLCA)がある．本論文では，これらの手法で広く用いられている SLCA を検索する手法を提案する．

2 章では，XPath を用いて指定した部分木を検索する手法を提案する．XML 文書に対する XPath にマッチする部分を検索する場合, 事前に XML 文書からハッシュと暗号化されたデータで構成されたデータをリレーショナルデータベース(以下 RDB)に登録する．検索する場合は RDB から XPath の一部の要素と位置関係が同じ部分木のデータを暗号化したままで取得し，取得したデータを復号して XPath とマッチするか判定する．想定される攻撃に対する安全性の評価と Brinkman[3]の手法との性能比較も行っている．

3 章では，キーワードを用いて SLCA を検索する手法を提案する．XML 文書に対するキーワードを検索する場合，事前に XML 文書から Bloom Filter と暗号化されたデータで構成されたデータを RDB に登録する．検索する場合はキーワードを利用して RDB の Bloom Filter を検索し，検索結果の Bloom Filter に対応する暗号化されたデータを取得し，取得したデータを復号して検索結果を作成する．想定される攻撃に対する安全性の評価と Xu[5]と暗号化 Xu[5]の手法との性能比較も行っている．

4 章では，結論として 2 章と 3 章での結果を要約し，今後の課題について述べる．

1.4 関連研究

1.4.1 XQuery や XPath を利用した検索手法

Brinkman[3]らは，XML 文書中のすべての要素から，XPath の条件にマッチする要素を残していくことで，XML 文書を検索する手法を 2004 年に提案している. 要素名のマッチングは Song[4]らが 2000 年に提案した暗号化された文字列を検索する手法を利用している．Yang[6]らは，利用者側

に XML 文書の構造を圧縮したインデックス，RDB に暗号化された XML 文書のデータを登録し，検索内容を利用者側のインデックスで処理した後で，必要なデータを RDB から取得する XQEnc という手法を 2006 年に提案している．Schrefl[7]らは，利用者側に文書の構造とサーバに要素のパスと要素名を暗号化したインデックスを作成し，検索する場合は利用者とサーバとの間で暗号化されたパスと要素名をやり取りして，結果を利用者側でチェックする手法を 2005 年に提案している．Lee[8]らは，XML 文書の要素名と要素の位置と復号に必要な鍵を暗号化したインデックスを作成し，利用者が持つ鍵でインデックスを復号して検索することで，XML 文書の選択的な暗号化に対応する検索手法を 2006 年に提案している．Jammalamadaka[9]らは，XML 文書に関係と構造と値を暗号化する要素を追加することで，XML 文書の選択的な暗号化に対応する検索手法を 2006 年に提案している．Wang[10]らは，XML 文書の要素の位置を 0 から 1 の実数で表現する手法(Discontinuous Structural Interval(以下 DSI))を利用して，暗号化された要素名と DSI のインデックス，DSI と対応する暗号化データのインデックスを RDB に登録し，検索する要素の DSI が暗号化データの場合は,利用者側で暗号化データを復号して検索することで，XML 文書の選択的な暗号化に対応する検索手法を 2006 年に提案している．Chang[15]らは，暗号化された XML 文書全体を入手して復号し検索するのは効率的ではないと指摘し，XML 文書のスキーマを利用して検索結果に必要な暗号化された部分のみ入手するように XQuery を翻訳することで，暗号化された部分を減らす手法を 2011 年に提案している．Unay[17]らは，暗号化された XML 文書検索に関する手法について，暗号化標準や暗号化インデックスやパスや DeweyOrder の暗号化等の視点でまとめたものを 2008 年に報告している．

1.4.2 キーワードを利用した検索手法

金子[2]らは， Non-ShuffledBF と ShuffledBF を組み合わせることで安全性を損なわずに RDB を高速に検索できる Semi-ShuffledBF と呼ばれる

手法を提案したが，検索する場合に選択されるタプル数に合わせて Bloom Filter の処理を変えることでさらに高速化した手法を 2012 年に提案している．Xu[5]らは，XML 文書からキーワードを含む要素を検索し，特定の位置関係にある要素から LCA を取得することで SLCA を検索する手法を 2005 年に提案している．Watanabe[11]らは，データと鍵でハッシュ化するだけでは同じデータは同じ結果となり出現数からデータが推測される可能性があると指摘し，暗号化されたデータをさらにハッシュ化することでデータの推測を防ぐ手法を 2009 年に提案している．Yan[12]らは，疑似乱数ビットを文書に対するインデックスに処理することで，時間計算量と通信量が少ないモバイル機器でも暗号化したままで検索できる手法を 2005 年に提案している．Liu[13]らは，検索可能公開鍵暗号である PEKS に対して，クラウド上の文書に対するモバイル機器のキーワード検索を想定した，時間計算量と通信量が少ない検索可能公開鍵暗号である CKPS と呼ばれる手法を 2012 年に提案している．

Bertino[14]らは， XML 文書中の部分木を異なる鍵で暗号化することで利用者に対するアクセス制御を行う手法を 2002 年に提案している．Li[16]らは，事前に XML 文書からアクセス制御の情報を追加したインデックスを作成することで検索結果にアクセス制御が反映される手法を 2010 年に提案している．

1.5 他の検索手法との比較

提案手法と他の XML 文書の検索手法(論文[3]と[5]～[10]と[14]～[16])の比較を表 1 に示す．表 1 の項目について，問い合わせ方法は検索する場合の問い合わせ方法，全体暗号化可能は XML 文書全体を暗号化して利用できる，利用者側データ不要は利用者側にインデックス等の XML 文書に関するデータがないままで利用できる，一部データ取得は暗号化データから検索に必要な部分だけ取得する，構造不要は検索する場合に XML 文書の構造を利用しないことを示す．表 1 内の記号について，○は XML 文書全体で対応，△は XML 文書の一部で対応，×は対応しないを示す．

提案手法は関連研究と比べ，XML 文書の構造を検索する場合に利用せず，暗号化データから必要な部分だけ検索に利用している点が他の手法にはない特徴である．

関連研究について，論文[3]と[15]は構造不要が×のように，検索する場合に要素の位置を利用するため，XML 文書の構造が必要である．論文[5]は一部データ取得が△のように，XML 文書中の同じ要素がまとめて保存されているため，検索に必要ない要素を取得する．論文[6]～[7]は利用者側データ不要が×のように，利用者側にインデックス等の XML 文書に関するデータが必要である．論文[8]～[10]と[14]と[16]は全体暗号化可能と一部データ取得が×のように，暗号化されたデータにある位置情報のみで検索を行うため，XML 文書全体を暗号化すると XML 文書全体の暗号化されたデータを利用者が取得する必要がある．

表 1 手法の比較

論文	問い合わせ方法	全体暗号化可能	利用者側データ不要	一部データ取得	構造不要
Brinkman[3]	XPath	○	○	○	×
暗号化Xu[5]	ｷｰﾜｰﾄﾞ	○	○	△	○
Yang[6]	XPath	○	×	○	○
Schrefl[7]	XPath	○	×	○	○
Lee[8]	XPath	×	○	×	△
Jammalamadaka[9]	XPath	×	○	×	△
Wang[10]	XPath	×	○	×	△
Bertino[14]	XPath	×	○	×	△
Chang[15]	XQuery	○	○	○	×
Li[16]	ｷｰﾜｰﾄﾞ	×	○	×	△
提案手法(2章)	XPath	○	○	○	○
提案手法(3章)	ｷｰﾜｰﾄﾞ	○	○	○	○

1.6　準備

XML[18]はインターネット上で構造化されたデータの交換を容易にするため，W3C により開発された拡張可能なマーク付け言語である．人が扱いやすいように，テキストで作成されており日本語等の英語以外の文字も利用できる．XML は要素(図 1①,②)と属性(図 1③)とテキスト(図 1④)によって構成されており，要素は要素を入れ子とすることができる(図 1②)．要素は<要素名>の開始タグと</要素名>の終了タグで挟むことで表現する．図 2 に XML 文書の例を示す．

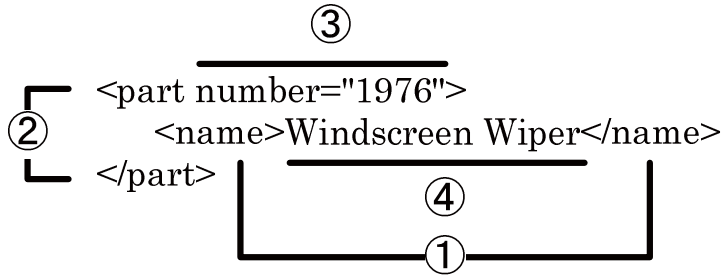


図 1　XML

```
<a>
 <a>
  <b></b>
  <c></c>
 </a>
 <b></b>
 <c></c>
</a>
```

図 2　XML 文書

XML は円滑にデータ交換できるように，XML 文書の妥当性を検証する仕組みを持っている．妥当性の検証は Document Type Definition(以下 DTD)(図 3(a))等の XML 文書を定義するスキーマ言語で作成された文書と XML 文書で行う．XML の文法規則のみに準拠している XML 文書は整形

式 XML 文書，XML の文法規則だけでなく DTD 等の XML 文書を定義するスキーマ言語で作成された文書にも準拠している XML 文書は妥当な XML 文書と呼ばれる．整形式 XML 文書はスキーマ言語で作成された文書により定義されていないが，柔軟に利用できることから多く利用されている．図 3 (a)の DTD に準拠している XML 文書を図 3 (b)に示す．

```
<!ELEMENT firstName (#PCDATA)>
<!ELEMENT secondName (#PCDATA)>
<!ELEMENT info ANY>
<!ELEMENT data (firstName,secondName,info?)>
<!ELEMENT myDocument (data)*>
<!ATTLIST data age CDATA #IMPLIED>
```

(a)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE myDocument SYSTEM "example.dtd">
<myDocument>
  <data age="29">
    <firstName>Fred</firstName>
    <secondName>Bloggs</secondName>
  </data>
  <data>
    <firstName>Tony</firstName>
    <secondName>Blair</secondName>
    <info>PM of UK</info>
  </data>
</myDocument>
```

(b)

図 3　DTD

XML 文書は要素をノードとする木構造で表すことができる．このとき，木のノードは要素名及びテキストでラベル付けされている．木のノードについて，深さは枝の長さを 1 とするときの根からノードまでの枝数，パスは根からノードまで辿る経路に出現するノードである．XML 文書の図 2 を同等の木で表現したものを図 4 に示す．

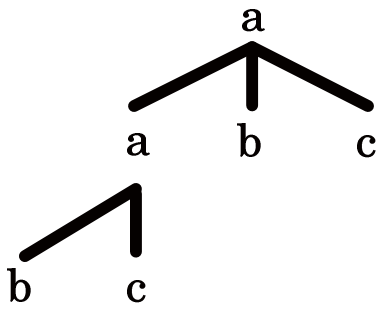


図 4　木構造

XML 文書を検索するには，要素及びテキストの位置関係(先祖,子孫,親,子,兄弟,系統等)を把握する必要があるが，提案手法はリージョン[19]とDeweyOrder を採用している．

リージョンは，木のノードに対して開始位置(以下 start)と終了位置(以下 end)の数字のペアで定義される. 木のノードを深さ優先探索順に番号を付け，これを位置番号とする．start はノードの位置番号，end はそのノードを根とする部分木のノードで一番大きな位置番号とする．

図 2 と同等の木のノードにリージョンと深さとパスをラベリング(start,end,深さ,パス)した木を図 5 に示す. ラベリングとはノード名と別の情報を木のノードに追加することである．パスのラベリングは，括弧内に左から右へ順に根まで辿る経路に出現するノードの start を”,”で区切って表現する．例えば，ノード a(図 5○)のラベリング(2,4,1,(2,1))は左から，ノードの位置番号の 2，ノード a(図 5○)を根とする部分木(図 5 灰色)のノードで一番大きな位置番号の 4，ノード a(図 5○)の深さの 1，パス上のノードの位置番号(2,1)となっている．

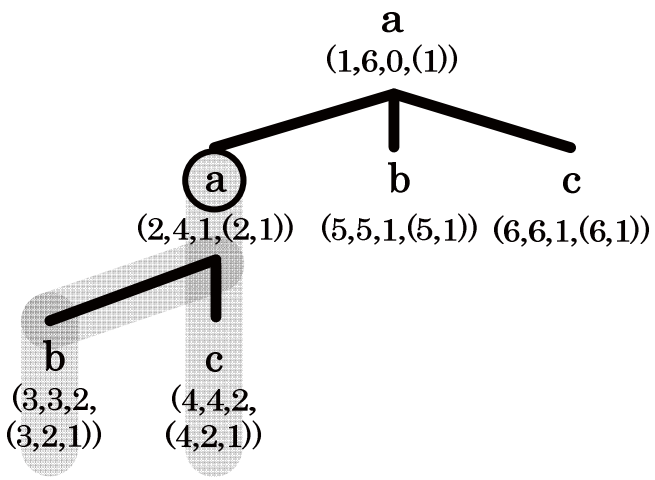


図 5　リージョン

DeweyOrder はノードの深さごとに”.”で区切り，兄弟の要素及びテキストの左端を 0 とし，右に向かって順に 1,2,...と番号を割り当てたものである. 図 2 と同等の木のノードに DeweyOrder を追加したものを図 6 に示す.

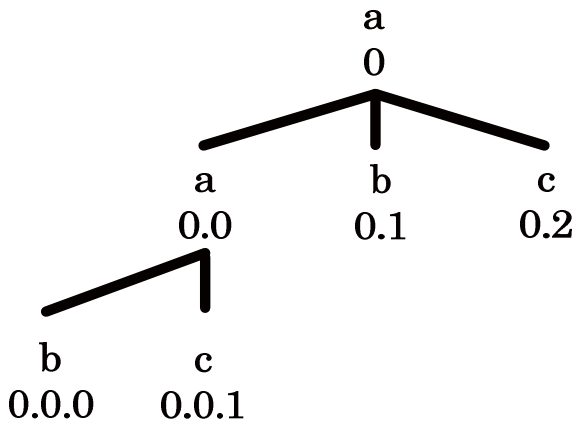


図 6　DeweyOrder

XPath[20]は，XML 文書の特定部分を指定するために W3C が開発した言語である．一般的に 1 つ以上のロケーションパスの並びとして記述される. 図 7 にロケーションパスを省略構文で記述した XPath の例を示す．ロケーションパスは軸(図 7①)とノードテスト(図 7②④)と述語(図 7③)で構成される. この例では, XML 文書中のすべての要素（図 7①）に対して title 属性の値が”第一章”(図 7③)である要素 chapter(図 7②)の子要素である要素 paragraph(図 7④)を指定している.

```
//chapter[@title=”第一章”]/paragraph
①   ②          ③              ④
```

図 7　XPath

XQuery[21]は，XML データベースに対してデータの参照を行うために W3C が開発した言語である．XML データベースから参照したデータの加工を行うため，処理の繰り返し(For)，変数との結合(Let)，判断(Where)，結果のソート(Order by)，結果の作成(Return)で構成される FLWOR 表現式で表現される．図 8 の(a)に XML 文書，(b)に XQuery，(c)に(a)を(b)で処理した結果を示す. この例では, (a)の要素 n の位置を取得し(図 8(b)①)，要素 p の数を取得し(図 8(b)②)，要素 n の属性 gender が”male”の要素 n を選択し(図 8(b)③)，要素 id の順でソートし(図 8(b)④)，要素 n の子孫の要素から結果を作成している(図 8(b)⑤).

```
<directory>
  <cust id="10001">
    <n gender="male">山下太郎</n>
    <p type="office">03-53xx-xxxx</p>
  </cust>
  <cust id="10002">
    <n gender="female">赤井道子</n>
    <p type="mobile">080-xxxx-xxxx</p>
  </cust>
  <cust id="10003">
    <n gender="male">大木伸城</n>
    <p type="office">03-xxxx-xxxx</p>
    <p type="mobile">070-xxxx-xxxx</p>
  </cust>
</directory>
```

(a)

```
for $i in fn:doc("customer.xml")//n ―①
let $p := fn:count($i/../p) ――――②
where $i/@gender = "male" ――――③
order by $i/../@id descending ――――④
return ――――――――――――――⑤
  <cust>
    { element id {fn:string($i/../@id)} }
    <n>{ $i/text() }</n>
    <p>{ $p }</p>
  </cust>
```

(b)

```
<cust>
  <id>10003</id>
  <n>大木伸城</n>
  <p>2</p>
</cust>
<cust>
  <id>10001</id>
  <n>山下太郎</n>
  <p>1</p>
</cust>
```

(c)

図 8　XQuery

XPath と XQuery を利用した検索手法は, 検索結果が XPath と XQuery が指定する XML 文書中の特定の部分となるため，検索手法に関わらず検索結果は同じとなる．一方で，キーワードを利用した検索手法は，検索結果がキーワードに関係する要素及びテキストを含む部分木となるが，検索手法によって検索結果を重視する点が異なるため，同じキーワードで検索しても検索手法によって検索結果が異なる．キーワードによる検索結果について，すべてのキーワード(b,c)(図 9○)を含む最小部分木の根は Lowest Common Ancestor (図 9□と灰色□)と呼ばれ，子孫に LCA が存在しない LCA は Smallest Lowest Common Ancestor (図 9 灰色□)と呼ばれる．

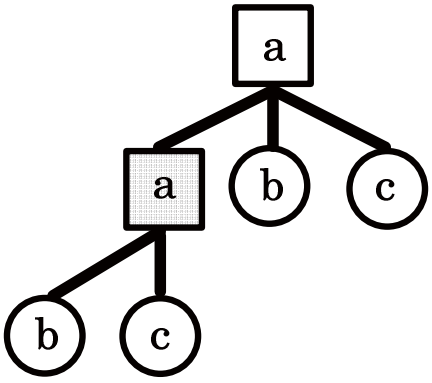


図 9　LCA と SLCA

暗号化したままでXML文書中の要素及びテキストの位置関係を判定し，一定の位置関係を持つ要素を抽出するには，暗号化したままで特定の文字列を含むデータを取得する必要があるが，提案手法はハッシュと Bloom Filter[22]を採用している．

ハッシュはデータから代表する数値(以下 ハッシュ値)を取得する仕組みである(図 10)．ハッシュはデータの検索やデータの改ざん検出等に利用され，ハッシュ値を取得する関数はハッシュ関数と呼ばれる．

ハッシュをデータの検索に利用する場合，ハッシュ値とデータで構成されたハッシュテーブルを作成し，ハッシュテーブルからハッシュ値に対応したデータを取得する．データの検索に利用する場合のハッシュ関数は，異なるデータに対して同じハッシュ値を取得すると，同じハッシュ値のデ

ータを再検索する必要があるため，異なるデータのハッシュ値はできるだけ重ならない必要がある．データを数値に変換してハッシュテーブルの大きさに対する余りをハッシュ関数とすることが多い．

ハッシュをデータの改ざん検出に利用する場合，データから取得したハッシュ値をデータと共に相手に送り，受け取った相手がデータからハッシュ値を取得して送られたハッシュ値と同じか確認することで，データが改ざんされていないか確認することができる．送られたハッシュ値も改ざんされる可能性があるためハッシュ値は暗号化される場合が多い．この場合のハッシュ関数の代表的なものに MD5 や SHA1 等がある．異なるデータのハッシュ値はできるだけ重ならないことに加えて，ハッシュ値から元のデータが推測されない，あるデータのハッシュ値と同じ別のデータを見つけることが難しい，ハッシュ値が同じ 2 つのデータを見つけることが難しい必要がある．本論文で扱うのは後者のハッシュ関数である．

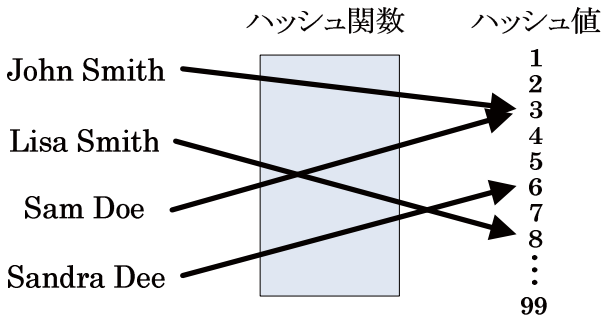


図 10　ハッシュ

Bloom Filter[22]は要素が集合に含まれるか判定するデータ構造である．ハッシュ等の他のデータ構造と異なり要素のデータ自体を持たないため，登録する要素に対して必要とするメモリ量は少ない．要素の追加はビット配列中の要素に対応するビットを 1 とすることで行うが，要素間で対応するビットと重なる場合がある．そのため，要素の削除と登録した要素の一覧を得ることはできない. ハッシュ関数はビット配列の長さの範囲(0,...,m-1)の整数を出力する．要素を集合に登録するには，要素名をハッシュ関数で数値に変換し，数値に対応するビット配列のビットを 1 にする．要素名

をハッシュ関数で数値に変換し，数値に対応するビット配列のビットが 1 であれば要素は集合に含まれる(図 11).

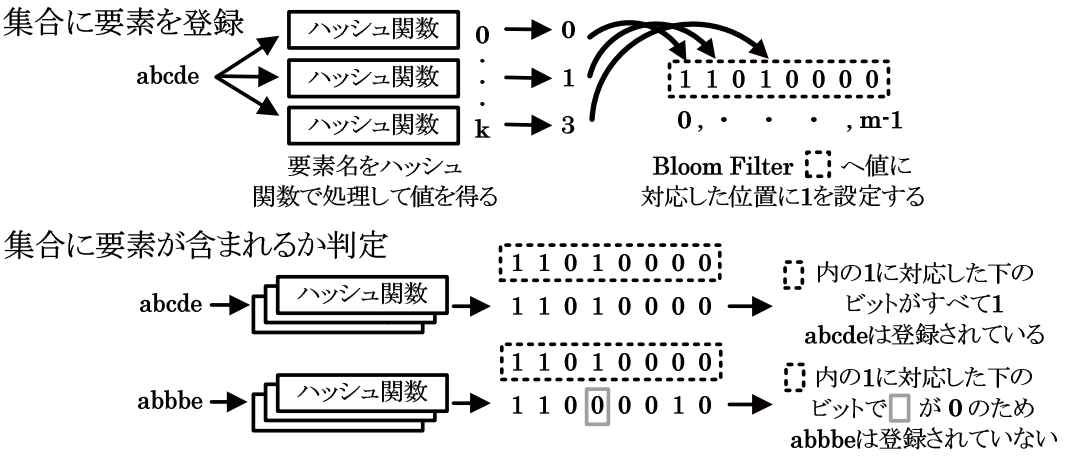


図 11　Bloom Filter

要素を登録するためビットを 1 に設定すると，登録していない要素に対応するビットも 1 となってしまい，集合に属していると判断を誤る偽陽性が発生する場合がある. 偽陽性が発生する確率について, 長さ m のビット配列，k 個のハッシュ関数，ビット配列に登録する要素数 n とした場合，要素を追加してもビットが 1 の確率を式(1)に示す.

$$1-\left(1-\frac{1}{m}\right)^{kn} \tag{1}$$

ハッシュ関数に対応するビットのすべてが 1 となっており，集合に属していると判断を誤る偽陽性が発生する確率を式(2)に示す.

$$\left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^{k} \tag{2}$$

暗号は第三者からデータを知られないようにするため，特別な知識なしではデータが読めないようにする仕組みである．主な暗号として暗号化と復号の鍵が同じ共通鍵暗号，暗号化と復号の鍵が異なる公開鍵暗号がある．公開鍵暗号において暗号化の鍵は公開鍵，復号の鍵は秘密鍵と呼ばれる．共通鍵暗号は公開鍵暗号と比べて処理速度は速いが，鍵の交換は第三者から知られないように行う必要がある．公開鍵暗号は以下の(1)～(3)の手順で鍵の交換を安全に行うことができるが，共通鍵暗号より処理が遅い．そのため，公開鍵暗号を利用して共通鍵暗号の鍵を安全に交換し，交換した鍵でデータを暗号化する手法がよく用いられる．公開鍵の入手について，信頼できる第三者機関が電子メール等の本人の情報と公開鍵で構成された表を公開し，表から公開鍵を入手することで本人の公開鍵を安全に入手することができる．(図 12).

(1) 公開鍵を相手に送る．
(2) 相手は送られた公開鍵で暗号化した情報を相手に返す．
(3) 返された情報を秘密鍵で復号する．

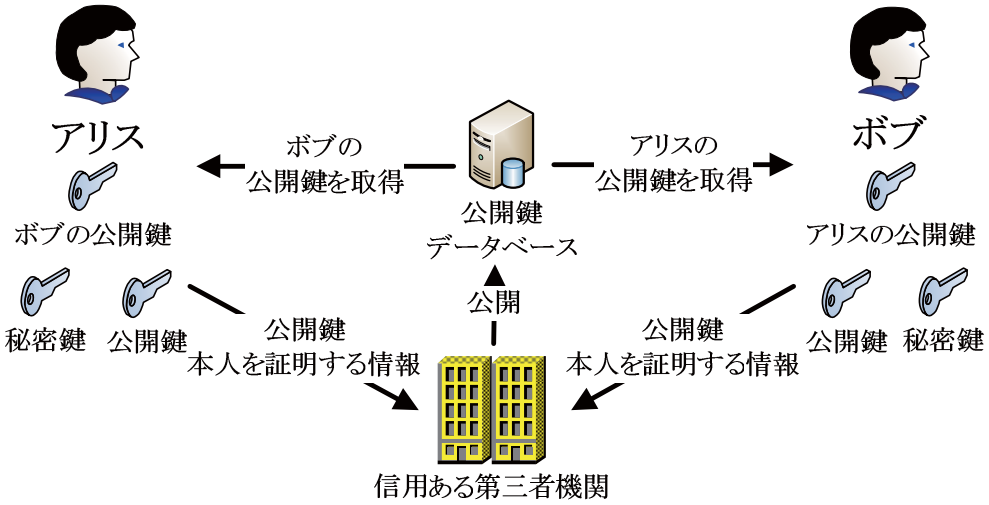


図 12　第三者機関の利用による公開鍵の交換

1.7　従来の手法による XML 文書検索の問題点

Brinkman[3]と暗号化 Xu[5]の手法は，要素及びテキストの位置を検索に利用するため，XML 文書中の要素の入れ子によって構成される構造は暗号化されていない. Brinkman[3]と暗号化 Xu[5]の手法を利用して, XML 文書で販売や公開されているデータベースの検索環境を RDB で構築した場合，RDB に登録された XML 文書と同じ内容の XML 文書を管理者が入手できれば，RDB に登録された XML 文書と入手した XML 文書は同じ位置に同じ要素及びテキストがあるため，以下の手法で検索内容と結果が推測される問題がある(図 13).

(1) 管理者が RDB に登録されている XML 文書を入手する.
(2) 利用者が検索している要素及びテキストの位置を管理者が入手する.
(3) 管理者が(1)で入手した XML 文書の(2)で入手した位置にある要素及びテキストを参照することで要素名及びテキストを推測する.

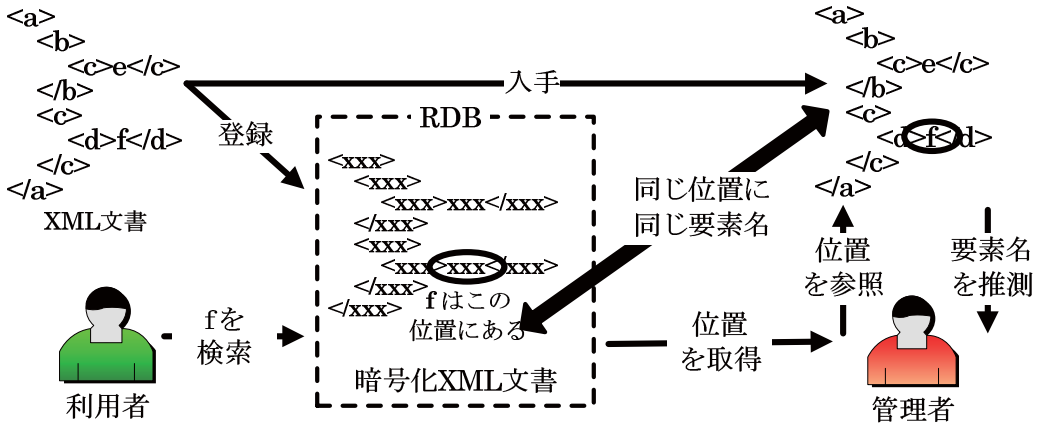


図 13　管理者による情報漏えい

Brinkman[3]と暗号化 Xu[5]の手法を利用して，XML 文書で販売や公開されている化合物データベースの検索環境を RDB で構築した場合，検索した化合物が管理者に推測されるだけでなく，検索した化合物から開発している化学製品の原料や組成等が管理者に推測される可能性がある(図 14).

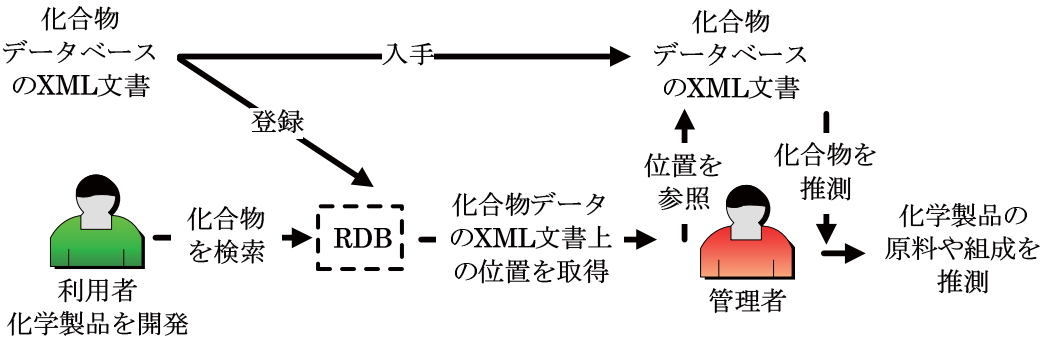


図 14　検索サービスの情報漏えい

第 2 章

XPath を用いた暗号化 XML 文書検索手法

2.1　暗号化 XML 文書検索システム

1.3 節で述べた問題点に対応した，XML 文書を暗号化したままで検索する暗号化 XML 文書検索システム(以下 システム)を以下に提案する．

システムの概要を図 15 に示す．前処理として XML 文書を RDB サーバに登録するため，利用者は XML 文書と暗号化に使用する鍵(以下 K)をクライアントに入力する．クライアントは K を用いて XML 文書の暗号化を行い，暗号化されたデータを RDB サーバに登録する．検索する場合は利用者が XPath と K をクライアントに入力する．クライアントは 2.2 節の手法に基づき，検索結果を返す．なお，システムは XML 文書の更新に対応しない．

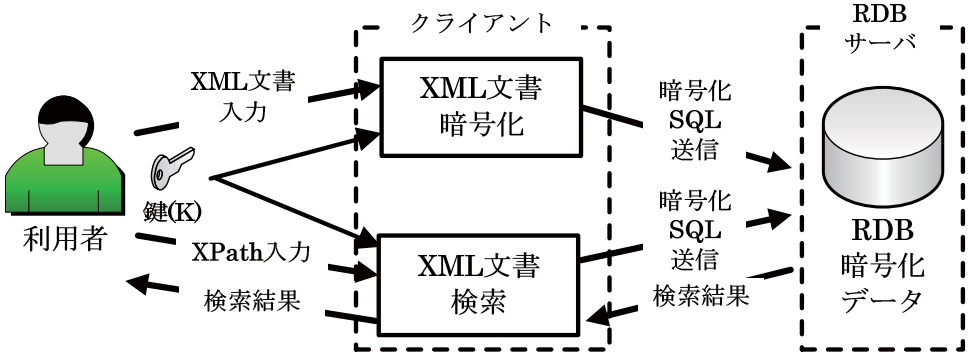


図 15　暗号化 XML 文書検索システムの構成

2.2　提案手法

2.2.1　概要

提案手法は，XPath の分岐点と葉の要素名を取得し(図 16(a))，分岐点と葉の要素名と同じ要素及びテキストを XML 文書から検索する(図 16(b))．次に，分岐点の要素名と同じ要素を根とした部分木に，すべての葉の要素名と同じ要素及びテキストがある部分木のデータを XML 文書から検索する(図 16(c))．最後に，部分木のデータと XPath がマッチするか XPath プロセッサを利用して判定する(図 16(d))．

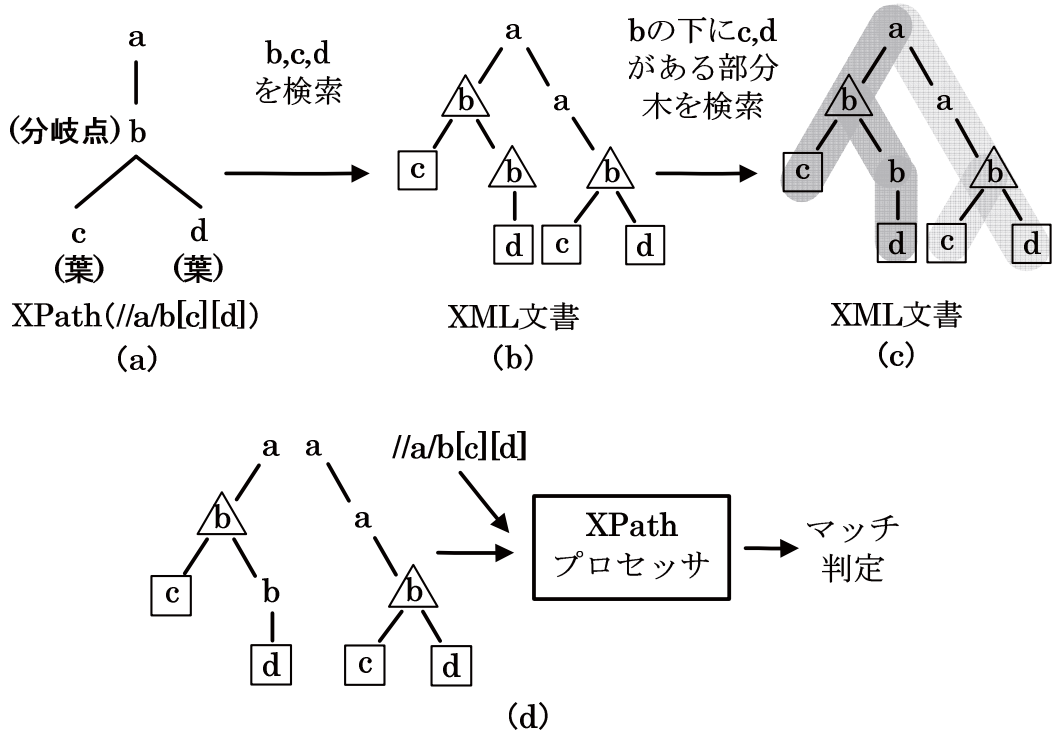


図 16　提案手法

システムは，クライアントと RDB サーバとの間で以下のデータのやり取りを行う．

(1) XML 文書からすべての要素及びテキストの情報等が暗号化されたデータテーブル，同じ要素名及びテキストの情報等が暗号化されたインデックステーブルを作成する(図 17(a)①).

(2) XPath とマッチングする部分木のデータを取得するため，XPath から分岐点と葉の要素名を取得し，インデックステーブルとデータテーブルを利用して，分岐点と葉の要素名と同じ要素及びテキストのデータを取得する(図 17(b)①).

(3) (2)で取得したデータで分岐点の要素名と同じ要素及びテキストについて，これを根とする部分木に，葉の要素名と同じ要素及びテキストがすべてあるか調べ，あればデータテーブルから要素及びテキストのパスのデータを取得する(図 17 (b)②).

(4) (3)で取得したデータを XML 文書に変換し，XPath プロセッサで XPath とマッチするか判定する(図 17 (b)③).

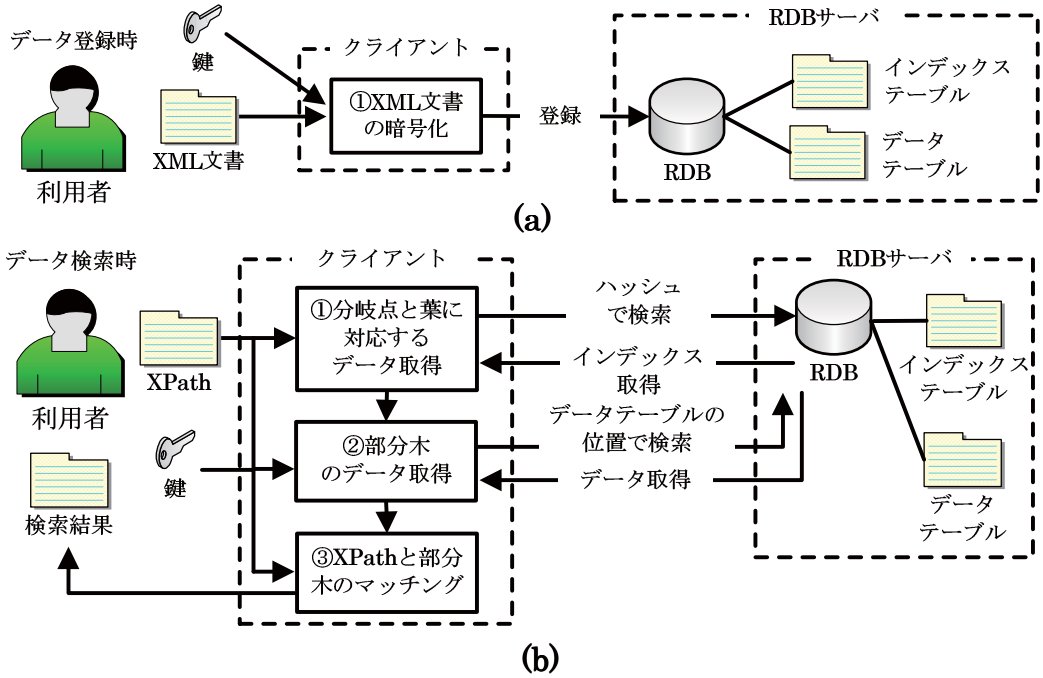


図 17　暗号化 XML 文書検索システムの構成

XPath における分岐点は，2 つ以上の子要素を持つ要素で最も根に近い要素を指す．分岐点から根までのパス上の要素は 1 つの子要素しか持たないため分岐点は XPath に高々1 つであり，分岐点を根とする部分木は必ずすべての葉を含む．そのため，XML 文書から分岐点と葉を利用して取得した部分木は，XPath とマッチしない場合もあるが，XPath とマッチする部分木が漏れることはない．そのため，XPath のすべての要素名と位置関係を比較せずに XPath とマッチする部分木の検索が可能である．

ただし，分岐点と葉のみを利用して部分木を取得するため，提案手法で扱うことのできる XPath の構文は，図 18 で示す Extended Backus–Naur Form で記述した範囲となる．<Tag>は XPath における要素名を指す．

以下に提案手法が扱うことのできない XPath の例を示す．

(1) 図 18 の Tag に'*'が含まれる XPath（例 /site/regions/asia/*）
(2) 図 18 の左端以外に'//'が含まれる XPath（例 /site//[name][phone]）

<XPath> ::= <Location> | '/' <Location> | '//' <Location>
<Location> ::= <Step> ('/' <Step>)*
<Step> ::= <Tag> ('[' <Tag> ']')*

図 18　提案手法が扱う XPath の構文

暗号化の際，平文と暗号文から鍵を推測されないようにするため，平文にランダム文字列を追加している．暗号化は平文のデータを D，ランダム文字列を S，鍵を K，$\mathrm{E_{enc}}$を暗号化関数，D'を暗号化データとして式(3)，復号は$\mathrm{E_{dec}}$を復号関数として式(4)に従って行っている．

$$\mathrm{D'} = \mathrm{E_{enc}}(\mathrm{K}, \mathrm{D}, \mathrm{S}) \tag{3}$$

$$\mathrm{D} = \mathrm{E_{dec}}(\mathrm{K}, \mathrm{D'}) \tag{4}$$

2.2.2 節から 2.2.5 節の提案手法の詳細で時間的計算量を述べるが，時間的計算量で使われる変数について，XML 文書中のすべての要素及びテキストの数をm，XML 文書中の XPath の分岐点の要素名と同じ要素及びテキストの数をn，XPath の葉の要素数をo，XML 文書中の XPath の葉の要素名と同じ要素及びテキストの数をp，マッチングする部分木の数をq，要素及びテキストの深さをrとしている．

2.2.2 XML 文書の暗号化

XML 文書を RDB サーバ に保存するため，XML 文書の情報を調べてデータテーブル(以下 DTBL)とインデックステーブル(以下 ITBL)を構築する．図 19(a)の XML 文書に対する DTBL と ITBL を図 20 に示す．図 19(b)は(a)と同等の木のノードに start，end，深さ，パスでラベリングした木である. 図 20 について, DTBL と ITBL の EncryptedData の括弧内は，平文の EncryptedData である．データが長いため DTBL と ITBL の EncryptedData の暗号列の途中を省略する．

DTBL の項目について，Loc は DTBL と ITBL の構築時にランダムに割り当てられる数値，EncryptedData は 2.2.1 節で述べた手法で暗号化された XML 文書の要素及びテキストの情報(要素名及びテキスト,start,end,深さ,パス)である．DTBL のパスはパス上の要素に対応する DTBL の Loc である．

ITBL の項目について，Hash は K で暗号化してハッシュ化した要素名及びテキスト，EncryptedData は 2.2.1 節で述べた手法で暗号化された同じ Hash である要素及びテキストの情報の集合 $((start_1,end_1,Loc_1),...,(start_i,end_i,Loc_i))$(i=Hash が同じ要素及びテキストの数)である．ITBL の Loc は DTBL の Loc と対応する．この処理における時間計算量について，DTBL と ITBL は XML 文書中のすべての要素及びテキストから構築されるため $O(m)$である．

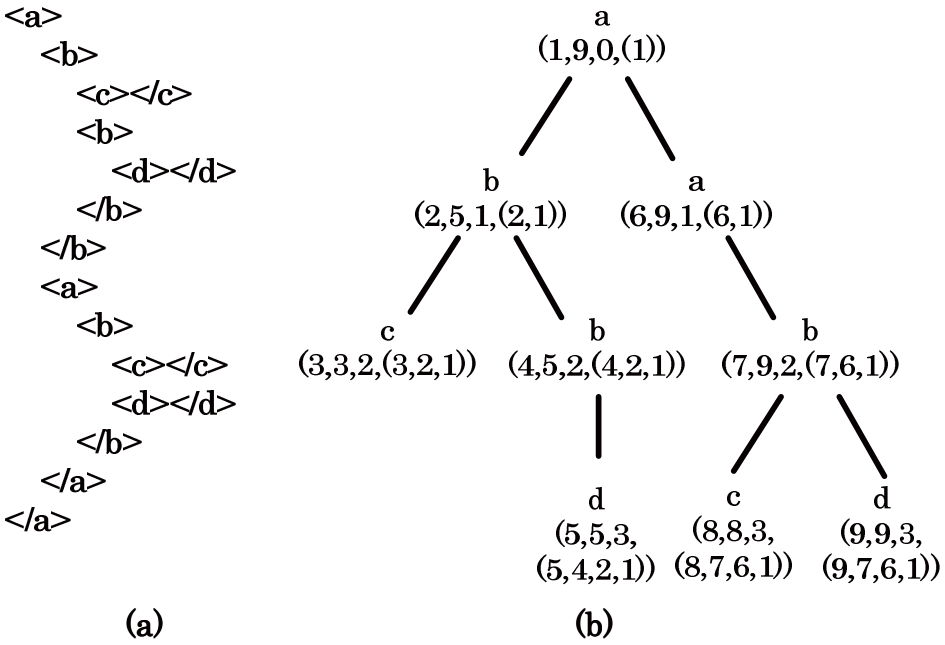


図 19　XML 文書と木構造

Loc	EncryptedData
62	69f8c...f8059 (c,3,3,2,(62,93,6))
86	0faa8...28d74 (d,5,5,3,(86,32,93,6))
32	73cde...e0d48 (b,4,5,2,(32,93,6))
93	cba2d...f1377 (b,2,5,1,(93,6))
84	f355d...fc166 (c,8,8,3,(84,72,88,6))
68	5bad8...b658c (d,9,9,3,(68,72,88,6))
72	8ada5...4a466 (b,7,9,2,(72,88,6))
88	b14d2...3f0a2 (a,6,9,1,(88,6))
6	71d18...b1d8a (a,1,9,0,(6))

DTBL

Hash	EncryptedData
119	adb71...2b1d6 ((5,5,86),(9,9,68))
677	5beea...2d024 ((6,9,88),(1,9,6))
296	4adc7...05810 ((4,5,32),(2,5,93),(7,9,72))
164	648b7...82d46 ((3,3,62),(8,8,84))

ITBL

図 20　図 19(a)を XML 文書とする DTBL と ITBL

2.2.3　分岐点と葉に対応するデータ取得

XPath の分岐点と葉の要素名が同じデータの取得を以下の(1)～(2)の手順で行う．なお，XPath に分岐点がない場合，一番深さの大きい要素と同じ要素名の要素及びテキストを取得する．この処理における時間計算量について，(1)は分岐点と葉の要素数を合計した回数を実行し，(2)は分岐点と葉の要素名が同じ要素及びテキストの数の回数を実行するため $O(n+p)$ である．(1)で実行する部分は分岐点と葉の要素数と非常に少ないため時間計算量から無視している．

(1) XPath から分岐点と葉の要素名を取得し，取得した要素名を K で暗号化してハッシュ化したものと，ITBL の Hash が同じデータを取得する(図 21(a)).

(2) 取得した ITBL のデータの Loc(図 21(b)○)と DTBL の Loc が同じデータを取得する(図 21(b)).

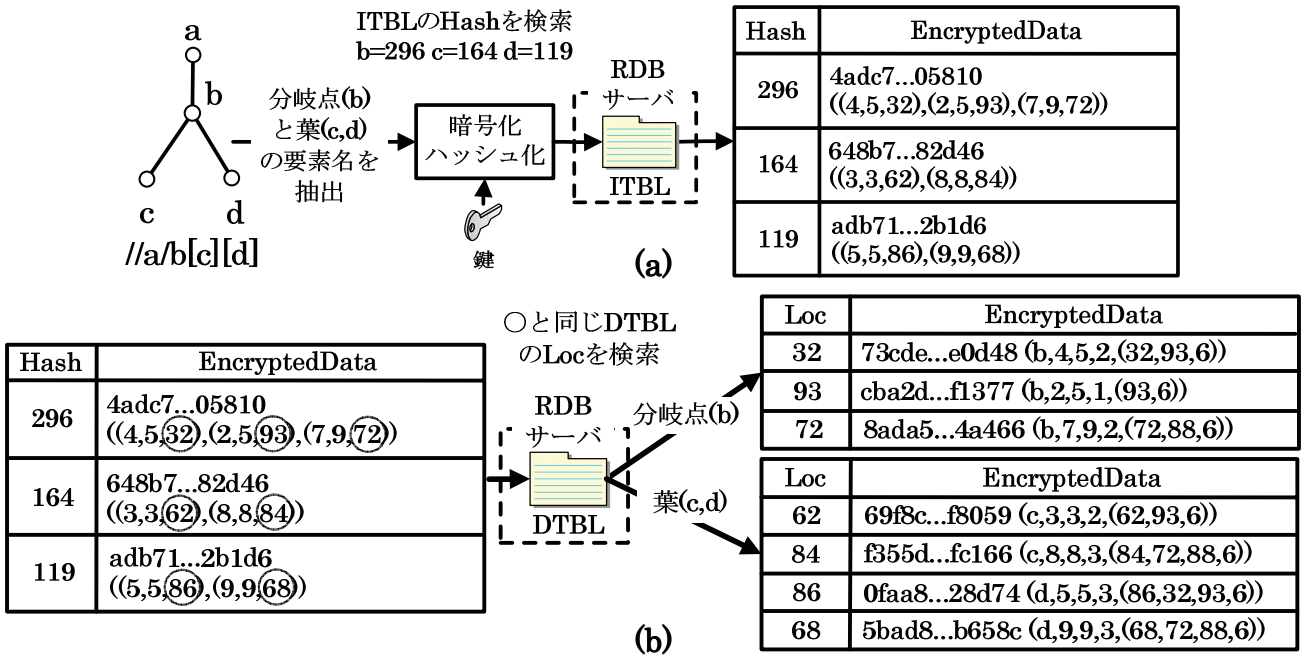


図 21　XPath の要素名の取得

2.2.4　部分木のデータ取得

XPath とマッチングする部分木を取得するため，2.2.3 節で取得した分岐点のデータ毎に以下の(1)～(2)の手順で部分木のデータの取得を行う(図 22)．なお，分岐点がない場合，XPath で一番深さの大きい要素と同じ要素名の要素及びテキストのパスを部分木のデータとして取得する．この処理における時間計算量について，(1)分岐点と要素名が同じ要素毎に葉の要素名が同じ要素数の回数を実行し，(2)は分岐点の数を実行するため $O(n \times p + o \times q \times r)$である．

(1) 分岐点の start と end(図 22○)の間に葉の start(図 22△)があるデータを検索する．

(2) 検索したデータの要素名が XPath のすべての葉の要素名を含む場合，検索したすべてのデータのパス(図 22□)と DTBL の Loc が同じデータを取得する．ただし，DTBL の Loc が重複するデータは除く．

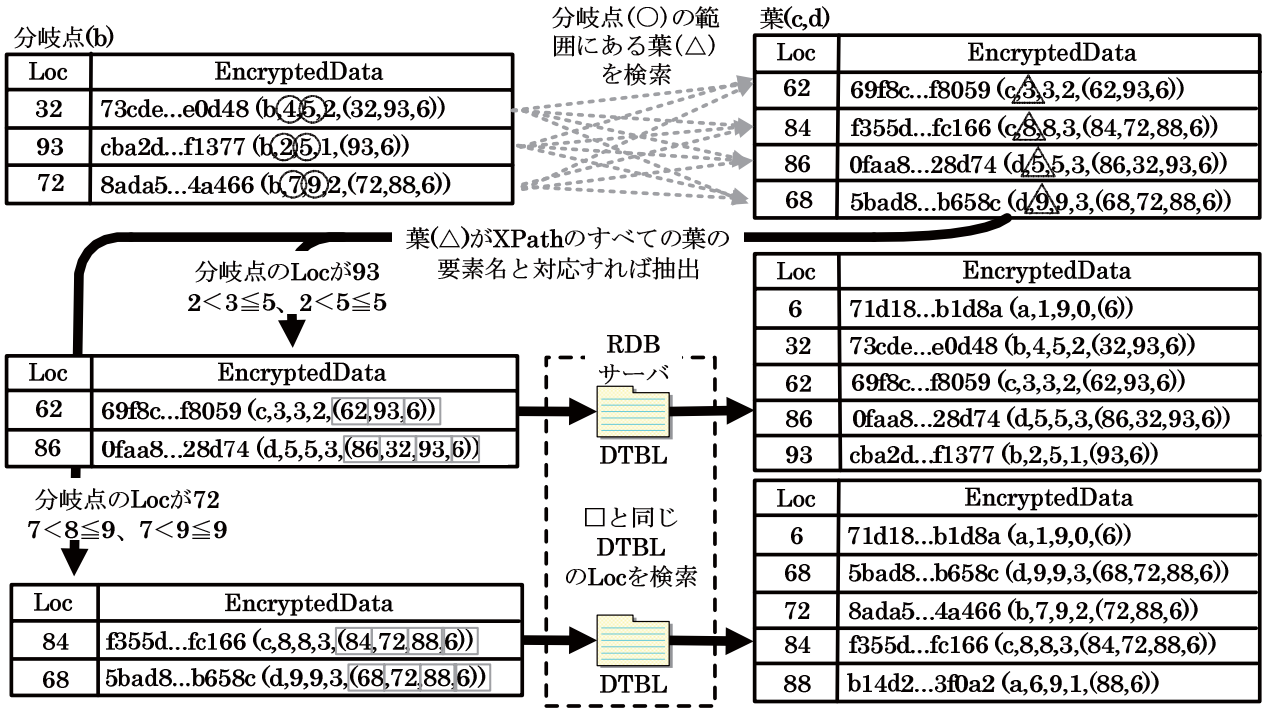


図 22　XPath との位置関係の確認

2.2.5　XPath と部分木のマッチング

2.2.4 節で取得した部分木のデータから，XPath とマッチングするための XML 文書を作成し，XPath プロセッサを利用してマッチングを行う(図 23)．この処理はすべてクライアントで行われる．この処理における時間計算量について，利用する XPath プロセッサに依存する．

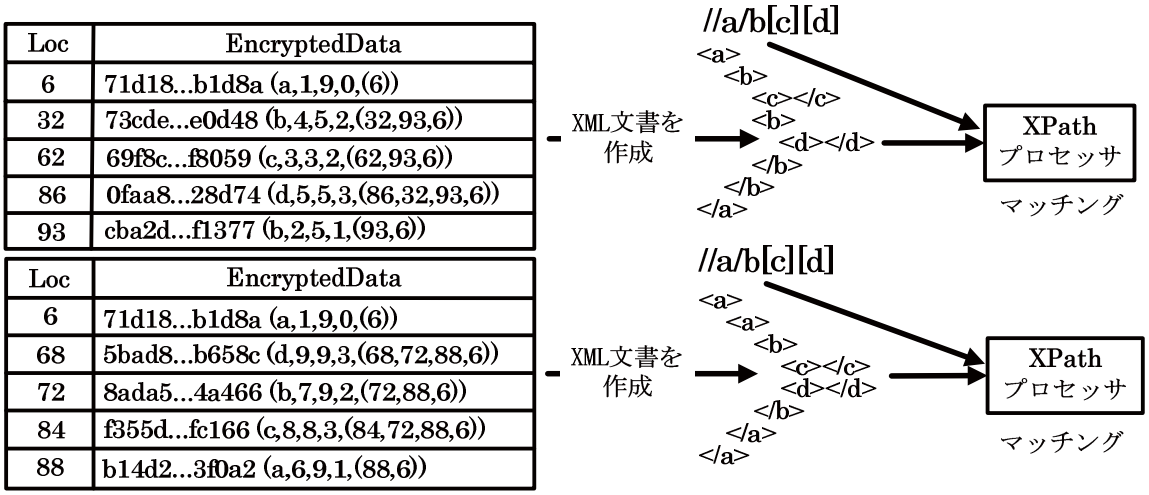


図 23　XPath プロセッサによるマッチング

2.3　安全性について

提案手法の安全性の評価を，想定される攻撃に対する対応策を分析することで行った．

(1) 要素に対する攻撃

① DTBL と ITBL の EncryptedData に対する既知平文攻撃と選択平文攻撃

既知平文攻撃を行うには要素及びテキストに関する情報がある平文, 選択平文攻撃を行うには任意の要素及びテキストに関する情報がある平文が必要だが，2.2.1 節で述べたように平文にランダムな文字列を追加して暗号化されており平文は入手できない. そのた

め，攻撃はできない.

② ITBL の Hash に対する攻撃

2.2.2 節で述べたように K で暗号化してハッシュ化した要素名及びテキストであるため, 要素名及びテキストに関する情報の取得は利用するハッシュ関数と暗号化アルゴリズムに依存する.

(2) 構造に対する攻撃

① DTBL と ITBL の EncryptedData に対する既知平文攻撃と選択平文攻撃

構造に関する情報は DTBL と ITBL の EncryptedData に含まれているが，2.2.1 節で述べたように平文にランダムな文字列を追加して暗号化されており平文は入手できない. そのため, 攻撃はできない.

② DTBL の Loc に対する攻撃

DTBL と ITBL 構築時にランダムに割り当てられる数値のため, 構造に関する情報を得ることはできない.

(3) RDB サーバとクライアント間のデータのやり取りに対する攻撃

① クライアントから送信されるデータに対する攻撃

2.2.3 節で述べたように, RDB サーバへハッシュのみ送信されるため, 安全性は利用するハッシュ関数と暗号化アルゴリズムに依存する.

② RDB サーバから送信されるデータに対する攻撃

2.1 節で述べたように，RDB サーバから暗号化されたデータが送信されるため, 安全性は利用する暗号化アルゴリズムに依存する.

(4) メモリ上のデータに対する攻撃

DTBL と ITBL はすべてハッシュ化か暗号化された状態でメモリ上にコピーされるため，安全性は利用するハッシュ関数と暗号化アルゴリズムに依存する.

2.4 提案手法の改良について

検索時間の高速化のため以下の改良を行っている．2.5 節で述べる実験はこの改良を適用している．

(1) クライアントが RDB サーバから同じデータを取得する場合，メモリに記憶したデータを利用する改良を行っている．メモリに記憶したデータの更新について，XML 文書は更新されないため一度メモリに記憶したデータの更新は行っていない．検索する度にメモリに記憶したデータは全て削除される．

(2) 2.2.4 節の処理に Al-Khalifa[23]らの Stack-Tree-Desc を利用することで，時間計算量を 2.2.3 節で取得した分岐点と葉の要素数をs，作成した先祖・子孫関係の要素のリスト数をtとした場合 $O(s+t)$とする改良を行っている．

Al-Khalifa[23]らの Stack-Tree-Desc は，木構造中の 2 つのノードが先祖と子孫の関係であるか判定するアルゴリズムである．2 つのノードが先祖と子孫の関係であるかすべての組み合わせで判定する場合，ノード数をnとすると時間計算量は $O(n^2)$となるが，Stack-Tree-Desc は事前にノードを深さ優先探索順でソートすることで，先祖と子孫の関係の判定が必要なノードを減らしている．

2.5 実験

2.5.1 実験環境について

提案手法と Brinkman[3]の手法との性能比較を行うため実験を行った．RDB サーバは，Intel Xeon 2GHz×2，メモリ 4GB，CentOS 6.0，クライアントは，Intel Core i5 1.9GHz，メモリ 8GB，Windows 8.1 の PC を 100Mbps のネットワークで接続して実験を行った. RDB は MySQL 5.1.52 を使用し，DTBL の Loc と ITBL の Hash はインデックスを構築した．提案手法と Brinkman[3]の手法のプログラムは, プログラミング言語を Perl, ハッシュ関数を SHA1, 暗号化アルゴリズムは DES, XPath プロセッサは XML::XPath[24]を採用した.

XML 文書は XMark[25]で公開されている XML 文書作成ソフトウェアで 1MB，10MB，100MB の XML 文書を作成した．XML 文書作成ソフトウェアは図 24 のようにオークションに関する情報を想定した XML 文書を作成する．XML 文書の詳細について，表 2(a)は作成した XML 文書の要素数等，表 2(b)は作成した XML 文書の要素数等の表，図 25(a)と(b)と(c)は作成した XML 文書の深さ毎の要素数のグラフを示す.

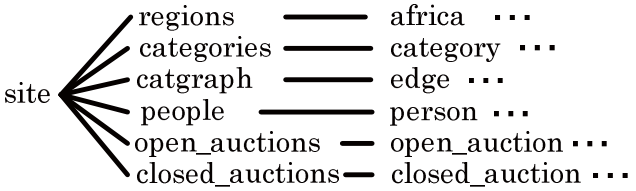


図 24　XML 文書の構造

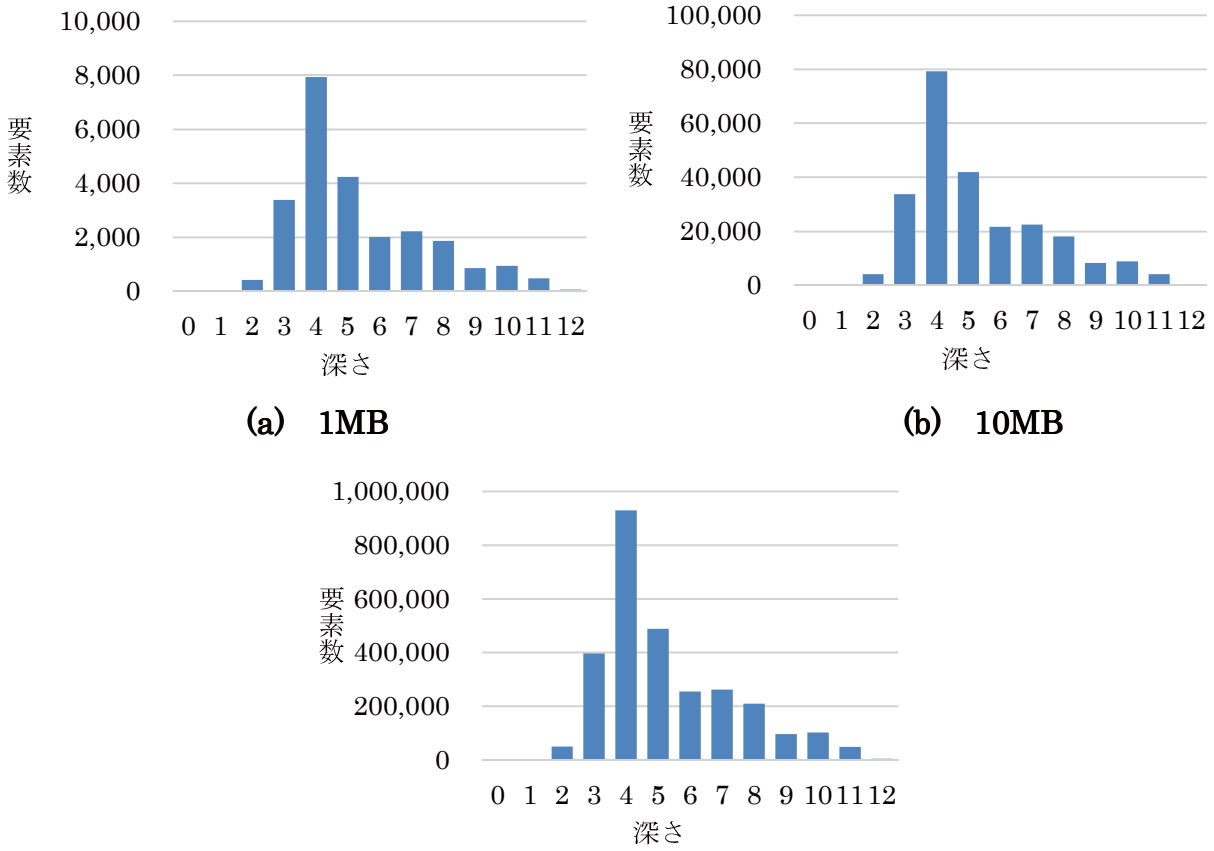
表 2　作成した XML 文書

	総要素数	要素種類数	深さ	平均の深さ
1MB	24,413	7,691	12	5.30
10MB	243,536	65,960	12	5.28
100MB	2,841,234	438,063	12	5.27

(a)

深さ	0	1	2	3	4	5	6	7	8	9	10	11	12
1MB	1	6	421	3,389	7,927	4,239	2,007	2,226	1,866	859	940	474	58
10MB	1	6	4,191	33,793	79,296	41,936	21,696	22,582	18,137	8,363	8,914	4,198	423
100MB	1	6	49,256	396,559	928,992	488,098	255,753	262,927	208,693	96,021	101,796	48,185	4,947

(b)



(a) 1MB　　(b) 10MB

(c) 100MB

図 25　作成した XML 文書

実験は，提案手法と Brinkman[3]の手法に対していくつかの XPath を検索することで行った．検索した XPath を表 3 に示す．結果はネットワーク上の通信量と PC 上の他のプログラムによる影響を少なくするため 3 回測定した検索時間の平均値とした．

性能比較の評価について，提案手法と Brinkman[3]の手法は扱うことができる XPath の範囲が異なっているため, 性能比較の評価は提案手法が扱うことができる XPath の範囲内での評価となる.

表 4 に提案手法と Brinkman[3]の手法で扱うことができる XPath のロケーションステップの範囲を示す. 項目について，Axis は軸の名称，Node Test はノードの指定範囲，Predicate はデータ型(DataType)と演算子(Operator)と関数(Function)を示す. ×は扱うことができないことを示す. 表 4 の①は XPath の先頭のみ, ②は子要素のみ扱うことができることを示す.

表 3　検索する XPath

Query	XPath
Q1	/site
Q2	/site/regions
Q3	/site/regions/asia
Q4	/site/regions/asia/item
Q5	/site/regions/asia/item/description
Q6	/site/regions/africa/item/description
Q7	/site/regions/europe/item/description
Q8	/site/regions/australia/item/description
Q9	/site/regions/namerica/item/description
Q10	/site/regions/samerica/item/description
Q11	/site/people/person[name][emailaddress][phone]
Q12	//item
Q13	//closed_auction/annotation[author][description][happiness]
Q14	//description/text

表 4　Brinkman[3]の手法と提案手法が対応する XPath の比較

LocationStep		Brinkman	Proposed
Axis		child, descendant-or-self	child, descendant-or-self①
Node Test		ノード名	ノード名
Predicate	DataType	文字列型, ノード集合②	ノード集合②
	Operator	=	×
	Function	×	×

2.5.2 実験結果

表 5 と図 26 に提案手法と Brinkman[3]の手法の検索時間の表とグラフを示す. 表 5 の(a)と図 26 の(a1)と(a2)は 1MB, 表 5 の(b)と図 26 の(b1)と(b2)は 10MB, 表 5 の(c)と図 26 の(c1)と(c2)は 100MB の検索時間を示す. 表 5 の項目について, Brinkman は Brinkman[3]の手法の実行時間, Proposed の Client は提案手法のクライアントの実行時間, RDB Server は RDB サーバとクライアントとの通信時間を含む提案手法の RDB サーバの実行時間, Total は Client と RDB Server の実行時間, Matches は XPath とマッチした XML 文書中の部分木の数を示す.

表 5 について, Brinkman[3]の手法と提案手法を比較すると, (a)はすべての Q において提案手法の方が約 2.6～41.5 倍速い. (b)は Q4 を除いて提案手法の方が約 1.2～48.7 倍速い. (c)は Q4,5,6,8,10 を除いて提案手法の方が約 1.1～60.1 倍速い. この理由として, Brinkman[3]の手法は XML 文書のすべての要素及びテキストを対象に, XPath にマッチする要素及びテキストを絞り込むことで検索する. 対して, 提案手法は XPath とマッチングする部分木を対象に, XPath と部分木をマッチングすることで検索する. 提案手法は Brinkman[3]の手法に比べて, RDB サーバが処理する要素及びテキストの数が少ないため速いと考えられる.

表 5(b)の Q4 と(c)の Q4,5,6,8,10 について, 提案手法は XPath に分岐点がない場合, XPath で一番深さの大きい要素と同じ要素名の要素及びテキストのパスを部分木のデータとして取得する. そのため, XPath に分岐点がなく, XPath で一番深さの大きい要素と同じ要素名の要素及びテキストが多い場合, Brinkman[3]の手法に比べて RDB サーバが処理する要素及びテキストの数が多くなり, Brinkman[3]の手法より遅くなったと考えられる.

表 5　実験結果

Query	Brinkman (ms)	Proposed (ms)			Matches
		Client	RDB Server	Total	
Q1	854	26	308	334	1
Q2	2,069	16	51	67	1
Q3	2,665	17	47	64	1
Q4	4,130	55	59	114	17
Q5	4,844	151	103	254	17
Q6	5,264	165	89	254	4
Q7	5,767	151	92	243	51
Q8	5,764	146	92	238	18
Q9	5,641	148	88	236	85
Q10	5,323	152	101	253	8
Q11	7,847	102	107	209	116
Q12	3,986	50	59	109	183
Q13	8,044	149	144	293	81
Q14	4,821	421	235	656	257

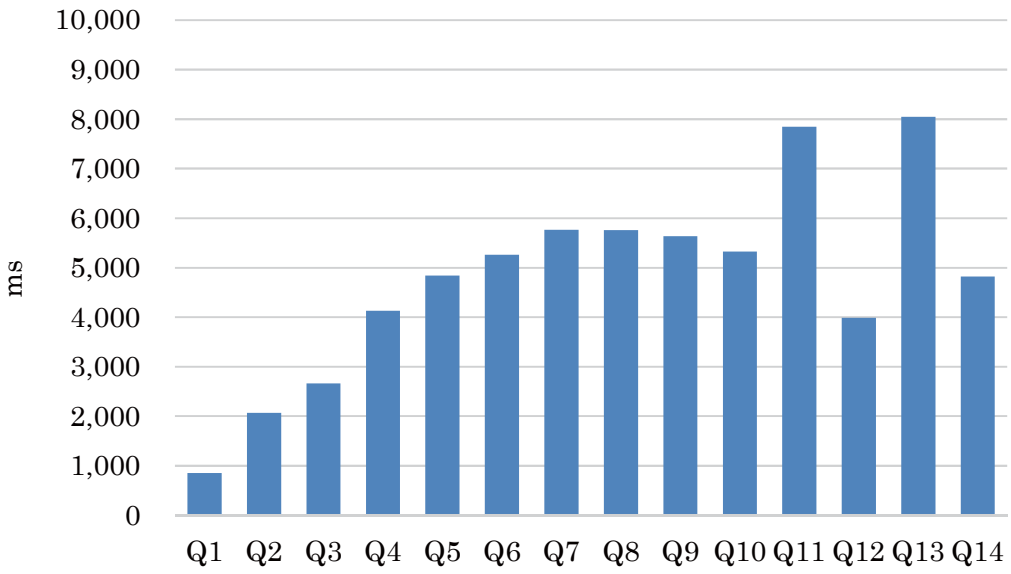
(a)　1MB

Query	Brinkman (ms)	Proposed (ms)			Matches
		Client	RDB Server	Total	
Q1	1,022	15	324	339	1
Q2	1,878	15	47	62	1
Q3	3,176	20	66	86	1
Q4	3,638	534	5,103	5,637	170
Q5	7,014	1,532	4,350	5,882	170
Q6	5,808	1,366	835	2,201	46
Q7	9,609	1,578	867	2,445	510
Q8	7,777	1,437	818	2,255	187
Q9	12,295	1,434	866	2,300	850
Q10	6,606	1,477	842	2,319	85
Q11	40,706	985	1,368	2,353	1,056
Q12	27,289	388	172	560	1,848
Q13	49,210	1,419	1,023	2,442	828
Q14	47,739	4,010	2,711	6,721	2,652

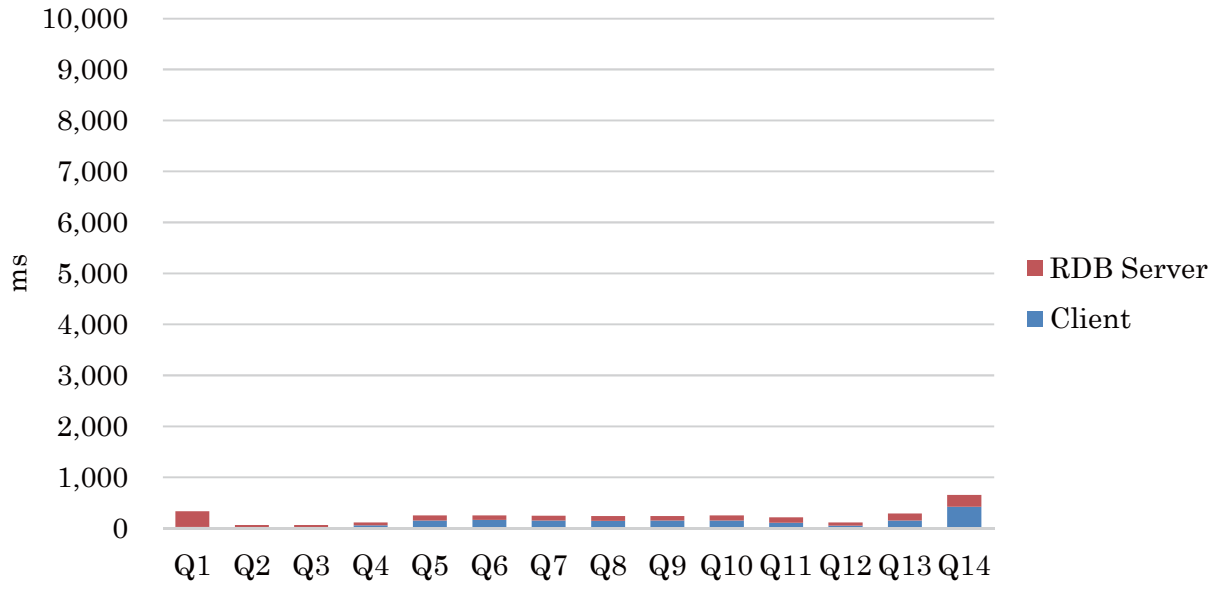
(b)　10MB

Query	Brinkman (ms)	Proposed (ms)			Matches
		Client	RDB Server	Total	
Q1	1,926	15	400	415	1
Q2	3,496	16	100	116	1
Q3	6,014	16	84	100	1
Q4	8,381	4,807	122,853	127,660	2,000
Q5	21,861	16,560	127,279	143,839	2,000
Q6	18,019	16,457	10,914	27,371	550
Q7	30,190	16,443	10,905	27,348	6,000
Q8	21,528	16,343	11,070	27,413	2,200
Q9	38,691	16,396	11,075	27,471	10,000
Q10	17,235	16,010	11,190	27,200	1,000
Q11	108,901	9,470	30,207	39,677	12,679
Q12	378,154	4,100	2,202	6,302	21,750
Q13	438,275	15,911	12,944	28,855	9,750
Q14	402,987	47,591	37,960	85,551	31,368

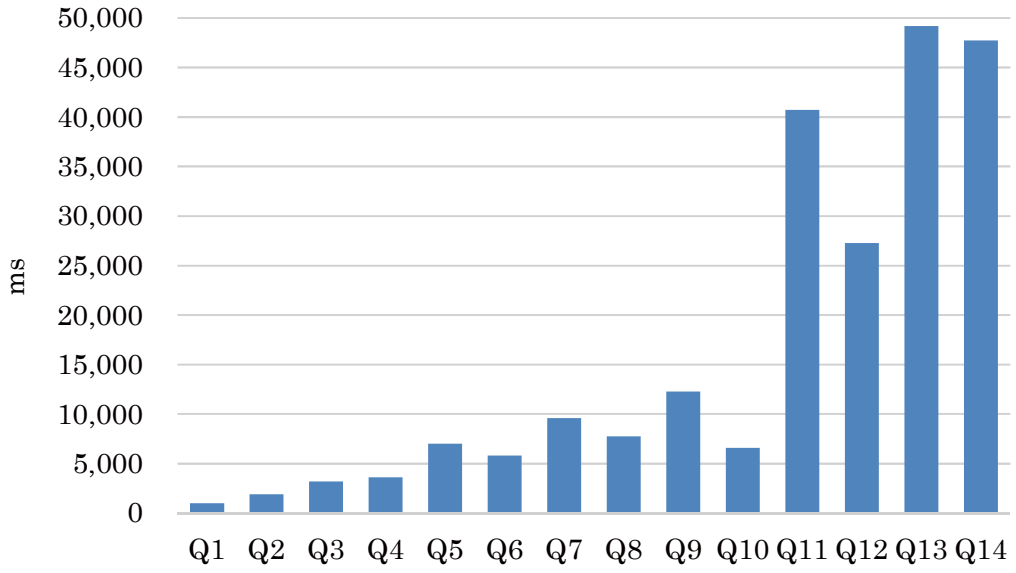
(c) 100MB



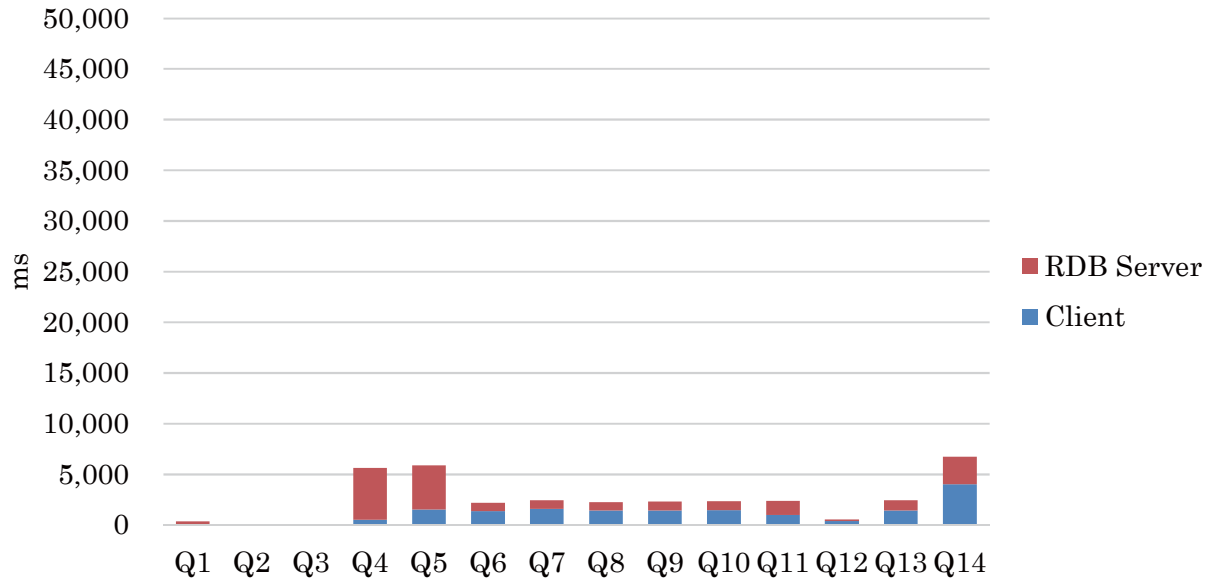
(a1)　Brinkman[3]の手法 1MB



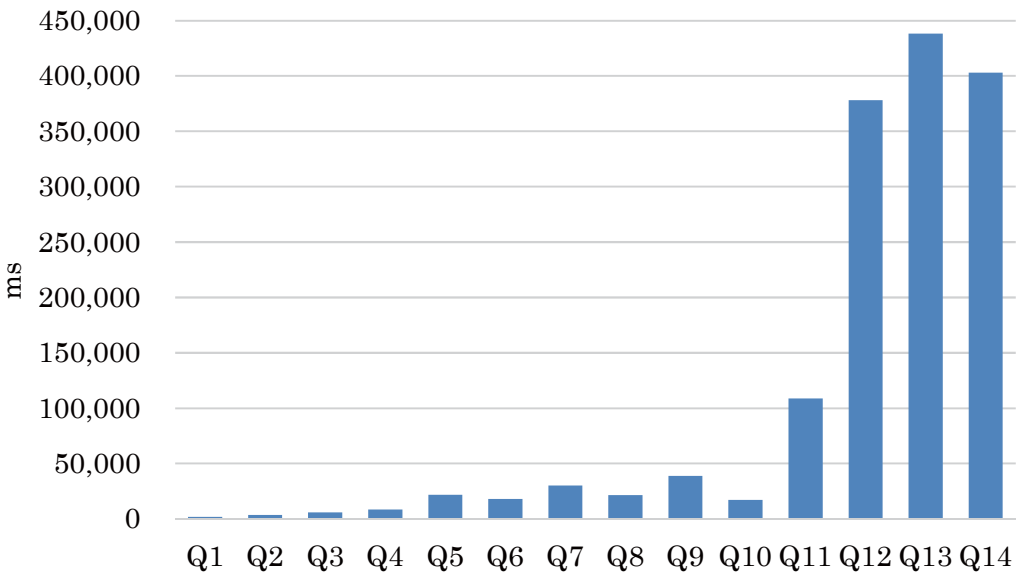
(a2)　提案手法 1MB



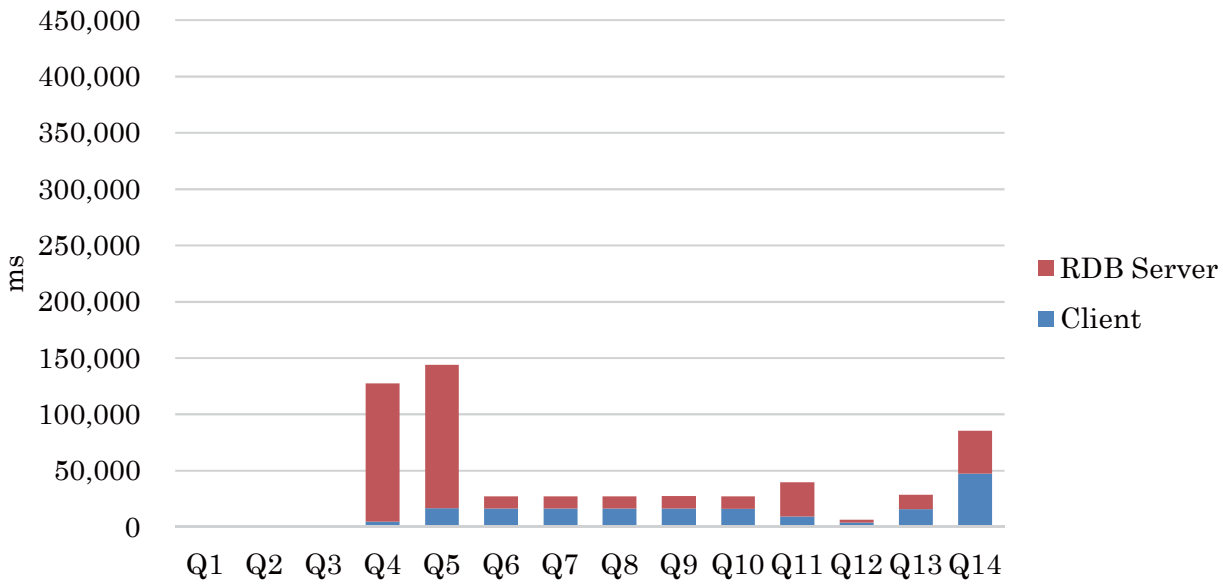
(b1)　Brinkman[3]の手法　10MB



(b2)　提案手法　10MB



(c1)　Brinkman[3]の手法　100MB



(c2)　提案手法　100MB

図 26　実験結果

第 3 章

キーワードを用いた暗号化 XML 文書検索手法

3.1 暗号化 XML 文書検索システム

1.3 節で述べた問題点に対応した，XML 文書を暗号化したままで検索する暗号化 XML 文書検索システム(以下システム)を以下に提案する．

システムの概要を図 27 に示す．前処理として XML 文書を RDB サーバに登録するため，利用者は XML 文書と暗号化に使用する鍵(以下 K)をクライアントに入力する．クライアントは K を用いて XML 文書の暗号化を行い，暗号化されたデータを RDB サーバに登録する．検索する場合は利用者がキーワードと K をクライアントに入力する．クライアントは 3.2 節の手法に基づき，検索結果を返す．なお，システムは XML 文書の更新に対応しない．

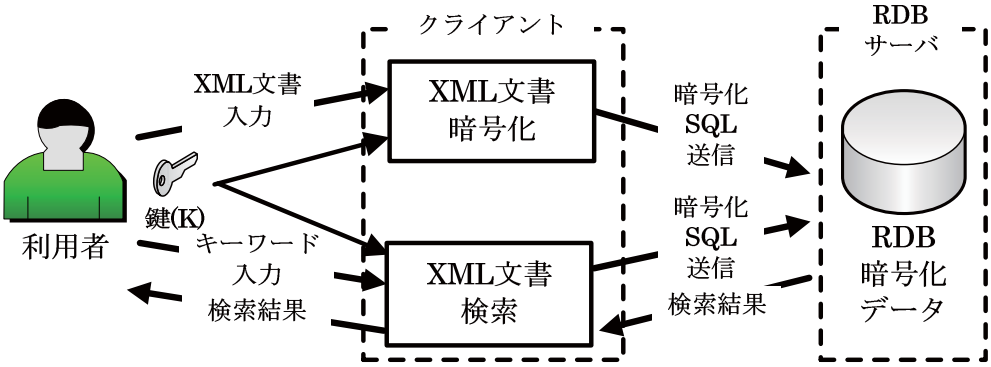


図 27　　暗号化 XML 文書検索システムの構成

3.2 提案手法

3.2.1 概要

提案手法がキーワード(d,e)の SLCA を検索する場合，はじめに XML 文書中の 1 つ目のキーワード(d)と同じ要素名の要素及びテキスト(図 28(a)○)を検索する．次に，検索した要素及びテキスト毎にパスと 2 つ目のキーワード(e)を利用して，2 つ目のキーワードと同じ要素名の要素及びテキスト(図 28(b)灰色○)との LCA(図 28(b)□)を取得する. 最後に, 取得した LCA で子孫に LCA がない要素を SLCA(図 28(c)灰色□)として取得する．キーワードが 3 つの場合，1 つ目と 2 つ目のキーワードの SLCA と 3 つ目のキーワードと同じ要素名の要素及びテキストとの SLCA を取得する. LCA の取得について，提案手法は 1 つ目のキーワードと同じ要素名の要素及びテキスト毎に，根までのパス上の要素及びテキスト(図 29(a)灰色)から子孫に対して，2 つ目のキーワードと同じ要素名の要素及びテキストがあるか検索(図 29(a)矢印)し，2 つ目のキーワードと同じ要素名の要素及びテキスト(図 29(a)灰色○)との LCA(図 29(a)□)を取得する. Xu[5]の手法は 1 つ目と 2 つ目のキーワードと同じ要素名の要素及びテキストをすべて検索し, 1 つ目のキーワードと同じ要素名の要素及びテキスト(図 29(b)○)毎に，根までのパス(図 29(b)灰色)を境に左右から最も近い 2 つ目のキーワードと同じ要素名の要素及びテキスト(図 29(b)灰色○)から 2 つの LCA(図 29(b)△□)を取得し，2 つの LCA のうち子孫の LCA(図 29(b)□)を取得する．提案手法は 1 つ目のキーワードと同じ要素名の要素及びテキストの周りから LCA を探すため, Xu[5]の手法と比べて検索する要素及びテキストの数は少ない.

LCA はすべてのキーワードを含む最小部分木の根を指すため，キーワードと同じ要素名の要素及びテキストから根までのパス上にある. そのため，キーワードと同じ要素名の要素及びテキストの 1 つから XML 文書の根に向かうパス上の要素を対象に，要素を根とする部分木にすべてのキーワードと同じ要素名の要素及びテキストがあるか調べることで LCA を漏らすことなく探すことができる.

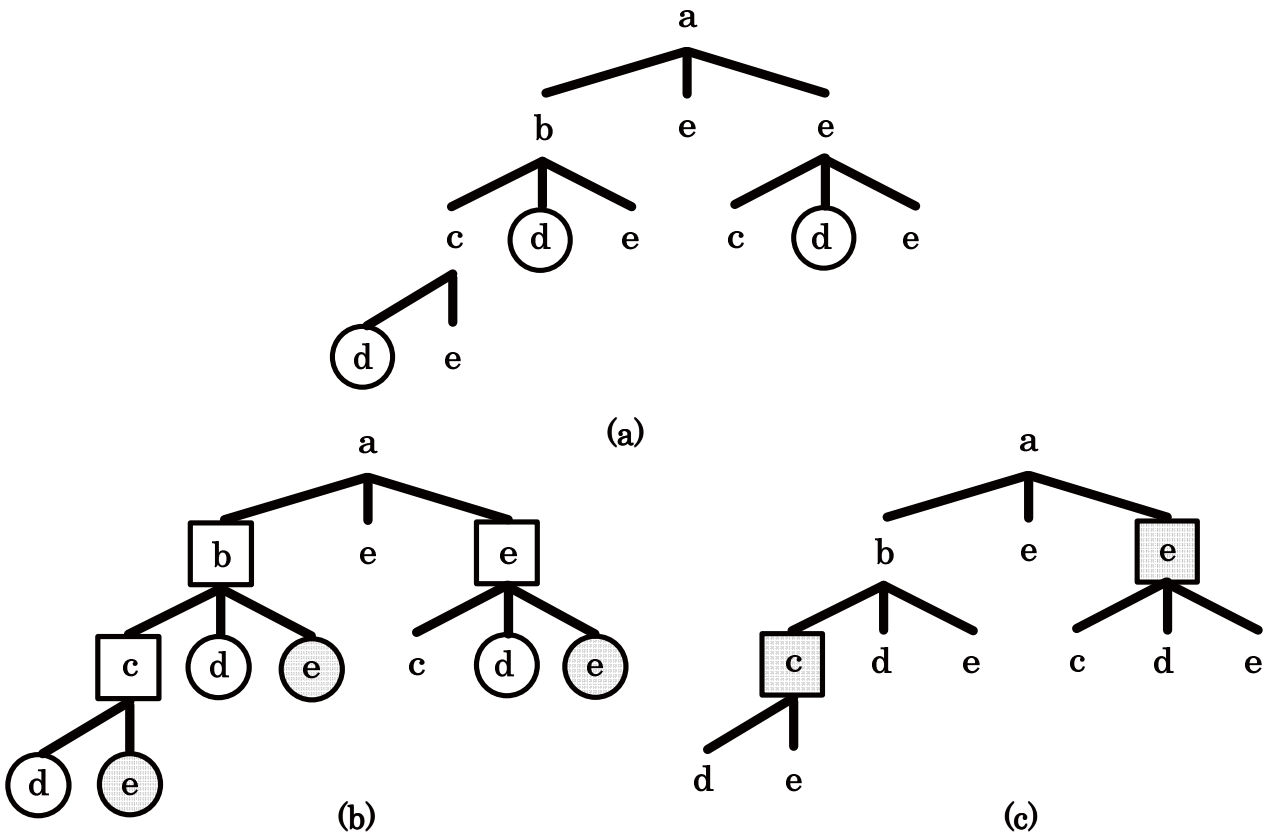


図 28　提案手法

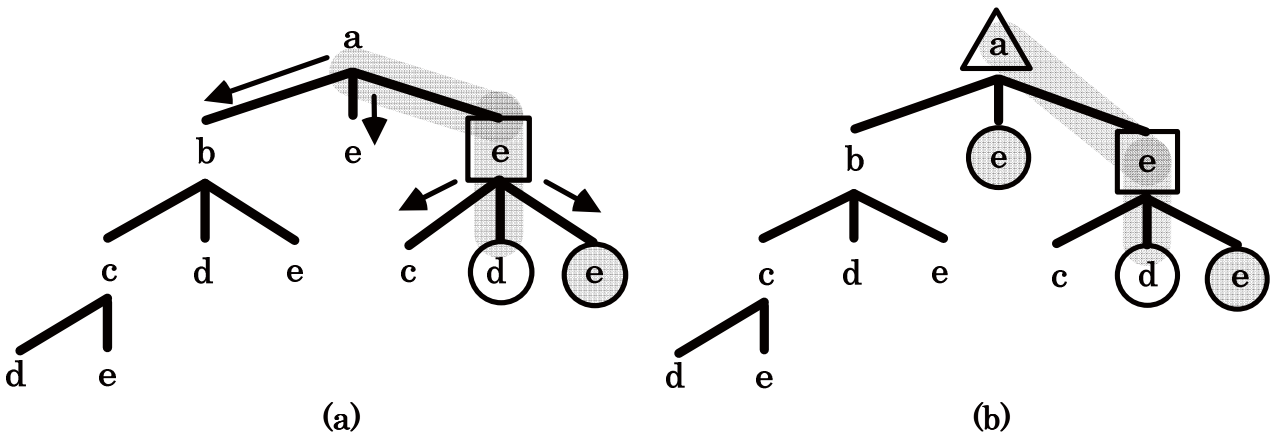


図 29　LCA の取得

システムは，クライアントと RDB サーバとの間で以下のデータのやり取りを行う．

(1) XML 文書から Bloom Filter と暗号化されたデータで構成されたデータテーブルを作成する(図 30(a)①).

(2) キーワードと同じ要素及びテキストを取得するため，要素名及びテキストとパスに関するデータから Bloom Filter を作成し，作成した Bloom Filter と同じデータが含まれるデータテーブルに対応した暗号化されたデータを取得する(図 30(b)①②④).

(3) (2)で取得した暗号化されたデータから LCA を取得し LCA から SLCA を取得する(図 30(b)③).

2 つ目のキーワードと同じ要素名の要素及びテキストの検索について，提案手法はデータ登録時に要素及びテキスト毎に要素名及びテキストとパスと DeweyOrder を登録した Bloom Filter を作成する．キーワードと同じ要素及びテキストを検索する場合，キーワードから BloomFilter を作成し，それを含むデータを検索する．パスを含む要素を検索する場合，文字列(/a/b/c)のまま Bloom Filter に登録すると，/a/b のパスを含んでいる要素を検索することができない．そのため，/a/b/c を a:0,b:1,c:2 と要素名と深さで構成されるデータを Bloom Filter に登録する．/a/b を含む要素を検索する場合，a:0,b:1 を含む要素を検索することで，/a/b のパスを含んでいる要素を検索することができる．

暗号化の際，平文と暗号文から鍵を推測されないようにするため，2.2.1 節の式(3)と式(4)に従って暗号化と復号を行っている．

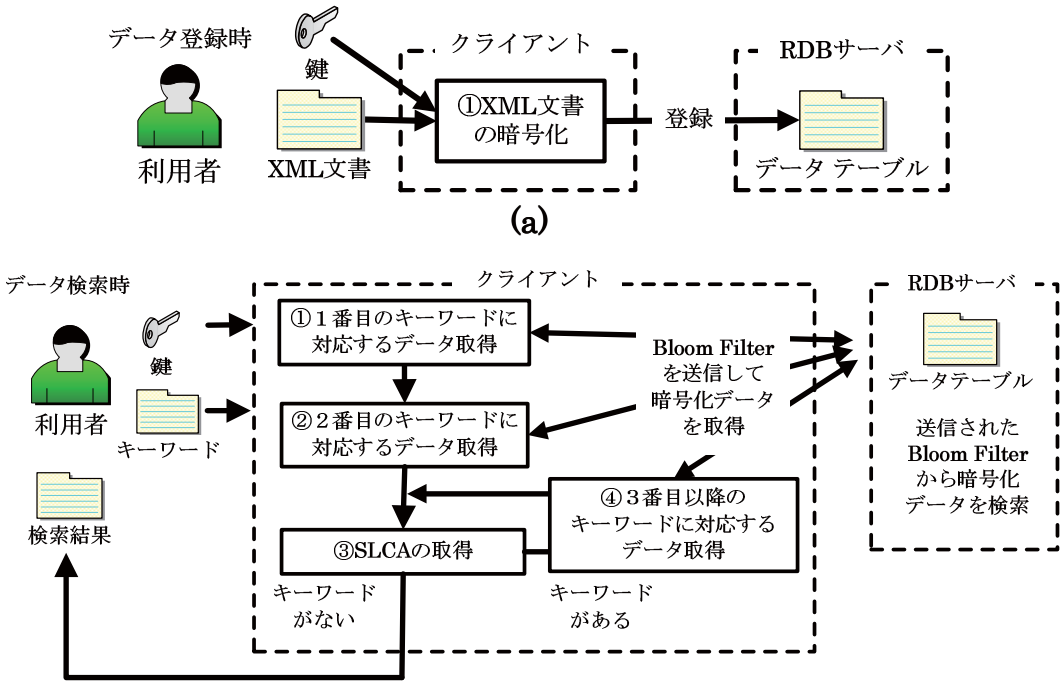


図 30　提案手法

3.2.2 節から 3.2.4 節の提案手法の詳細で時間的計算量を述べるが，時間的計算量で使われる変数について，XML 文書中のすべての要素及びテキストの数をm，1 つ目のキーワードと同じ要素名の要素及びテキストの数をn，2 つ目のキーワードが見つかるまでの検索回数をo，LCA の数をpとしている．

3.2.2 XML 文書の暗号化

XML 文書を RDB サーバ に登録するため，データテーブル(以下 DTBL)を構築する．図 31(a)を XML 文書とする DTBL を図 32 に示す．図 32 について，DTBL の EncryptedData の括弧内は平文の EncryptedData である．データが長いため DTBL の Bloom Filter と EncryptedData の暗号列の途中を省略している．図 31(b)は(a)の XML 文書の要素及びテキストをノードとして DeweyOrder でラベリングし，テキストに下線を引いた木を示す．説明の簡略化のため，XML 文書中の要素名及びテキストそのものをキーワードとし，キーワードと同じテキストと言った場合，キーワードと内容が完全に一致するテキストを意味するものとする．

DTBL は，要素名及びテキストに K を追加してハッシュ化したものと要素及びテキストの位置情報と DeweyOrder が登録された Bloom Filter，2.2.1 節で述べた手法で暗号化されたデータである EncryptedData で構成されている．位置情報はパス上の要素及びテキストに対して要素名及びテキストと深さを”:”でつなげたものである．DeweyOrder が 0.0.2.0 の位置情報は a:0,b:1,f:2,c:3 であり，それぞれを Bloom Filter に登録している．DeweyOrder は Bloom Filter がすべて異なるようにするため登録している．この処理における時間計算量は，DTBL は XML 文書中のすべての要素及びテキストから構築されるため $O(m)$である．

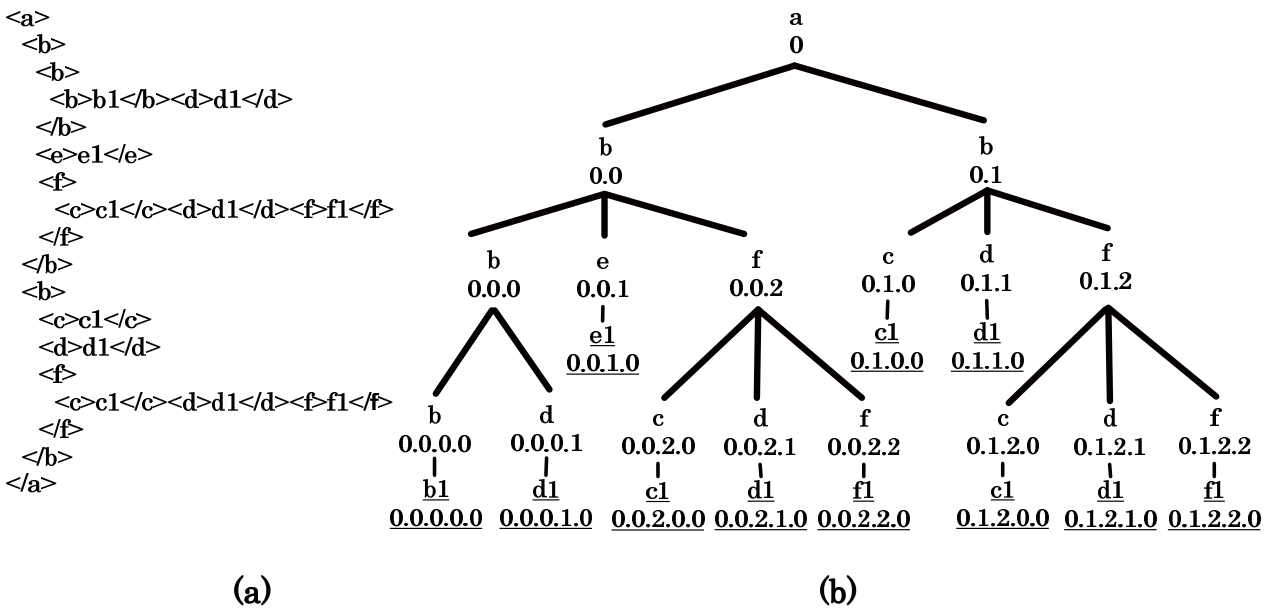


図 31　XML 文書と木構造

Bloom Filter	EncryptedData
01001...00011	vih0I…etA== (b1,0.0.0.0.0, (a:0,b:1,b:2,b:3,b1:4))
00000...00011	FX3Fc…tgtCD (b,0.0.0.0, (a:0,b:1,b:2,b:3))
00101...00011	EvXkk…kHw== (d1,0.0.0.1.0, (a:0,b:1,b:2,d:3,d1:4))
⋮	
Bloom Filter	EncryptedData
00101...00110	qzUUB…c2+A= (b,0.1, (a:0,b:1))
00100...00000	97OwL…K+22z (d1,0.1.1.0, (a:0,b:1,d:2,d1:3))
00000...00001	uAzpM…OU4W7 (f,0.1.2, (a:0,b:1,f:2))

図 32　図 31(a)の XML 文書に対応する DTBL

3.2.3　キーワードに対応するデータ取得

キーワードと同じ要素名の要素及びテキストの取得を以下の(1)～(3)の手順で行う．キーワードが 3 つ以上の場合は，今までに取得した SLCA に対して(3)を行う．Bloom Filter は偽陽性のため誤ってデータを取得する場合があるため，取得後に要素名及びテキストがキーワードと一致するか確認している．(3)で検索結果はクライアントに記憶するため，以前に作成した Bloom Filter で再度検索は行わない．この処理における時間計算量は，(1)と(2)は 1 回実行し，(3)は 1 つ目のキーワードと同じ要素名の要素及びテキスト毎に 2 つ目のキーワードと同じ要素名の要素及びテキストが見つかるまで繰り返し実行するため $O(n \times o)$である．(1)と(2)で実行する部分は 1 つ目のキーワードと同じ要素名の要素及びテキストの数と非常に少ないため時間計算量から無視している．

(1) 1 つ目のキーワードに K を追加してハッシュ化したものから Bloom Filter を作成し，Bloom Filter を含む DTBL のデータに対応した EncryptedData を取得する(図 33).
(2) (1)で取得した EncryptedData 毎に 2.2.1 節で述べた手法で復号し，最も深さが大きいものを除いた位置情報を取得する(図 34①).
(3) 2 つ目のキーワードに K を追加してハッシュ化したものと(2)で取得した位置情報から Bloom Filter を作成し，Bloom Filter を含む DTBL のデータに対応した EncryptedData を取得する(図 34②)．Bloom Filter を含む DTBL のデータがなかった場合は，(2)で取得した位置情報から最も深さが大きいものを除いて(3)を繰り返す.

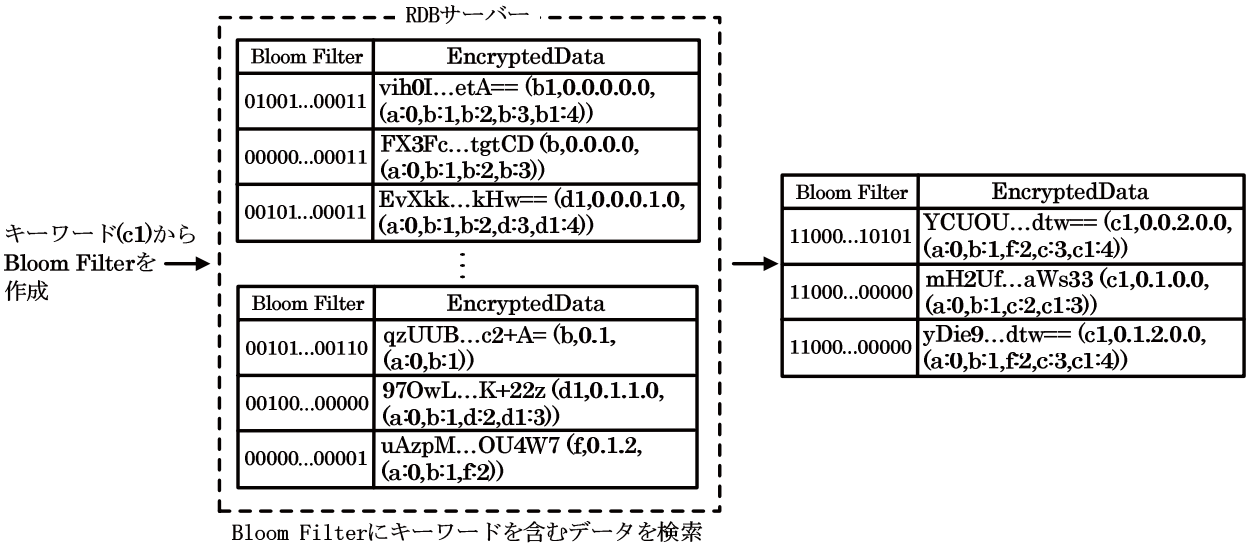


図 33　DTBL からキーワード(c1)を含むデータを取得

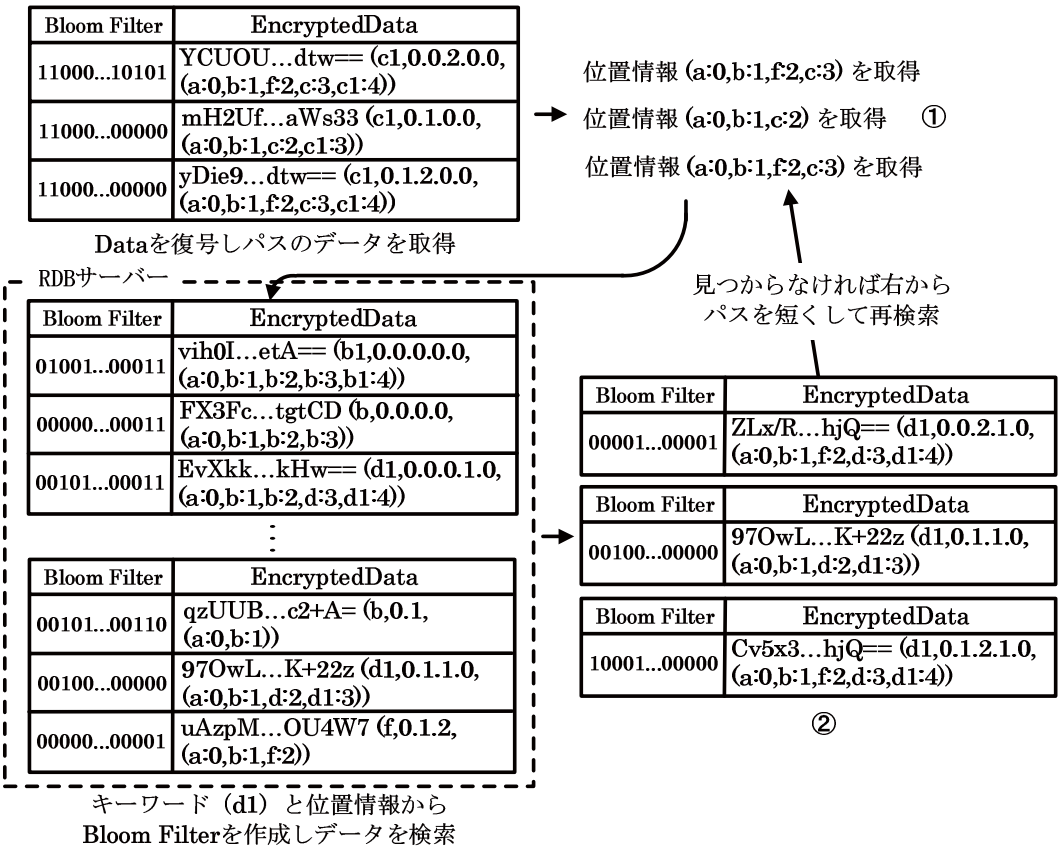


図 34　DTBL からキーワード(d1)を含むデータを取得

3.2.4　SLCA の取得

3.2.3 節で取得したデータの EncryptedData を 2.2.1 節で述べた手法で復号し，EncryptedData 中の DeweyOrder を利用して LCA を作成する．作成した LCA の子孫に LCA がないものを SLCA とする(図 35)．この処理はすべてクライアントで行われる．この処理における時間計算量は LCA 同士で子孫に LCA があるか調べるため $O(p^2)$である．

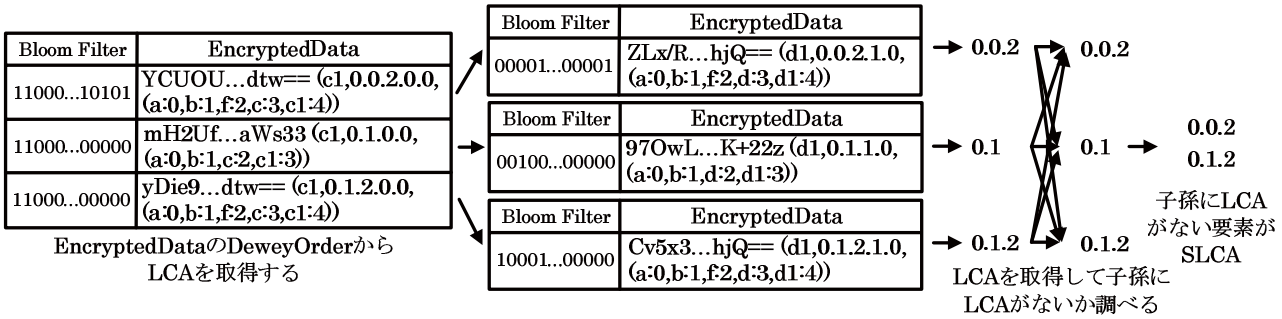


図 35　SLCA の取得

3.3　安全性について

提案手法の安全性の評価を，想定される攻撃に対する対応策を分析することで行った．

(1) 要素に対する攻撃

① DTBL の EncryptedData に対する既知平文攻撃と選択平文攻撃

既知平文攻撃を行うには要素及びテキストに関する情報がある平文, 選択平文攻撃を行うには任意の要素及びテキストに関する情報がある平文が必要だが，2.2.1 節で述べたように平文にランダムな文字列を追加して暗号化されており平文は入手できない. そのため，攻撃はできない．

② DTBL の Bloom Filter に対する攻撃

3.2.2 節で述べたように Bloom Filter は要素名及びテキストに K を追加してハッシュ化したものと要素及びテキストの位置情報と

DeweyOrder を登録しているため，安全性は Bloom Filter に依存する．

(2) RDB サーバとクライアント間のデータのやり取りに対する攻撃

① クライアントから送信されるデータに対する攻撃

3.2.3 節で述べたように，RDB サーバへ Bloom Filter のみ送信されるため，安全性は利用する Bloom Filter に依存する．

② RDB サーバから送信されるデータに対する攻撃

3.1 節で述べたように，RDB サーバから暗号化されたデータが送信されるため，安全性は利用する暗号化アルゴリズムに依存する．

(3) メモリ上のデータに対する攻撃

DTBL はすべて Bloom Filter か暗号化された状態でメモリ上にコピーされるため，安全性は Bloom Filter と暗号化アルゴリズムに依存する．

3.4 提案手法の改良について

検索時間の高速化のため，以下の改良を行っている．3.5 節で述べる実験はこの改良を適用している．

3.2.3 節で実行する SQL は，論理積で Bloom Filter に 1 が設定されているデータを抽出し，排他的論理和で Bloom Filter に 1 が設定されている部分が 0 の場合は，データを取得しない条件としていた(図 36 (a)①)．しかし，実験で使用した RDB は，条件に論理積や排他的論理和等の論理演算を使用するとインデックスを使用しないことがわかった．そのため，Bloom Filter のビット配列が列と対応するテーブル DTBL_INDEX を構築し(図 36(b))，条件を Bloom Filter のビット位置に対応する列が 1 と等しいか比較する SQL とすることでインデックスを利用できるようにした(図 36(a)②)．ただし，この手法は Bloom Filter のビット配列の長さ分のインデックスが必要なため，DTBL_INDEX の構築に時間がかかる．

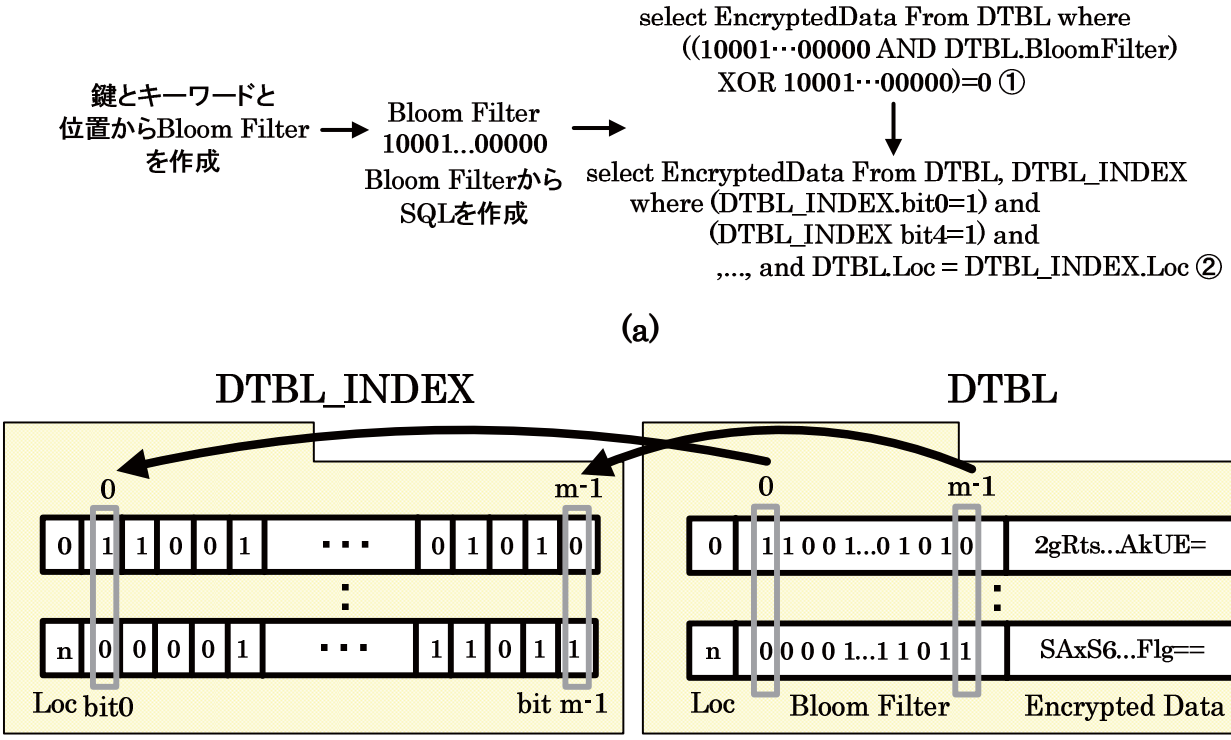


図 36　Bloom Filter のインデックス構築

3.5 実験

3.5.1 実験環境について

提案手法と Xu[5]と暗号化 Xu[5]の手法との性能比較を行うため実験を行った. RDB サーバは, Intel Xeon 2GHz×2, メモリ 4GB, CentOS 6.0, クライアントは, Intel Core i5 1.9GHz, メモリ 8GB, Windows 8.1 の PC を 100Mbps のネットワークで接続して実験を行った. RDB サーバは MySQL 5.1.52 を使用し, 提案手法は 3.4 節で述べた手法で Bloom Filter, Xu[5]と暗号化 Xu[5]の手法は要素名及びテキストでインデックスを構築した. 提案手法と Xu[5]と暗号化 Xu[5]の手法のプログラムは, プログラミング言語を Python, ハッシュ関数を SHA1, 暗号化アルゴリズムは AES を採用した. XML 文書は 2.5.1 節で述べた 100MB の XML 文書を使用した.

Bloom Filter について, 偽陽性が発生する確率を 0.01%未満と想定し 1.6 節の式(2)を用いて, 登録する要素数は要素名及びテキストと位置情報と DeweyOrder を想定し(n=5), 長さ 108(m=108)のビット配列, 8 個のハッシュ関数(k=8)を設定した.

実験は, 提案手法と Xu[5]と暗号化 Xu[5]の手法に対して, 同じ要素名及びテキストの出現数と検索するキーワード数を変化させて行った. 実験結果は検索するキーワードをランダムに 20 回選んで測定した平均値とした. 実験と同じ出現数の要素名及びテキストは少ないため, 実験の出現数の 0.8 倍～1.2 倍の出現数の要素名及びテキストをキーワードの対象としている.

3.5.2 実験結果

表 6 と図 37 に提案手法と Xu[5]と暗号化 Xu[5]の手法の検索時間の表とグラフを示す．表 6 の(a)と図 37 の(a1)と(a2)はそれぞれ異なる出現数の 2 つのキーワード，表 6 の(b)と図 37 の(b1)と(b2)はすべて同じ出現数の 2 つと 4 つのキーワードによる検索時間を示す．表 6 の項目について，キーワードは出現数をキーワード数”,”でつなげたもの，Xu の平文は Xu[5]の手法の実行時間, 暗号化は暗号化 Xu[5]の手法の実行時間, 提案手法の RDB はサーバとクライアント間の通信を含む RDB サーバの実行時間, Decrypt はデータを復号する実行時間, Other は RDB と Decrypt 以外の実行時間, Total は RDB と Decrypt と Other の実行時間，Text は RDB から入手した要素及びテキストの数を示す.

検索時間について，提案手法と Xu[5]の手法と比較すると，表 6(a)は Xu[5]の手法に比べて約 217.6～15511.8 倍遅く，表 6(b)は Xu[5]の手法に比べて約 1548.0～17582.0 倍遅い．この理由として，Xu[5]の手法はキーワードと同じ要素名の要素及びテキストを 1 回で検索しているが，提案手法はキーワードを Bloom Filter で検索していること，2 つ目のキーワードの検索は Bloom Filter で複数回の検索を行っているためと考えられる．提案手法は検索時間のほとんどを RDB が占めている．Text について，Xu[5]の手法は 2 つ目のキーワードと同じ要素名の要素及びテキストをすべて RDB サーバから入手するが，提案手法は 1 つ目のキーワードと同じ要素及びテキスト毎に根までのパス上の要素から子孫に向かって，2 つ目のキーワードと同じ要素名の要素及びテキストを検索するため Xu[5]の手法と比べて少ない.

表 6(a)について，キーワード 10，100 と 100，1000 は他のキーワードと比べて Text が少ないが検索時間が遅いのは, 出現数が少ないためキーワードと同じ要素名の要素及びテキストの位置が離れており，Bloom Filter で複数回検索をしているためと考えられる．キーワード 1000，100000 は他のキーワードと比べて Text が多いが検索時間が速いのは, 出現数が多い

ためキーワードと同じ要素名の要素及びテキストの位置が近く，Bloom Filter の検索回数が少ないためと考えられる．

表 6(b)について，キーワード 1000，1000 と 1000，1000，1000，1000 は他のキーワードと比べて Text が多いが検索時間が速いのは，表 6(a)のキーワード 1000，100000 と同じように Bloom Filter の検索回数が少ないためと考えられる．キーワード 1000，1000，1000，1000 より 100，100，100，100 の方の検索時間が遅いが，これは出現数が少ないためキーワードと同じ要素名の要素及びテキストの位置が離れていると考えられる．

暗号化 Xu[5]の手法について，Xu[5]の手法と比べて約 1.0～3.4 倍遅いが，暗号化されたデータを復号する処理があるためである．暗号化以外のアルゴリズムは同じため傾向は Xu[5]の手法と同じである．

SLCA を取得する手法は Xu[5]の手法以外にも Basic Multiway Search(以下 BMS)や Incremental Multiway Search (以下 IMS)[26]やハッシュを用いた手法[27]等がある．提案手法にこれらの手法を適用することで検索時間が改善する可能性について，BMS や IMS は Xu[5]の手法を基に LCA の取得を減らすことで検索を高速にした手法のため，BMS と IMS を適用することで LCA の取得に必要な Bloom Filter の検索が少なくなり検索時間の改善が期待できる．ハッシュを用いた手法は，XML 文書中のすべての要素及びテキストと要素を根とする部分木に含まれるすべての要素及びテキストで構成されるハッシュテーブルを利用して検索する手法であるが，この手法はハッシュテーブルを多くの回数検索する必要があるため，適用しても検索時間の改善は期待できない．

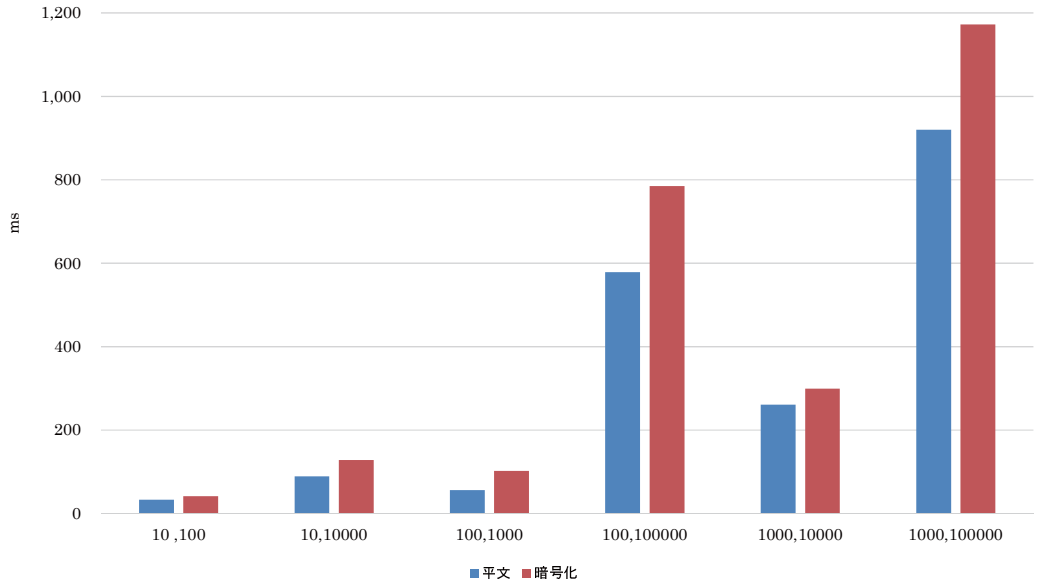
表 6　実験結果

キーワード	Xu			提案手法				
	平文	暗号化	Text	Total (ms)	RDB (ms)	Decrypt (ms)	Other (ms)	Text
	Total (ms)	Total (ms)						
10 ,100	34	41	101	521,686	520,574	295	817	97
10,10000	90	129	11,177	556,502	554,657	427	1,419	10,480
100,1000	56	102	1,040	838,080	836,660	403	1,016	967
100,100000	578	785	90,926	868,249	860,292	461	7,496	78,407
1000,10000	261	299	11,778	375,924	374,042	377	1,504	10,472
1000,100000	920	1,172	94,200	200,110	188,177	533	11,400	44,107

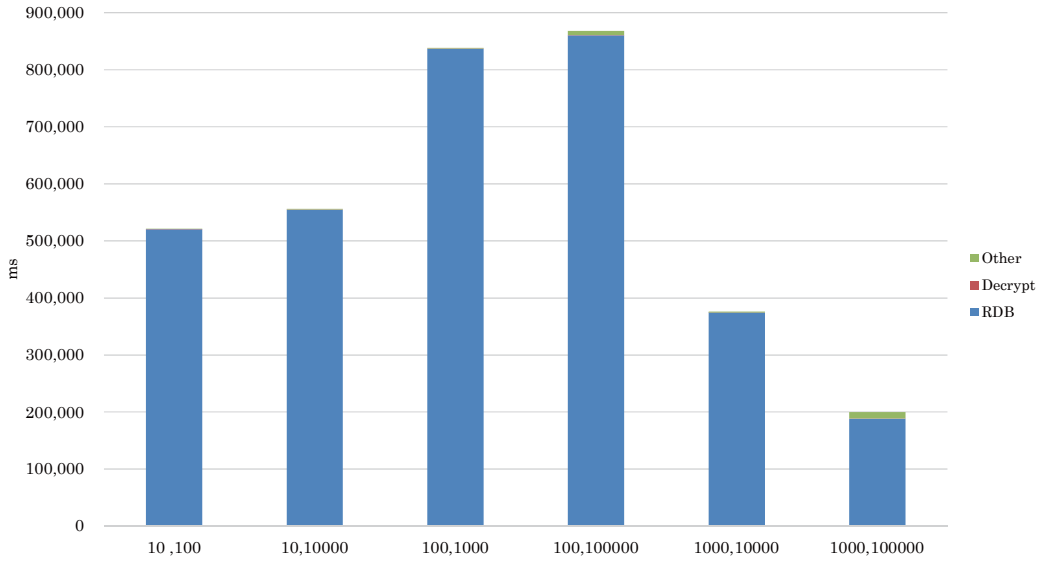
(a)

キーワード	Xu			提案手法				
	平文	暗号化	Text	Total (ms)	RDB (ms)	Decrypt (ms)	Other (ms)	Text
	Total (ms)	Total (ms)						
10 ,10	32	108	19	560,896	560,114	263	519	17
10,10,10,10	72	127	68	546,011	543,461	467	2,083	65
100,100	52	101	188	674,566	673,436	304	826	187
100,100,100,100	63	113	365	840,876	837,327	628	2,921	362
1000,1000	238	248	1,940	340,796	339,269	365	1,161	1,685
1000,1000,1000,1000	263	274	3,865	406,597	404,002	645	1,951	3,525

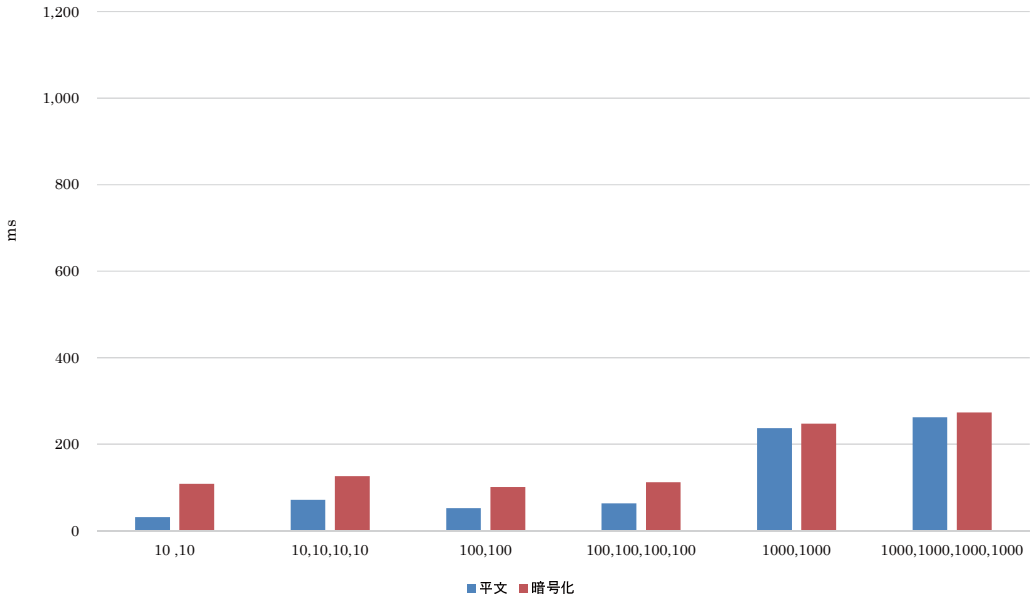
(b)



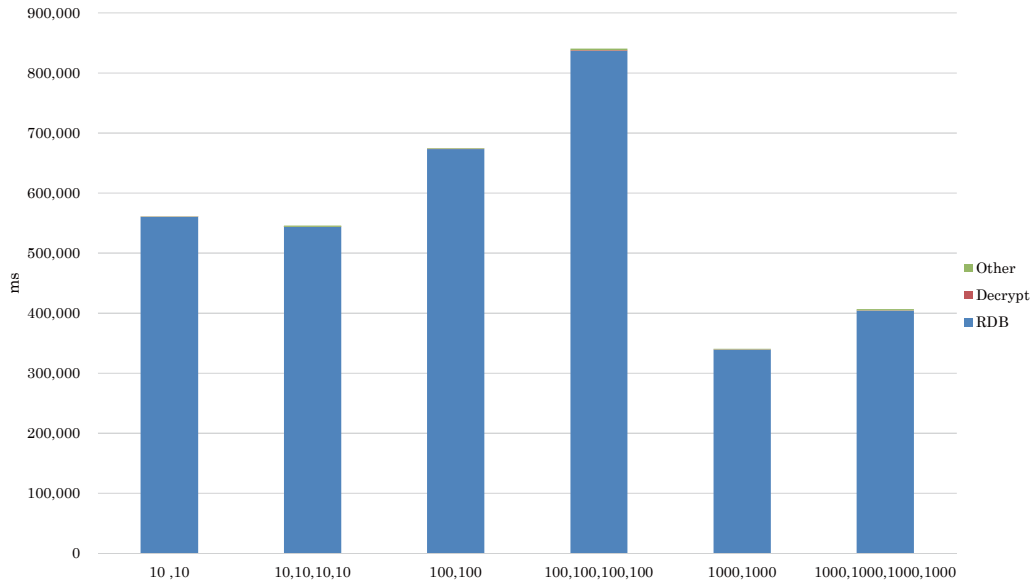
(a1)　Xu[5]と暗号化 Xu[5]の手法



(a2)　提案手法



(b1)　Xu[5]と暗号化 Xu[5]の手法



(b2)　提案手法

図 37　実験結果

第 4 章

結論

4.1 本論文で得られた成果

本論文では，データを暗号化したままで XML 文書を検索する手法について述べた．従来の研究では，XML 文書中の要素の入れ子によって構成される構造が暗号化されていないため，管理者が同じ内容の XML 文書を入手すると構造から検索内容と結果が推測される問題がある．

本論文では，XML 文書の要素だけでなく要素の入れ子によって構成される構造も暗号化したままで検索する手法に関して，それぞれ次のような成果を得た．

(1) XPath を用いた暗号化 XML 文書検索手法

① 以下の手順で手法を実現した．

1) XML 文書からすべての要素及びテキストの情報等が暗号化されたデータテーブル，同じ要素名及びテキストの情報等が暗号化されたインデックステーブルを作成する

2) XPath とマッチングする部分木のデータを取得するため，XPath から分岐点と葉の要素名を取得し，インデックステーブルとデータテーブルを利用して，分岐点と葉の要素名と同じ要素及びテキストのデータを取得する．

3) 2)で取得したデータで分岐点の要素名と同じ要素及びテキストについて，これを根とする部分木に，葉の要素名と同じ要素及びテキストがすべてあるか調べ，あればデータテーブルから要素及びテキストのパスのデータを取得する．

4) 3)で取得したデータを XML 文書に変換し，XPath プロセッサで XPath とマッチするか判定する.

② 以下の検索時間を高速化する改良を行った.

1) クライアントがデータベースから取得したデータの再利用を可能とした.

2) 部分木を XML 文書から検索するアルゴリズムとして Al-Khalifa[23]らの Stack-Tree-Desc を採用した.

③ 提案手法について要素と構造とデータのやり取りとメモリ上のデータの点から安全性を分析し, 提案手法が安全であることを示した.

④ XMark[25]で公開されている XML 文書作成ソフトウェアで作成した XML 文書で, 提案手法と Brinkman[3]の手法との性能比較を行い，提案手法が扱うことのできる XPath の範囲において，一部の XPath 以外は提案手法の方が約 1.2～60.1 倍速い結果となった.

(2) キーワードを用いた暗号化 XML 文書検索手法

① 以下の手順で手法を実現した.

1) XML 文書から Bloom Filter と暗号化されたデータで構成されたデータテーブルを作成する.

2) キーワードと同じ要素及びテキストを取得するため，要素名及びテキストとパスに関するデータから Bloom Filter を作成し，作成した Bloom Filter と同じデータが含まれるデータテーブルに対応した暗号化されたデータを取得する.

3) 2)で取得した暗号化されたデータから LCA を取得し LCA から SLCA を取得する.

② 以下の検索時間を高速化する改良を行った.

1) Bloom Filter の処理にデータベースのインデックスを利用可能とした.

③ 提案手法について要素とデータのやり取りとメモリ上のデータの点から安全性を分析し，提案手法が安全であることを示した.

④ 2.5.1 節で述べたソフトウェアで作成した XML 文書で，Xu[5]と暗号化 Xu[5]の手法との性能比較を行い，検索時間はそれぞれ異なる出現数の 2 つのキーワードの場合は Xu[5]の手法より約 217～15511 倍，すべて同じ出現数の 2 つと 4 つのキーワードの場合は Xu[5]の手法より約 217.6～17582.0 倍遅い結果となった．

(3) 提案手法は以下の情報セキュリティ上の問題に対して安全である．

① 管理者が RDB サーバのファイルを持ち出して解読する．

2.3 節の(1)の①と(2)の①，3.3 節の(1)の①で述べたように，ファイルに攻撃できないため解読できない．

② 管理者がクライアントと RDB サーバの通信を盗聴して解読する．

2.3 節の(3)の①と②，3.3 節の(2)の①と②で述べたように，クライアントからハッシュと Bloom Filter，RDB から暗号化されたデータのみ送信されるため解読できない．

③ 管理者がメモリを読み出して解読する．

2.3 節の(4)，3.3 節の(3)で述べたように，ハッシュ化か暗号化されたままでメモリ上にコピーされるため解読できない．

(4) 1.3 節で述べたデータを暗号化したままで検索する手法のさらなる実用化に向けた課題である「検索される XML 文書と同じ内容の XML 文書を管理者が入手した場合，検索している要素及びテキストの位置と入手した XML 文書を突き合わせることで，検索内容と結果が推測される問題」に対して，提案手法は 4.1 節の(3)の①②③で述べた問題に対して安全なため，管理者は検索している要素及びテキストの位置を知ることはできず課題にある問題は発生しない．そのため，クラウドのような情報セキュリティの管理が難しい情報環境でも，安全な XML 文書の検索環境を構築する場合に提案手法は有効である．

4.2 今後の課題

本論文において，以下の課題があることがわかった．

(1) XPath を用いた暗号化 XML 文書検索手法

① 扱うことのできる XPath に制限がある

XPath と XML 文書を木構造としてとらえて，XPath 中の特定ノードの位置関係が一致する XML 文書中の部分木を検索するため，扱うことのできる XPath に制限がある点が課題である．

XPath の軸について，//(子孫の要素)は XPath の先頭(左端)しか扱うことができない．//は複数の要素を含む場合があり，分岐点があると要素を特定できないためである．葉と同じ要素名の要素及びテキストのパスで共通する部分から分岐点を検索することで対応できるが，提案手法を大きく修正する必要があると考えられる．

XPath のノードテストについて，*(すべての要素を指定)は扱うことができない．*が分岐点と葉の場合は要素を特定できないためである．分岐点から葉の途中に*がある場合は提案手法で扱うことはできる．しかし，分岐点と葉が*の場合は XML 文書中のすべての種類の要素名を対象に検索することになり，XPath プロセッサとマッチングする部分木が非常に多くなってしまう．XPath の軸と同様の手法で対応できるが，提案手法を大きく修正する必要があると考えられる．

XPath の述語について，テキストや属性等の述語を扱うことができない．これは，提案手法が検索を主な目的としていたためである．分岐点と葉と同じ要素名の要素及びテキストを検索する処理で述語を判定する部分を追加することで対応できる．提案手法の修正は少ないと考えられる．

(2) キーワードを用いた暗号化 XML 文書検索手法

① Bloom Filter の処理時間が遅い

3.4 節で述べた DTBL_INDEX は XML 文書中のすべての要素及びテキストの数を行数，Bloom Filter のビット配列の長さを列数とするテーブルである．XML 文書中の要素及びテキストを検索する度に Bloom Filter の処理の一部として DTBL_INDEX 全体が検索される．RDB のインデックスを有効にしてもこの処理時間の遅い点が課題である．処理時間の高速化について，RDB が Bloom Filter の処理に対応する，メモリ上で Bloom Filter を実行する手法が考えられる．RDB が Bloom Filter の処理に対応する手法について, select 文に Bloom Filter の検索機能を追加することで, RDB 内部で論理演算ができるため高速化が期待できる．RDB 内部で Bloom Filter のビット毎に 1 となっている行を把握していれば，検索する Bloom Filter のビットが 1 となっている位置から検索する行を絞り込むことができるため，さらに高速化が期待できる．

メモリ上で Bloom Filter を実行する手法について，RDB の機能を利用してメモリ上にデータを登録する，SSD や Ramdisk 等の HDD に比べて入出力が高速なストレージ上に RDB を構築することで高速化が期待できる．

(3) 共通の課題

① XML 文書の更新に対応していない

これは，提案手法が検索を主な目的としていたためである．XML 文書を RDB に登録する際に要素及びテキスト毎に start を 50 や 100 等と間隔をあけて割り当て，要素及びテキストを追加する場合は間隔をあけた数値を start として割り当てることで対応できるが，提案手法を大きく修正する必要があると考えられる．追加中は start や end 等を修正しているため, 追加する要素及びテキストの子孫は追加が完了するまで検索できない. 何回も要素及びテキストの追加を行うと, 間隔をあけた間の数値をすべて割り当ててしまうため，追加する要素及びテキストの周辺で start を再度割り当て直す作業が必要である．

参考文献

[1] “Amazon web services”, http://aws.amazon.com/jp/.

[2] 金子 静花, 渡辺 知恵美 , 天笠 俊之: “ブルームフィルタを用いたプライバシ保護検索システム Semi-ShuffedBF の問合せ高速化についての諸検討” , 第 4 回データ工学と情報マネージメントに関するフォーラム(DEIM2012), B7-4 (2012).

[3] R.Brinkman, L.Feng, J.Doumen, P.H.Hartel and W.Jonker:“Efficient Tree Search in Encrypted Data”, Information Security Journal: A Global Perspective,Volume 13, Issue 3, pp.14–21 (2004).

[4] D.X.Song, D.Wagner and A.Perrig:“Practical techniques for searches on encrypted data”, IEEE Symposium on Security and Privacy, pp.44–55 (2000).

[5] Yu Xu, Yannis Papakonstantinou:“Efficient Keyword Search for Smallest LCAs in XML Databases”, SIGMOD '05 Proceedings of the 2005 ACM SIGMOD international conference on Management of data, pp.527-538 (2005).

[6] Y. Yang, W. Ng, H. L. Lau and J. Cheng:“An efficient approach to support querying secure outsourced XML information”, CAiSE'06 Proceedings of the 18th international conference on Advanced Information Systems Engineering, pp.157–171 (2006).

[7] M. Schrefl, K. Grun and J. Dorn:“SemCrypt – Ensuring Privacy of Electronic Documents Through Semantic-Based Encrypted Query Processing”, Data Engineering Workshops, 2005. 21st International Conference on Data Engineering, pp.1191 (2005).

[8] J.Lee and K.Whang:“Secure query processing against encrypted XML data using Query-Aware Decryption”, Information Sciences, 176(13), pp.1928-1947(2006).

[9] R.Jammalamadaka and S.Mehrotra:“Querying Encrypted XML Documents”, Database Engineering and Applications Symposium, IDEAS '06, pp.129-136(2006).

[10] H.Wang and L.V.S.Lakshmanan:“Efficient Secure Query Evaluation over Encrypted XML Databases”, In Proc. of the 32nd International Conference on Very Large Data Bases (VLDB'06), pp.127-138 (2006).

[11] Watanabe C. and Arai Y.:“Privacy-Preserving Queries for a DAS model using Two-Phase Encrypted Bloomfilter”, Database Systems for Advanced Applications, pp.491-495 (2009).

[12] Yan-Cheng, Chang, and Michael Mitzenmacher:“Privacy preserving keyword searches on remote encrypted data”, Applied Cryptography and Network Security, pp.442-455 (2005).

[13] Liu, Qin, Guojun Wang, and Jie Wu:“Secure and privacy preserving keyword searching for cloud storage services”, Journal of network and computer applications 35.3 pp.927-933 (2012).

[14] Bertino, Elisa, and Elena Ferrari:“Secure and selective dissemination of XML documents”, ACM Transactions on Information and System Security (TISSEC) 5.3 pp.290-331(2002).

[15] Chang, Tao-Ku, and Gwan-Hwan Hwang:“Developing an efficient query system for encrypted XML documents”, Journal of Systems and Software 84.8 pp.1292-1305 (2011).

[16] Xiaodong Li, Weidong Yang, Jiangang Zhu, Hao Zhu:“SXKSearch: A System of XML Keyword Search Based on Secure Access Control”, Proceedings of the 2010 International Conference on Internet Computing, pp.77-82(2010).

[17] Ozan Ünay, Taflan I. Gündem:“A survey on querying encrypted XML documents for databases as a service”, ACM SIGMOD Record 37.1 pp.12-20 (2008).

[18] “Extensible Markup Language (XML) 1.1 (Second Edition)”,

http://www.w3.org/TR/xml11/.

[19] Masatoshi Yoshikawa, Toshiyuki Amagasa, Takeyuki Shimura and Shunsuke Uemura:“XRel: A path-based approach to storage and retrieval of XML documents using relational databases”, ACM Transactions on Internet Technology, Vol. 1, No. 1, pp. 110–141 (2001).

[20] “XML Path Language (XPath) 3.0”, http://www.w3.org/TR/xpath-30/.

[21] “XQuery 3.0: An XML Query Language”, http://www.w3.org/TR/xquery-30/.

[22] Burton H. Bloom:“Space/time trade-offs in hash coding with allowable errors”, Commun. ACM 13, 7, pp.422-426 (1970)

[23] S. Al-Khalifa, H.V. Jagadish, N. Koudas, J.M. Patel, D. Srivastava, and Yuqing Wu:“Structural joins: a primitive for efficient XML query pattern matching”, Proceedings 18th International Conference on Data Engineering, pp. 141–152 (2002).

[24] “XML::XPath - search.cpan.org”, http://search.cpan.org/~msergeant/XML-XPath-1.13/.

[25] “XMark - An XML Benchmark Project”, http://www.xml-benchmark.org/.

[26] C. Sun, C.Y. Chan, and A.K. Goenka:“Multiway slca-based keyword search in xml data”, Proc. of WWW, pp.1043-1052 (2007).

[27] W. Wang, X. Wang and A.Zhou:“Hash-Search: An Efficient SLCA-Based Keyword Search Algorithm on XML Documents”, Proc. of DASFAA, LNCS 5463, pp.496-510 (2009).

その他文献

国内学会（査読あり）

1. 中村伸一, 山本博章: “XPath を用いた暗号化 XML 文書検索手法の提案”, 日本データベース学会論文誌, Vol 11, No.2, pp.31-36 (2012).
2. 中村伸一, 山本博章: “Bloom Filter を利用した暗号化 SLCA 検索手法の提案”, 日本データベース学会論文誌, Vol.12, No.2, pp.1-6 (2013).

国内学会（査読なし）

1. 中村伸一, 山本博章：“XML データの暗号化に対応した安全な検索方法の提案”, 情報科学技術フォーラム講演論文集, pp.43-44 (2008).
2. 中村 伸一, 山本 博章：“XML データの暗号化に対応した安全な木パターン照合方法の提案”, 電子情報通信学会総合大会講演論文集, D-4-1 (2011).
3. Shin-ichi Nakamura, Hiroaki Yamamoto：“Searching an Encrypted XML Document Using Keywords”, 電子情報通信学会総合大会講演論文集, D-4-1 (2013).

謝辞

本論文を作成するにあたり，指導教官の山本博章教授から，丁寧かつ熱心なご指導を賜りました．ここに感謝の意を表します．